Proceedings of the First Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop

JKAW '90

Edited by
H. Motoda
R. Mizoguchi
J. Boose
B. Gaines

Oct. 25 (Thu.) — 26 (Fri.) 1990
Kyoto International Conference Hall,
Kyoto
Oct. 29 (Mon.) — 31 (Wed.) 1990
Advanced Research Laboratory, Hitachi, Ltd.,
Hatoyama, Saitama

Proceedings of the First Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop JKAW '90

Oct. 25 (Thu.) — 26 (Fri.) 1990
Kyoto International Conference Hall,
Kyoto

Oct. 29 (Mon.) — 31 (Wed.) 1990
Advanced Research Laboratory, Hitachi, Ltd.,
Hatoyama, Saitama

Co-Sponsored by
Advanced Research Laboratory, Hitachi, Ltd.
Kansai Institute of Information Systems

In Cooperation with
American Association for Artificial Intelligence
Information Processing Society of Japan
Japanese Society for Artificial Intelligence
Japan Society for Software Science and Technology
The Institute of Electronics, Information and
Communication Engineers

PREFACE

The knowledge acquisition bottleneck has become the major impediment to the development and application of effective knowledge-based systems. Many research groups around the world have been working on knowledge acquisition methodologies, techniques and tools to overcome this problem. Knowledge acquisition research, however, is still in its early stages and there are many fundamental problems to be solved, new perspectives to be generated, tools to be developed, refined and disseminated, and so on - the work seems endless. Like many modern technologies, knowledge acquisition requires a large-scale cooperative international effort. It is virtually impossible for one research and development group to have world-class expertise in all the issues, technologies and experience necessary to develop integrated knowledge acquisition tools for a wide range of knowledge-based systems.

JKAW'90(The First Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop) is one of the series of workshops which took place in Banff, Canada(KAW) and Europe(EKAW), all of which encourage the sharing of results and experience across countries. JKAW'90 consists of two parts: one is an open workshop held at Kyoto International Conference Hall(Kyoto) in October 25 and 26, 1990, in which two tutorials and eight invited talks were given. The other is a closed workshop held at Advanced Research Laboratory, Hitachi Ltd.(Saitama) during October 29 and 31, 1990, in which seventeen papers from nine countries were given together with one invited talk and panel. This proceedings include papers presented at these two workshops plus two additional papers not presented there.

Expert systems in Japan have entered its second phase of practical use. Building highly advanced systems with intelligent behavior is no longer a novel idea; it is already well accepted. Two major trends exist in Japan:

one is building innovative or large scale expert systems for practical use, and the other is exploring new technologies for building expert systems to overcome difficulties in the current rule-based techniques.

Knowledge acquisition bottleneck is one of the toughest problems that needs to be overcome for the next generation expert systems to be really useful and practical. Better methodologies for building expert systems are badly needed to reduce the developmental cost. Recent rapid increase of interest in the research on knowledge acquisition in Japan reflects this need.

JKAW'90 was held under this situation. It is extremely timely and instructive. Tutorials and invited talks in the open workshop were valuable for both researchers and engineers to get acquainted with the current state of the art of knowledge acquisition. Papers presented at the closed workshop and discussions there were also exciting and stimulating.

We believe that JKAW'90 made a considerable impact on the future research activities of people involved in both expert system development and fundamental research in Japan and also made a contribution to the promotion of mutual understanding among researchers in Japan and other countries. Finally, on behalf of the program committee, we would like to express our sincere thanks to all the invited speakers for their instructive talks and to KIIS and ARL, Hitachi for their co-sponsorship of this workshop.

October, 1990

Hiroshi Motoda
Riichiro Mizoguchi
John Boose
Brian Gaines
Workshop Co-Chairmen

ORGANIZING COMMITTEE

Workshop Co-chairmen

John Boose	Boeing Computer Services, USA
Brian Gaines	University of Calgary, Canada
Hiroshi Motoda	Hitachi, Ltd., Japan
Riichiro Mizoguchi	Osaka University, Japan

International Program Committee

Tom Addis	University of Reading, UK
Guy Boy	Centre d'Etudes et de Recherche de Toulouse, France & NASA AMES, USA
Jeffrey Bradshaw	Boeing Computer Services, USA
B. Chandrasekaran	Ohio State University, USA
William Clancey	Institute for Research on Learning, USA
Jean-Gabriel Ganascia	University Pierre et Marie Curie, France
Thomas Gruber	Stanford University, USA
Koichi Hori	University of Tokyo, Japan
Nancy Johnson	Brunel University, UK
Georg Klinker	Digital Equipment Corporation, USA
Shigenobu Kobayashi	Tokyo Institute of Technology, Japan
Yves Kodratoff	CNRS, France & George Mason University, USA
Marc Linster	GMD, Germany
John McDermott	Digital Equipment Corporation, USA
Ryszard Michalski	George Mason University, USA
Katharina Morik	GMD, Germany
Toyoaki Nishida	Kyoto University, Japan
Mark Musen	Stanford University, USA
Bruce Porter	University of Texas at Austin, USA
Ross Quinlan	Sydney University, Australia
Alain Rappaport	Neuron Data, USA
Mildred Shaw	University of Calgary, Canada
Hirokazu Taki	Institute for New Generation Computer Technology, Japan
Masanobu Watanabe	NEC Corporation, Japan
Bob Wielinga	University of Amsterdam, Holland

AUTHORS INDEX

Barfoursh, A.A. ····· 363
Bergadano, F. ····· 301
Boose, J.H. ····· 25
Clancey, W.J. ····· 65
Cummins, F. ····· 235
Cunningham, P. ····· 235
Dallemagne, G. ····· 134
Fujisawa, H. ····· 235
Gaines, B.R. ····· 3,413
Giordana, A. ····· 301
Gruber, T. ····· 148
Hagiwara, K. ····· 207
Hederman, L. ····· 235
Hofmann, M. ····· 249
Hori, K. ····· 289
Kawaguchi, A. ····· 81
Kitahashi, T. ····· 331
Klinker, G. ····· 134
Kobayashi, S. ····· 113,261
Langendorfer, H. ····· 249
Leung, K.S. ····· 161
Marques, D. ····· 134
Matsuda, K. ····· 277
McDermott, J. ····· 134
Medow, M. ····· 191
Mizoguchi, R. ····· 81,113,277,331
Mo, D.H. ····· 431
Motoda, H. ····· 81,113,345
Nakamura, K. ····· 261
Nishida, S. ····· 175
Nomura, Y. ····· 277
Ohsuga, S. ····· 289
Probst, A.R. ····· 382
Quinlan, J.R. ····· 97
Rogers, M.H. ····· 363
Saitta, L. ····· 301
Schreiweis, U. ····· 249
Shaw, M.L.G. ····· 3
Spirgi, S. ····· 382
Tajima, M. ····· 398
Takegaki, M. ····· 175
Taki, H. ····· 315
Terano, T. ····· 113
Terasaki, S. ····· 315
Tijerino, Y.A. ····· 331
Travis, L. ····· 191
Tsujino, K. ····· 175
Tung, D. ····· 134
Webb, G.I. ····· 219
Wenger, D. ····· 382
Wong, M.L. ····· 161
Yoshida, K. ····· 345

PROGRAM OF JKAW'90

At Kyoto International Conference Hall (open)

Thursday, October 25

9:00- 9:30 Opening Session *by KIIS and B. Gaines*

TUTORIAL

9:30-10:45 Foundations of Knowledge Acquisition *by B. Gaines and M. Shaw*

10:45-12:00 Knowledge Acquisition Tools, Methods, and Mediating Representations *by J. Boose*

12:00-13:30 Lunch

INVITED TALK

13:30-14:45 Real-Time Problem Solving and Knowledge Acquisition *by B. Chandrasekaran*

14:45-15:15 Coffee Break

15:15-16:30 Implications of the System-Model-Operator Metaphor for Knowledge Acquisition *by W. Clancey*

16:30-17:45 An Architecture of Knowledge Acquisition by Interview Based on Dynamic Analysis *by A. Kawaguchi, H. Motoda and R. Mizoguchi*

18:00-20:00 Banquet

Friday, October 26

INVITED TALK

9:00-10:15 Inductive Knowledge Acquisition from Structured Data *by J. Quinlan*

10:15-11:30 Research Activities of Knowledge Acquisition and Learning in Japan *by S. Kobayashi, T. Terano, H. Motoda and R. Mizoguchi*

11:30-13:00 Lunch

13:00-14:15 Explorations in How to Make Application Pro-

	gramming Easier *by J. McDermott, G. Dallemagne, G. Klinker, D. Marques and D. Tung*
14:15-15:30	Justification-Based Knowledge Acquisition *by T. Gruber*
15:30-16:00	Coffee Break
16:00-17:15	Integration of Knowledge Acquisition and Performance Systems *by A. Rappaport*
17:15-17:45	Closing Session *by J. Boose*

Monday, October 29

8:00	Bus to ARL from Hotel
9:00- 9:40	Opening *by H. Motoda and J. Boose*
	RULE INDUCTION 1
9:40-10:20	AKARS-1: An Automatic Knowledge Acquisition and Refinement System *by K. Leung and M. Wong*
10:20-11:00	A Knowledge Acquisition System that Aims at Integrating Inductive Learning and Explanation-Based Reasoning *by K. Tsujino, M. Takegaki and S. Nishida*
11:00-11:20	Break
11:20-12:00	Novice: Getting Textbook Knowledge into Expert Systems *by M. Medow and L. Travis*
	INVITED TALK
12:00-13:00	Tasks, Methods and Knowledge *by B. Chandrasekaran*
13:00-14:00	Lunch
	RULE INDUCTION 2
14:00-14:40	A New Approach to Classification Knowledge Refinement *by K. Hagiwara*
14:40-15:20	Rule Optimization and Theory Optimization: Heuristic Search Strategies for Data Driven Machine Learning *by G. Webb*
15:20-15:40	Break

HYPERTEXT

15:40-16:20 A Combined Approach to Text Retrieval Using Concept Networks Linked to Hypertext *by P. Cunningham, H. Fujisawa, L. Hederman and F. Cummins*

16:20-17:00 Knowledge Acquisition with a Special Hypertext System *by H. Langendorfer, U. Schreiweis and M. Hofmann*

17:00-18:00 Demo

18:20- Bus to Hotel

19:30- Dinner

Tuesday, October 30

8:20 Bus to ARL

EBL/CBR

9:20-10:00 Knowledge Acquisition from Machine Adjustment Cases Using Causal Model and Operationality Criteria *by K. Nakamura and S. Kobayashi*

10:00-10:40 ISAK: Interview System for Acquiring Design Knowledge -A New Architecture of Interview Systems Using Examples- *by R. Mizoguchi, K. Matsuda and Y. Nomura*

10:40-11:00 Break

INTERVIEW 1

11:00-11:40 Assisting the Articulation of the Mental World *by K. Hori and S. Ohsuga*

11:40-12:20 Automated Versus Manual Knowledge Acquisition: A Comparison in a Real Domain *by F. Bergadano, A. Giordana and L. Saitta*

12:20-13:00 A Proposal Guided Knowledge Acquisition Support System *by H. Taki and S. Terasaki*

13:00-14:00 Lunch

INTERVIEW 2

14:00-14:40 A Task Analysis Interview System that Uses a Problem-Solving Model *by Y. Tijerino, T. Kitahashi and R. Mizoguchi*

14:40-15:20	Hierarchical Knowledge Representation Based on Approximations *by K. Yoshida and H. Motoda*
15:20-15:40	Break
15:40-17:20	Panel or Position Statements *by Invited Speakers*
17:20-18:00	Demo
17:20-	Bus to ARL(for Accompanying Person)
18:30-	Banquet at ARL's Guest House
21:00-	Bus to Hotel

Wednesday, October 31

8:20	Bus to ARL

MISCELLANEOUS

9:20-10:00	A Knowledge Acquisition Method for Distributed Information Systems: A Multi-DataBase Approach *by A. Barfoursh and M. Rogers*
10:00-10:40	Knowledge Acquisition in a Development Methodology for Knowledge-Based Applications *by S. Spirgi, A. Probst and D. Wenger*
10:40-11:00	Break
11:00-11:40	A Framework of Learning System RLS *by M. Tajima*
11:40-12:20	Demo
12:20-13:00	Summary and Concluding Remarks *by J. Boose*
13:00-15:00	Lunch and Farewell Beer
15:30	Bus to Hotel

TABLE OF CONTENTS

TUTORIAL

Foundations of Knowledge Acquisition
B. Gaines and M. Shaw 3

Knowledge Acquisition Tools, Methods, and Mediating Representations
J. Boose 25

INVITED TALK

Implications of the System-Model-Operator Metaphor for Knowledge Acquisition
W. Clancey 65

An Architecture of Knowledge Acquisition by Interview Based on Dynamic Analysis
A. Kawaguchi, H. Motoda and R. Mizoguchi 81

Inductive Knowledge Acquisition from Structured Data
J. Quinlan 97

Research Activities of Knowledge Acquisition and Learning in Japan
S. Kobayashi, T. Terano, H. Motoda and R. Mizoguchi 113

Explorations in How to Make Application Programming Easier
J. McDermott, G. Dallemagne, G. Klinker, D. Marques and D. Tung 134

Justification-Based Knowledge Acquisition
T. Gruber 148

RULE INDUCTION

AKARS-1: An Automatic Knowledge Acquisition and Refinement System
K. Leung and M. Wong 161

A Knowledge Acquisition System that Aims at Integrating Inductive Learning and Explanation-Based Reasoning
K. Tsujino, M. Takegaki and S. Nishida ····175
Novice: Getting Textbook Knowledge into Expert Systems
M. Medow and L. Travis ····191
A New Approach to Classification Knowledge Refinement
K. Hagiwara ····207
Rule Optimization and Theory Optimization: Heuristic Search Strategies for Data Driven Machine Learning
G. Webb ····219

HYPERTEXT

A Combined Approach to Text Retrieval Using Concept Networks Linked to Hypertext
P. Cunningham, H. Fujisawa, L. Hederman and F. Cummins ····235
Knowledge Acquisition with a Special Hypertext System
H. Langendorfer, U. Schreiweis and M. Hofmann ····249

EBL/CBR

Knowledge Acquisition from Machine Adjustment Cases Using Causal Model and Operationality Criteria
K. Nakamura and S. Kobayashi ····261
ISAK: Interview System for Acquiring Design Knowledge -A New Architecture of Interview Systems Using Examples-
R. Mizoguchi, K. Matsuda and Y. Nomura ····277

INTERVIEW

Assisting the Articulation of the Mental World
K. Hori and S. Ohsuga ····289
Automated Versus Manual Knowledge Acquisition: A Comparison in a Real Domain
F. Bergadano, A. Giordana and L. Saitta ····301
A Proposal Guided Knowledge Acquisition Support System
H. Taki and S. Terasaki ····315

A Task Analysis Interview System that Uses a Problem-Solving Model
Y. Tijerino, T. Kitahashi and R. Mizoguchi ····331
Hierarchical Knowledge Representation Based on Approximations
K. Yoshida and H. Motoda ····345

MISCELLANEOUS

A Knowledge Acquisition Method for Distributed Information Systems: A Multi-DataBase Approach
A. Barfoursh and M. Rogers ····363
Knowledge Acquisition in a Development Methodology for Knowledge-Based Applications
S. Spirgi, A. Probst and D. Wenger ····382
A Framework of Learning System RLS
M. Tajima ····398

Papers not presented at the Workshop

Knowledge Representation Servers: A Generic Technology for Knowledge Acquisition and Knowledge-Based Systems
B. Gaines ····413
Acquiring Text Editing Programs from Examples
D. Mo ····431

FOUNDATIONS OF KNOWLEDGE ACQUISITION

Brian R Gaines, Killam Memorial Chair, & Director, Knowledge Science Institute

Mildred L G Shaw, Professor, Department of Computer Science, and KSI

University of Calgary, Alberta, Canada T2N 1N4

gaines@cpsc.ucalgary.ca

Knowledge acquisition is recognized as a major bottleneck in the development of knowledge-based systems. The first generation of knowledge acquisition techniques in the 1970's were based on human interviewing. The second generation in the 1980's have been based on computer-aided application of specific techniques such as repertory grids, behavior modeling and text analysis. Current third generation research is targeting tools for the 1990's to provide integrated knowledge acquisition environments supporting a wide range of complementary tools and techniques. This paper surveys the foundations of knowledge acquisition, providing a framework for the wide range of techniques in use.

1 INTRODUCTION

At the time of the first AAAI Workshop on Knowledge Acquisition for Knowledge Based Systems in Banff, November 1986, the literature on knowledge acquisition was sparse, scattered in conference publications, and difficult to access. The 1986 Workshop, and the ensuing journal and book publication of some forty papers representing the state of the art world-wide, changed that situation [7, 22]. The inclusion of sessions on knowledge acquisition in many conferences, the European Workshops on Knowledge Acquisition for Knowledge Based Systems in London, Bonn, Paris and Amsterdam, and the later AAAI Banff Workshops, have changed the situation even further. We now have a world wide network of communicating researchers aware of one another's activities and an open, widely disseminated and readily available literature on knowledge acquisition.

The problem now is not so much to access research and experience in knowledge acquisition but to make sense of the wide ranging material available and, in particular, to apply the results to improve effectiveness in the development of knowledge-based systems. There are a number of impediments to such understanding and application:

- A diversity of techniques and tools that overlap in their applications—where it is not clear whether they are competitive alternatives or complementary partners;
- Lack of variety, detail and evaluation in the case histories of applications of the techniques and tools;
- Lack of access to the techniques and tools outside the narrow research communities originating them;

5. AN EXAMPLE

5.1 Experimental problem

In this section, we will discuss KAISER by illustrating a simplified example of a famous computer game 'TETRIS'. Figure 3 shows one scene of TETRIS. A brick gradually drops down from the top of the screen. We can move it right and left, rotate it and make it drop to the bottom. When a row is filled up with bricks, the row vanishes away and the upper rows slide down. The objective of this game is to pile up and crash as many bricks as possible until the peak of stack reaches the top of the screen.

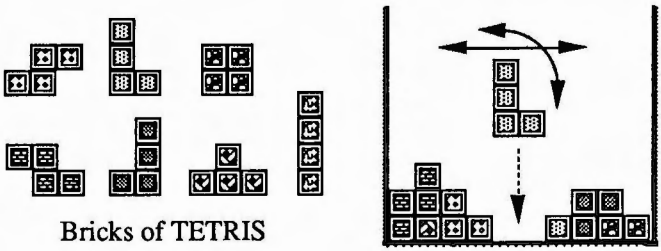


Figure 3. One scene of the target game TETRIS.

Although this is a kind of planning problem, it can be handled as a classification problem that classifies a new brick to a adequate place to settle according to the pattern of piled bricks. To make the problem simple, we adopt a simplified version of TETRIS that has the following restrictions: (1) The width of the screen is four columns to make the problem space small; (2) A brick does not come down but stays always at the top of the screen; (3) So we can't bring a brick under an overhang to fill the hollow; and (4) The bricks of different shapes are dealt separately. Figure 4 shows a training instance of a J-shaped brick. An instance is described by six attributes, i.e., D12, D13, D14, D23, D24 and D34, where D12 represents the difference of height between the first and the second column from the left. Each attribute has one of seven values, i.e., S, -2, -1, 0, 1, 2 or M, where -2 means that the difference is two steps, S and M respectively means that the difference is less than or more than two steps. These six-dimensional vectors are classified into ten classes, i.e., FU3 (Face-Up at the third column, i.e., the flat side down and the right edge at the third column), FU4, FD3 (Face-Down at the third), FD4, ST2 (STraight at the second), ST3, ST4, HS2 (HandStand at the second), HS3 or HS4. Sixty-three examples of J-shaped brick are prepared out of 352 trials that consist of all types of brick.

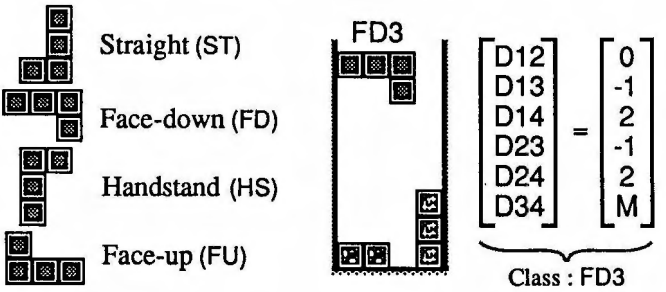


Figure 4. An example of J-shaped brick represented in OAV-form.

5.2 Domain knowledge

Table 1 shows the domain knowledge used for this problem. For example, Econd of FD3-1, (D12 0)(D23 -1), means that if D12 = 0 and D23 = -1, i.e., if the heights of the first and the second column are same, and the third column is one step lower than the second, then we can put the J-brick with face-down at the third column. The Dcond (D34 1 2 M) means that if the fourth column is higher than the third column, it's more desirable for FD3. The Pcond (D23 -2 S) means that even if the third column is two or more steps below to the second, we may put the brick at FD3. In addition to the class-attribute relationships, inter-attribute and ordinal relationships are also defined for the TETRIS problem. For example, D13 attribute is defined as (D13 (calc-D-from-DD D12 D23)). It denotes that the first element D13 can be derived from D12 and D23 by applying the function calc-D-from-DD to these known attributes. Ordinal relationships are also defined as (D12 -1 -2 0).

Table 1. Domain knowledge for TETRIS.

	Econd	Dcond	Pcond
ST2	(D12 0)	(D23 1 2 M)	()
ST3	(D23 0)	(D12 -1 -2 S) (D34 1 2 M)	()
ST4	(D34 0)	(D23 -1 -2 S)	()
FD3-1	(D12 0) (D23 -1)	(D34 1 2 M)	(D23 -2 S)
FD3-2	(D34 2)		
	(D13 -1 0 1 2 M)	(D13 -1) (D23 -1)	(D34 M)
	(D23 -1 0 1 2 M)		
FD4-1	(D23 0) (D34 -1)	(D12 -1 -2 S)	(D34 -2 S)
FD4-2	(D14 -2)		
	(D24 -1 0 1 2 M)	(D24 -1) (D34 -1)	(D14 S)
	(D34 -1 0 1 2 M)		
HS2	(D12 2)	(D23 1 2 M)	()
HS3	(D23 2)	(D12 -1 -2 S) (D34 1 2 M)	()
HS4	(D34 2)	(D23 -1 -2 S)	()
FU3-1	(D12 0) (D23 0)	(D34 1 2 M)	()
FU3-2	(D14 1)		
	(D13 0 -1 -2 S)	(D13 0) (D12 0)	(D14 2 M)
	(D12 0 -1 -2 S)		
FU4-1	(D23 0) (D34 0)	(D12 -1 -2 S)	()
FU4-2	(D12 -1)		
	(D23 0 -1 -2 S)	(D23 0) (D24 0)	(D12 -2 S)
	(D24 0 -1 -2 S)		

5.3 Experimental results

Figure 5 shows a part of a decision tree for J-shaped bricks. The labels depicted under the leaves are the results of TBM. By checking the differences between the induced labels and the TBM results, i.e., support factors, KAISER recommends to:

(1) modify the class of leaf 50 and 51 to FD4 according to the two support factors, FD4-1 and FD4-2;
(2) choose proper classes for leaf 53, 54 and 56 out of their support factors; and
(3) ignore the result of leaf55 and choose the class out of the support factors, if there is few examples in the leaf, otherwise generalize the Econd in FD3 rules.

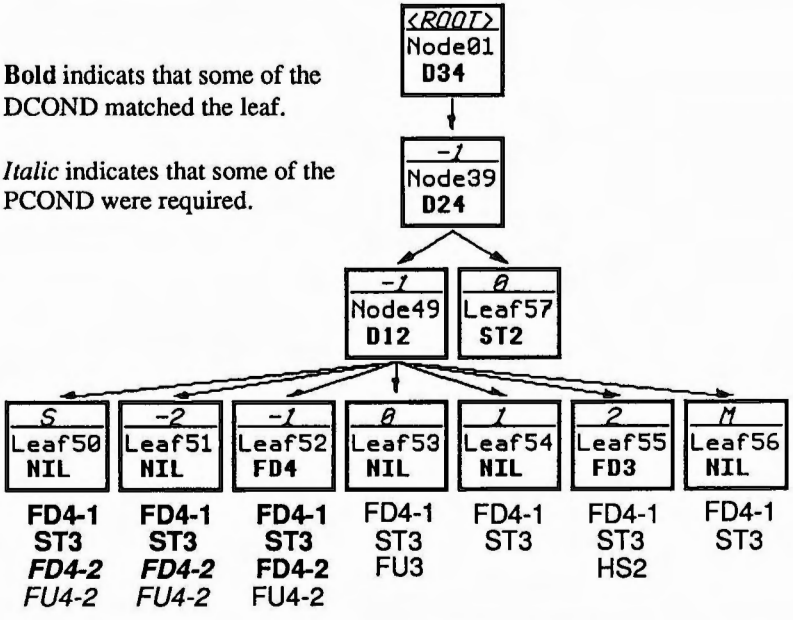


Figure 5. Some examples of TBM support factors.

In this figure, we can also find that the leaf 57 could not get any support by domain knowledge although the class is specified by induction. This is the clue for RBM. Based on the domain knowledge of ST2 (shown in Table 1), we can know that the path to leaf 57 does not have a condition about D12, which is necessary (Econd) for ST2. In this case, KAISER marks leaf 57 as an insufficient leaf that may lack the condition about D12, and propose additional conditions as shown in Figure 6. TBM is examined for the generated paths to the new leaves from 101 to 107 and the support factors are calculated for them.

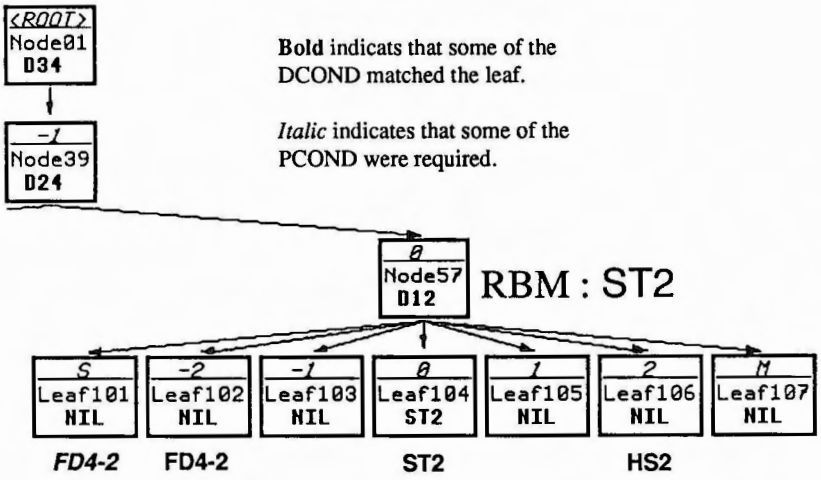


Figure 6. Additional conditions proposed by the RBM result at node 57.

5.4 Implementation

KAISER has been implemented on Symbolics Lisp machine 3650 with Common-Lisp and OPS5 [15]. Impropriety detection and interpretation are performed by OPS5, which makes it easy to describe and modify improprieties. Figure 8 shows a snapshot of KAISER.

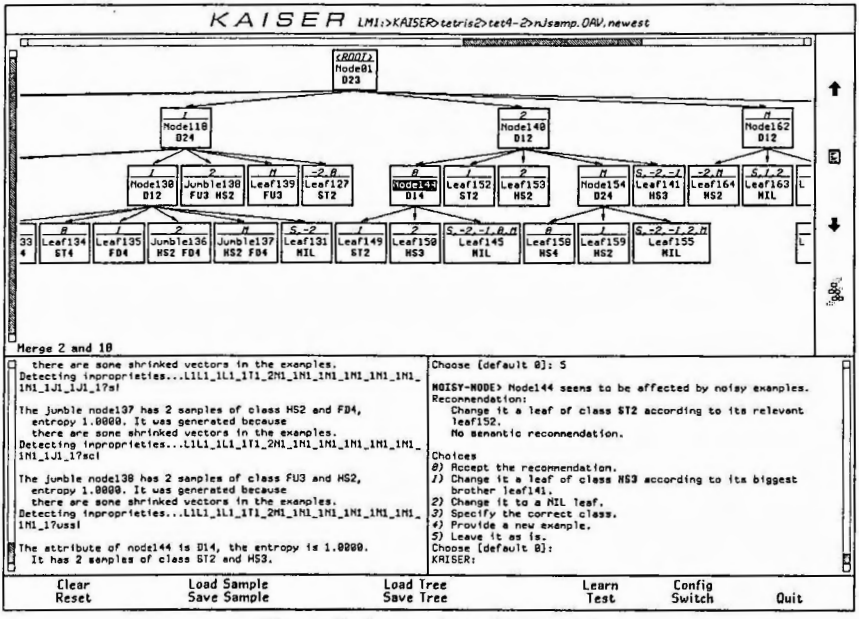


Figure 8. A snapshot of KAISER.

The upper panel shows a decision tree under examination, and the lower left panel displays some useful information about the tree and improprieties. The questionnaires are printed on the lower right panel.

6. CONCLUSION

We have discussed an inductive and interactive knowledge acquisition system named KAISER. This system aims at integrating similarity-based inductive learning of classification knowledge and explanation-based knowledge evaluation to realize sophisticated knowledge acquisition and reliable knowledge formalization. Since KAISER is still in an initial stage of research, there are many pending issues, however, the proposed impropriety-based approach turned out to be efficient for conducting intelligent interview as well as promoting inductive learning.

Acknowledgement

Some of the basic ideas of this work benefit from discussion with Prof. Riichiro Mizoguchi and other fellows at I.S.I.R. Osaka university while one of the author, Katsuhiko Tsujino, was a doctoral student of the laboratory.

References

[1] Utgoff, P.E., *Machine Learning of Inductive Bias*, Kluwer, Boston (1986).

[2] Tsujino, K., Takenouchi, S., Sakurai, T., Chigusa, S., Nomura, Y., Mizoguchi, R. and Kakusho, O., "Adaptive Rule Induction System: ARIS," (in Japanese) *Trans. of IEICE*, J72-D2, pp.121-31 (1989).

[3] Quinlan, J.R., "Induction of Decision Trees," *Machine Learning*, 1, pp.81-106 (1986).

[4] Mitchell, T.M., Keller, R.M. and Kedar-Cabelli, S.T., "Explanation-Based Generalization: A Unifying View," *Machine Learning*, 1, pp.47-80 (1986).

[5] McDermott, J., "Preliminary Steps Towards a Taxonomy of Problem-Solving Methods," *Automating Knowledge Acquisition for Expert Systems*, Marcus,S. (Ed.), Kluwer, Boston (1988).

[6] Lebowitz,M.,"Integrated Learning: Controlling Explanation," *Cognitive Science*, 10, pp.219-40 (1986).

[7] Pazzani, M.J., "Integrated Learning with Incorrect and Incomplete Theories," *Proc. of the 5th International Conference on Machine Learning*, pp.291-297 (1988).

[8] Mooney,R. and Ourston, D., "Induction Over the Unexplained: Integrated learning of concepts with both explainable and conventional aspects," *Proc. of the 6th International Workshop on Machine Learning*, pp.5-7 (1989).
[9] Kahn, G., Nowlan, S. and McDermott, J., "MORE: An Intelligent Knowledge Acquisition Tool," *Proc. of IJCAI'85*, pp.581-84 (1985).
[10] Eshelman, L. and McDermott, J., "MOLE: A Knowledge Acquisition Tool that Uses Its Head," *Proc. of AAAI'86*, pp. 950-55 (1986).
[11] Marcus, S., "Knowledge Acquisition for Constructive Systems," *Proc. of IJCAI'85*, pp.637-39 (1985).
[12] Boose, J.H., "Personal Construct Theory and the Transfer of Human Expertise," *Proc. of AAAI'84*, pp.27-33 (1984).
[13] Boose, J.H. and Bradshaw, J.M., "Expertise Transfer and Complex Problems: Using AQUINAS as a Knowledge Acquisition Workbench for Knowledge-Based Systems," *Man-Machine Studies*, 26, pp.3-28 (1987).
[14] Kelly, G.A., *The Psychology of Personal Constructs*, Norton, New York (1955).
[15] Forgy, C.L., "OPS5 user's manual," *CMU tech. rep.*, CMU-CS-81-135 (1981).

Novice: Getting Textbook Knowledge into Expert Systems

Mitchell Medow
Larry Travis

Department of Computer Sciences and the Medical School[1]
University of Wisconsin-Madison
Madison, WI 53706

Abstract

Hamatmid is a developmental learning system for heuristic classification. Its development, patterned after the human development of expertise, begins with the didactic material used to train human novices, continues with supervised learning while solving easy problems, proceeds to unsupervised learning and finally tries to compile its knowledge. Its staging of development is motivated by the human development of expertise, but Hamatmid is not a psychological model. Novice, the first stage of Hamatmid, develops a tangled web of solution classes based on background knowledge and then extracts problem solving rules to traverse this network. It uses similarity based learning and conceptual clustering techniques with the prototypical cases described in textbooks. The problem solving rules are too general, lack some of the expert's knowledge and are dependent on the network of solution classes for context. Yet the rules' applications should be critiquable by an expert and, as such, prepare the system for learning through problem solving.

Most research in knowledge acquisition ignores the developmental history of a human's expertise. Divining for an expert's knowledge without taking this historical context into account may complicate the knowledge acquisition process. The developmental history of expertise is important because explanation (though not necessarily reasoning) is based on the didactic material that is the starting point on the path to expertise. Further, the developmental path provides additional constraints on the expert's knowledge organization.

Experts have difficulty expressing the knowledge they use for problem solving without being provided with some context or structure for this description. Solving a particular case (*e.g.*, [16]), personal construct theory [6], describing the process [35] or using the vocabulary and primitives of the problem solving system [4] can often provide sufficient context. None of these methods is necessarily tied to the domain's training material. Attempts to learn from the raw data of a domain (*e.g.*, cases) have also proven difficult. Background knowledge has been used to bias the inductive generalizations and the selection criteria for the generated generalizations [29]. Even with these biases the generated knowledge structures might not reflect the expert's structures. Knowledge acquisition remains a major bottleneck.

A new approach, as exemplified by Hamatmid, is to pattern the development of machine expertise after the pattern of human development of expertise. This method simplifies learning by limiting the cases considered (they are prototypes representing whole classes instead of individual cases to be generalized) and by biasing the vocabulary (with the domain's background knowledge and by the pruning heuristics). It enhances the efficacy of its explanations by basing its knowledge on the material taught all new initiates in the domain. Further learning occurs, as in humans, through solving cases of various degrees of difficulty and developing an "intuitive" case recognition style. The method is still time consuming, although the time commitment of the expert knowledge source is reduced. By considering the expert's development instead of focusing on the finished product, important clues to the expert's data structures are retained; by using the domain's training materials, learned knowledge structures should be similar to an expert's, and adequate explanations should be possible; and by following a path related to the human development of expertise, similar causes (*e.g.*, the assimilation of training material or types of problem solving failures) should have similar effects on the learned knowledge structures, creating a final system that is close to human expertise.

[1]The first author is a student in both the Department of Computer Sciences and the Medical School. The second author is a Professor in the Department of Computer Sciences.

Background

There are three sources of constraints for knowledge acquisition: the domain (*e.g.*, infectious diseases), the task (*i.e.*, heuristic classification) and the pattern of development in human experts (*e.g.*, medical education). Solution classes for heuristic classification [11] are often described as prototypes [42]. For instance, medical knowledge describing diseases is prototypical, described as abstract cases. This knowledge can be found in textbooks and is modeled well using current AI techniques such as frames [32]. Background knowledge provides context for prototypical concepts. It is responsible for determining what attributes are attended to and what values are considered similar [34, 27]. If the knowledge acquisition system is to learn intermediate groupings of the solution classes (*i.e.*, intermediate concepts), then background knowledge is needed to decide which similarities are relevant and what generalizations humans have developed (*e.g.*, for data abstraction). Finally, background knowledge has evolved from human experience in the domain and is tailored to the tasks humans perform, such as problem solving [40]. Background knowledge thus serves to summarize human experience in the domain. It constrains generalization, provides coherence for concepts, is used in explanation, has evolved from human experience and is directed at typical uses. Recently, similarity based learning and conceptual clustering systems have tried to exploit background information (*e.g.*, [29, 43, 33, 5]).

The goal of a diagnostic expert system is heuristic classification. Well designed expert systems require that structural, support and strategic knowledge be made explicit and kept separate from the problem solving rules [10] (at least for the epistemological engineer). This helps the system provide explanations, deal with novel situations, reason about itself and learn. While rules can represent most of this knowledge with frames to represent the solution classes, the rules need to be augmented with links to make the relationships among classes explicit (see, for example, [37] for a discussion of these issues as they relate to medical classification systems). Finally, trigger rules are needed to suggest solution classes for consideration and to force the system to shift its attention appropriately during problem solving.

A diagnostic system is modeling a domain [12], but (somewhat contrary to Clancey) this model *should* be constrained by the expert's model (it need not *model* the expert but should possess many commonalities with the expert). To be useful the system must explain itself to these experts in a manner that resonates with the expert's domain model. Otherwise statistical diagnostic systems, with their proven superiority to humans in diagnostic tasks [17], would dominate all other approaches. Experts base their knowledge initially on textbook material. Later, with experience, their knowledge structures are altered and even their reasoning style may change (although this change might be due to the underlying change in the knowledge organization and influenced by the content of a particular case). It seems reasonable to assume that: (1) these initial knowledge structures are important to the expert's development (a conclusion we implicitly endorse in our role as educators); (2) that much of the knowledge reorganization is subconscious and a response to experience in problem solving; and (3) that to be adequate, an explanation must relate to these knowledge structures (whether the organization is consciously learned or subconsciously developed). If we are to create explanations resonant with an expert's model it seems reasonable to try to follow a similar developmental path to produce a similar knowledge structure.

Human experts tend to use a perceptual and directed style of reasoning, at least in part due to the underlying knowledge representation of the expert. An expert's knowledge is organized in functional units representing the deeper structure of the domain [8, 9] with many connections among classes [20]. Reasoning proceeds in a forward directed manner — from the perceived concepts in the problem (*e.g.*, conservation of momentum), which suggest a method, to the solution. Novices arrange their knowledge by the surface structure of a problem (*e.g.*, an inclined plane) and tend to rely on less directed searches (*e.g.*, backward reasoning) in problem solving. This difference in reasoning style is not due to increased general cognitive abilities of the experts but due to differences in the underlying organization of their knowledge. There is still a debate whether these findings hold for medical diagnosis. Some studies find the hypothetico-deductive method (a "generate and test" weak method) is used by students as well as experienced clinicians (*e.g.*, [19, 3]). Others find perceptual, forward reasoning processes in experienced clinicians (*e.g.*, [25, 36]), as opposed to patterns in non-expert clinicians that involve at least some backward reasoning.

Expertise is not developed in a single monolithic training period. Instead it is developed over time, often with different, recognizable phases in training. As an example of the development of expertise let us follow a typical medical student in the pursuit of clinical expertise. First the student spends two years in the classroom learning "textbook" knowledge: facts about various diseases, scientific

theories underlying normal and pathologic processes (*i.e.*, a causal understanding of the domain) and some procedural knowledge (*e.g.*, how to take a history or how to calculate the acid base balance of a patient). Most of this textbook knowledge is either background knowledge (which for humans serves the additional purpose of mnemonic aid) or abstract descriptions of diseases and other pathologic conditions. Pointers to appropriate background knowledge are given in the disease descriptions. These descriptions are usually presented in simplified form (rare instances, unless they have another import, are often left out). This simplifies the learning process by building a scaffolding upon which to insert details later. During this period there is some, albeit little, application of this knowledge in problem solving. When such problem solving occurs it is limited to idealized, textbook cases. This first stage of learning involves collecting numerous facts useful in the domain. Later stages of learning serve to reorganize, correct and supplement these facts with the knowledge necessary for efficient problem solving.

The next two years are spent in the hospital on clinical rotations. It is here that the student learns how the knowledge memorized in the first two years is applied to clinical problem solving. Thus the student learns diagnosis and treatment based on an understanding of classroom material and driven by the problem solving environment of patient care. The student not only learns how to apply the knowledge but learns additional facts and many more procedures, as well as develops a perceptual "database" of the symptoms, signs and tests mentioned in the texts. This stage concentrates on reorganizing and supplementing the problem solving knowledge.

Next the student graduates and enters a period of training aimed at acquiring enough experience to become an independent practicing physician. Finally, the individual may set out independently, continuing to acquire the experience necessary to become an expert, compiling her knowledge to develop a perceptual, case recognition style. These last two stages are concerned with fine tuning the clinician's knowledge base.

Hamatmid[2] is a knowledge acquisition framework which parallels the above pattern. It is an outline for the development of expertise by a computer system, creating initial knowledge structures from communal sources and modifying those structures with experience. Hamatmid mimics the phases in the development of human expertise — it is not a psychological model of those processes. The system is not limited to medicine — it will apply to any domain where the knowledge is presented as prototypes and the task is classification.

Hamatmid

Hamatmid is a research system for the phased development of expertise. Like humans, Hamatmid will develop its knowledge structures originally from textbook descriptions (as transcribed into frames and background knowledge by an epistemological engineer), augmented by training and experience. Novice, the first phase of Hamatmid, is the focus of this report. Before describing the details of Novice we will outline the stages in the development of expertise as envisioned for Hamatmid. Hamatmid is a system for the development of expertise for heuristic classification tasks.

Novice takes prototypical descriptions of solution classes (*e.g.*, diseases) and background knowledge from textbooks as input (encoded by an epistemological engineer). This knowledge includes the necessary information for heuristic classification. The prototypes are the solution classes and include heuristic information (links) in their descriptions. The background knowledge encodes data abstraction and some generalizations in the classification hierarchy. Using similarity based learning and conceptual clustering (*e.g.*, [28, 43, 21]) Novice creates a network of the prototypic classes, learning higher order categories (*i.e.*, generalizations of subsets of the solution classes, intermediate concepts) using the background knowledge. Then problem solving rules for traversing the network are extracted. These rules are not "expert" rules. The inherent simplifications of the communal sources that Novice's knowledge structures are based on mean that Novice's problem solving rules will be too general, miss rare classes and special cases, and have little knowledge (albeit some) of reasoning about the data as presented in practice, as opposed to reasoning about the concepts those data imply. Novice lacks information about how data arise in the problem solving context and often how to reason about the data's validity. Most importantly these rules are critiquable. Novice also learns a special class of rules, trigger or pathognomonic rules. These data driven rules jump to particular nodes in the network based on unique features of the node (*i.e.*, pathognomonic features) or the importance of the class (*e.g.*, a heart attack). The benefit of basing this phase on didactic material is that the inherent simplifications

[2]Hamatmid is the title of a poem by Chaim Nachman Bialik about a persistent Talmud student. Miron Livny suggested the name.

make the learning process easier by removing much of the noise and focusing on relevant features. This allows a simple framework to be built up that can later be elaborated. Novice builds a critiquable, novice (as opposed to expert) problem solving system from communal sources.

Next, novice rules will be used in a problem solving and learning system. Initially simple problems found in texts will be attempted with the system's behavior being overseen by a domain teacher (who need not be an expert, but must be proficient). This work will proceed in the style of TEIRESIAS [16], problem solving with a teacher ready to correct errors based on the system's explanation of its own problem solving. This phase augments the knowledge base developed by Novice with knowledge missing from the prototypes, correcting incorrect knowledge, adding procedural knowledge necessary to work in the domain (*e.g.*, reasoning from the data as presented in problem solving situations), adding exclusionary information (*i.e.*, reasons to exclude a particular diagnosis from consideration) and beginning to select appropriate knowledge from Novice's output structures (*i.e.*, selecting the relevant hierarchical links and rules). Finally, the rules must be weaned from the network. The result of this stage should be a system that is correct for the classes it knows (it will still be missing many rare classes and many special cases) but still a fumbling, one rule at a time diagnostician. This phase should be limited to seeing a few cases in each class. It may gradually flow into the next phase when it notices fewer major additions to its knowledge base (*i.e.*, the weaned rules seem to be operating correctly).

Afterward Hamatmid must learn on its own with feedback only about the correct solution. Much of the learning here will be similar to that in PROTOS [2] where pointers are learned to link a case with others that are its natural competitors (*e.g.*, cases that are often confused with it or otherwise related to it). This is similar to how an expert clinician might develop a list of possible diagnoses to consider as a solution (*i.e.*, develop a differential diagnosis). Also, "remindings" (a datum, clusters of data or background knowledge applied to data that suggest possible classes as solutions) will begin to be compiled out of the prototypes with the application of the problem solving rules. The remindings will augment Novice's trigger rules to suggest initial diagnostic classes for the system's consideration. Another major emphasis is the further strengthening and selecting of rules and hierarchical links, since more cases will be experienced in this stage. Such selection need not only be based on strength alone (for example, [1, pp. 249–255] or [23, pp. 70–75]). Rules modified to handle special cases must be retained; data useful in problem solving rules should be kept in the prototypical descriptions. Additionally, the system can begin to collect statistics on classes and their features now that it is beginning to see many cases. These statistics can be used to aid in uncertain reasoning (along with the importance of making or missing a particular diagnosis [14, 13]) and for the refinement of its concept hierarchy (*cf* [21, 22]). This phase must gradually give way to the next, class by class, based on the system's perceived evaluation of its problem solving abilities within that class. When effort need not be placed on learning something new (*e.g.*, no new solutions had to be considered) then Hamatmid can move to the next phase of learning with a class.

Finally, the system will be left on its own to develop experience. By compiling its rules (*e.g.*, [26, 18, 44]), and relying on remindings and differential links, a more perceptual style of expertise should emerge. This will allow more efficient diagnosis and the development of more patterned forms of data acquisition and reasoning. At the same time Hamatmid will refine its network on its own — merging or splitting classes depending on experience (especially therapeutic outcomes).

Hamatmid should produce explanations that resonate with human understanding because the problem solving rules are derived from the background knowledge, abstract prototypical descriptions of solution classes and networks of their relationships through a pattern of development similar to that of humans. It also should create systems that are easier to develop (because it uses readily available sources, requires less of an expert's time and learns from its own experience), more accurate classification (since it will continually refine its knowledge base to match the distribution of cases it sees) and be better accepted by experts (since it has knowledge structures similar to their own and produces explanations acceptable to them). Finally, as the name "Hamatmid" implies, it will be a system that continually learns from its experience.

Hamatmid assumes that expertise is based on previous knowledge and experience, and that this development should follow a particular pattern. Educational material has evolved to provide a reasonable basis for the development of expertise by simplifying concepts for later elaboration and pointing out connections among concepts in the domain. If these assumptions are negated, *e.g.*, if expertise is a complete relearning of the domain and the previous knowledge does not play a direct role in the final knowledge structures, then Hamatmid is at a loss. This could be possible. For example, textbook "knowledge" may only allow one to stumble around in a domain long enough to gain, *de novo*, a clear understanding of the domain. Then, basing Hamatmid's knowledge mainly on

textbook presentation may block its development of expertise. If this is true not only is Hamatmid in trouble, but, our educational system is in need of drastic revision. Back to the days of case-by-case apprenticeship. This is an empirical question which Hamatmid may help answer. Otherwise, Hamatmid should be applicable in any domain where heuristic classification is the task and where knowledge of solution categories is initially presented as idealized cases.

Let us now look at Novice's details to see how it takes textbook knowledge, organizes it and extracts problem solving rules.

Novice

Novice, with an epistemological engineer, begins the development of machine expertise. First, the epistemological engineer manually builds into the system the necessary background knowledge and class prototypes from the communal knowledge taught to all new initiates (*e.g.*, textbooks, slide presentations and review articles). In medicine, for example, much of the disease knowledge taught novices is presented as idealized cases. These can be naturally captured as frames extracted by an epistemological engineer. One problem is that the textbook presentations may still be a little too detailed. Additional abstraction of the prototype must often be done by the epistemological engineer. Pointers to other subjects mentioned within the textbook descriptions must be followed to find relevant background knowledge. It is not always obvious what background knowledge needs to be included for later problem solving (although most of the necessary knowledge is obvious, the rest may become obvious with a few hints from an expert). As mentioned above, this knowledge includes support, structural and heuristic knowledge (*e.g.*, causal knowledge, definitions, and empirical associations among patient types and diseases). This knowledge is the starting point for Novice.

The Representation of the Prototypes

In Novice's formalism each prototype is represented as a set of *attributes*, each with different *values*. The values for the attributes are further refined by an *aspect* describing the role that the value plays for the attribute. Attributes of an object include such things as cause (*e.g.*, the organism causing the disease), properties (*e.g.*, the color of an object) and associations (*i.e.*, heuristic links to other information, *e.g.*, predisposing factors associated with a particular disease). An aspect specifies the type of information about an attribute or adds descriptive information about its values. For example, an aspect can be:

and, or: the usual logical meaning,

values: a list of values associated with the attribute,[3]

never, rare: these values never (or rarely) occur for this attribute,

always, common: these values are mandatory (or very common) for this attribute,

sequence: these values represent a sequence, or

age: the age of the patient, an example of a descriptive aspect.

Some of the aspects, *e.g.*, **values**, **never**, **always** and **sequence**, are built into Novice. Others, *e.g.*, **age**, can be created by the epistemological engineer and are handled as a list of values, just like **values**. Aspects allow comparisons along a single dimension of an attribute. There is no attempt here to find a complete set of aspects.

One domain that we have used in developing and testing Novice is the problem of representing rooms in a typical house. This example includes ten prototypes: a master bedroom, a guest bedroom, a bedroom for one child, a bedroom for two children, a full bath with a shower, a full bath with a bathtub, an half bath, a kitchen, a living room and a den (see Figure 1).

[3]There is a short hand notation used by Novice and in this paper. If there is only a single value for the attribute, then it can be listed alone. Thus `blue` is the same as `(values blue)`.

Name	Master-Bedroom	Name	Kitchen
BED	KING	SINK	T
CHAIRS	1	OVENS	((values MICROWAVE CONVECTION))
DRESSERS	T	WASH	DISH-WASHER
TV	T	TV	T
TELEPHONE	T	TABLE	T
ACTIVITIES	((values SLEEP WATCH-TV DRESS))	CHAIRS	SOME
		TELEPHONE	T
		ACTIVITIES	((values COOK WATCH-TV EAT))
Name	Guest-Bedroom	Name	Living-ROOM
BED	KING	CHAIRS	MANY
CHAIRS	2	SOFA	T
DRESSERS	T	READING	((values MAGAZINES BOOKS))
ACTIVITIES	((values SLEEP DRESS))	ACTIVITIES	((values ENTERTAIN) (never EAT) (always COMPANY))
Name	Single-Child-Bedroom	Name	Den
BED	TWIN	CHAIRS	4
CHAIRS	1	SOFA	T
DESK	1	READING	((values MAGAZINES BOOKS NEWSPAPERS))
DRESSERS	T	TV	T
ACTIVITIES	((VALUES SLEEP DRESS PLAY))	ACTIVITIES	((values WATCH-TV PLAY) (rare SLEEP) (common RELAX))
Name	Two-Children-Bedroom		
BED	BUNK		
CHAIRS	2		
DESK	2		
DRESSERS	T		
ACTIVITIES	((VALUES SLEEP DRESS PLAY))		
Name	Full-Bath-Shower	Name	Full-Bath-Bath
SINK	T	SINK	T
TOILET	T	TOILET	T
READING	((values MAGAZINES))	READING	((values MAGAZINES))
WASH	SHOWER	WASH	BATH-TUB
ACTIVITIES	((values WASH DEFECATE))	ACTIVITIES	((values WASH DEFECATE))
Name	Half-Bath		
SINK	T		
TOILET	T		
READING	((values MAGAZINES BOOKS))		
WASH	((never WASH-HUMAN))		
ACTIVITIES	((values WASH DEFECATE))		

Figure 1: Prototypes for the House Example

The Representation of Background Knowledge

Background knowledge includes definitions of terms used in the domain, associations among related values (causal and empirical), and relevant common sense knowledge. Definitions are represented as Lisp predicates, Lisp functions[4] or lists of values. These can be viewed as extensional definitions (*i.e.*, lists), intensional definitions (*i.e.*, predicates and functions) and combinations of the two (*i.e.*, definitions expressed as a predicate with additional values specified with a list).

The task of the epistemological engineer in transcribing the background knowledge is constrained by the prototypes. Only knowledge that is referred to by the prototypes need be represented. If the knowledge is not used in prototypical descriptions, then, at first pass, it can be ignored. Some of this knowledge may be needed later when clinical considerations (*e.g.*, how the knowledge is used in problem solving situations) are introduced.

The background knowledge is used in the generalization phase to bias the generalizations along the lines that experts use. The potential human description of the domain is implicitly represented within the terms and relationships of the background knowledge. The background knowledge for the house example consists of the following:

- Showers and bathtubs are all used to wash humans (**WASH-HUMAN**).
- King, queen, twin, bunk and futon are all types of beds (**BED**).
- The terms many, few and some, as well as all numbers greater than zero are used to count (**COUNTING-MEASURE**).
- Books, magazines and newspapers are all reading material (**LITERATURE**).

The Hierarchy Growing Stage

Using conceptual clustering [28, 43, 21] and the background knowledge, Novice builds multidimensional tangled hierarchies from the prototypes. The system is learning new structural knowledge and

[4]Functions have not been used in any of the data sets. The mechanism provides a hook to allow constructive generalization, if desired.

organizing itself for its diagnostic task. By applying domain independent generalization techniques (*e.g.*, equality and subset relations) and domain knowledge (*e.g.*, the definitions and associations in the background knowledge), groups of prototypes are generalized. By recursively applying this generalization procedure to the newly generated descriptions, multidimensional tangled hierarchies are grown. Filtering techniques are then used to fine tune the network.

Novice's process of generalization is as follows. First create subsets of the descriptions to be generalized (the last layer of the network, initially the prototypes) and generalize each of these subsets. For each subset, look at every aspect of every attribute that is to be generalized[5] one at a time and try to generalize the values. If there are multiple alternative generalizations they are used to generate multiple descriptions, unless one of the alternatives is considered a "poor" generalization.[6] Each new generalization is a potential node in the network. When done with all the subsets for this set of nodes to be generated:

1. merge all equal nodes (*i.e.*, generalizations or descriptions);
2. remove subsumed nodes;
3. move up any nodes that did not generalize with other nodes;
4. if there are any nodes in the old and new layers that have identical descriptions and have the same descendants, merge them; and
5. pick up any other lower nodes that meet any of these nodes' definitions and are not already descendants of the node.

This produces the next layer in the network. Novice recursively operates on these nodes to generate a new level of the network until a stopping criterion is met.

Aggregation

To direct the generalization process groups of descriptions (*e.g.*, prototypes) are created. Generalizations of these aggregates are then used as potential superclasses. Generalization occurs from subsets of lower level descriptions, as opposed to individual descriptions, to provide some direction for the generalization process by providing actual lines of generalization that may occur. For example, if the aggregate contains the numbers 3 and 9 we might generalize the class as `ODD` numbers and not `PRIMES`. Using subsets to generalize over instead of creating generalizations for isolated descriptions constrains the number of potential generalizations.

Which subsets of descriptions should we look at? Do we need to look at every possible subset (*i.e.*, at every member of the power set of the set of descriptions)? Without any further knowledge of the domain we can make one observation: nodes in human hierarchies tend to have few children. But what is this number? As an approximation, we can consider the limit on human short term memory, a limit consisting of 7 ± 2 chunks [31]. This is weakly based on psychological evidence and seems a reasonable approximation. The one instance where this limit is exceeded is where a number of descriptions (greater than our cutoff) generates identical generalizations. Here all the generalizations are merged and the set of descendants of the merged nodes is the union of the descendants of the individual equivalent nodes. This does not require our looking at larger subsets of descriptions but only looking at all the new generalizations.

Combinatoric algorithms are normally frowned upon. But, since we generaly start with fewer than two dozen prototypes and since computers are still good at this monotonous type of work, this is a reasonable approach. The approach must be compared with humans, who often base their learning on problem solving failures. Humans build an initial structure and only when it fails them in practice are they forced to reevaluate their knowledge and possibly reorganize the structure. Computers are better at keeping around the alternatives and choosing among them. This is a point where the motivation to follow human psychology gives way to techniques better suited to the computer, while still sharing certain describable endpoints among the two systems.

Generalization

Once the subsets are generated Novice looks at each subset and decides how to generalize it. The hierarchy growing phase must not generalize too fast (*e.g.*, going from the set of all the prototypes

[5] Attributes that should not be used for generalization can be specified by the epistemological engineer.
[6] Again, these can be specified by the epistemological engineer.

to a single root) or too slowly (*e.g.*, creating many new layers that are not even remotely isomorphic with human hierarchies). To satisfy the first criterion Novice attempts to generalize as slowly as possible, creating maximally specific generalizations for the subsets at each step. Novice relies on the background knowledge to approximate the correct step size (*i.e.*, to generalize fast enough), meeting the second criterion. The step size and direction of generalization is thus controlled by producing maximally specific generalizations of subsets of lower level descriptions with a biased vocabulary. This should ensure that generalizations relevant to human problem solving are considered while limiting the number of generalizations formed.

In generalizing a subset, Novice looks at the values for every aspect of every attribute, for each description and tries to generalize them. Domain independent means (*e.g.*, equality) and domain dependent knowledge (*i.e.*, the background knowledge) are used. The resulting generalization is the description for this subset.

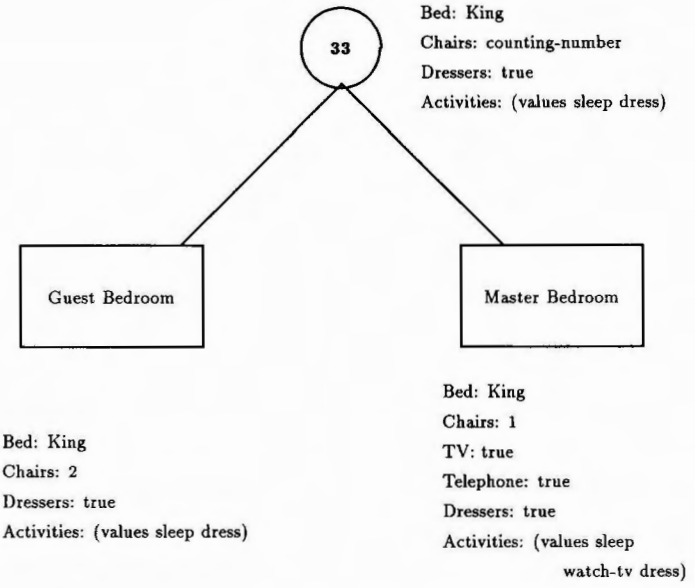


Figure 2: A Sample Generalization from the House Domain.

Let us briefly look at an example. Figure 2 shows the generalization of the `Guest-Bedroom` and the `Master-Bedroom`. Both descriptions, `Guest-Bedroom` and `Master-Bedroom`, have the value `KING` for the attribute `BED`, so the new description, `Interior-Node-33`, has the value `KING` for the attribute `BED`. This is an example of domain independent generalization using equality. Similarly for the attribute `DRESSERS`. For the attribute `ACTIVITIES` another domain independent generalization is found using set intersection. Finally, for the attribute `CHAIRS`, no domain independent generalization can be found so the background knowledge is applied returning the generalization `COUNTING-MEASURE`.

Local *vs.* Global Decisions. All the generalizations are local. Novice only looks at the values for one aspect of the same attribute of the different descriptions in the subset at a time. No values for other attributes or even other aspects of the same attribute are used. In Figure 2 for example, the `values` aspect of the `ACTIVITIES` attribute was generalized independent of any other aspects that might have been under the `ACTIVITIES` attribute and independent of all other attributes.

Domain Independent Generalizations. There are two obvious domain independent generalizations: equality of terms and intersection between lists, for which we have already seen examples. Both are preferred over using background knowledge. For example, for the attribute CHAIRS, both the master bedroom and the single child's bedroom have 1 chair and so would their generalization (as opposed to the generalization COUNTING-MEASURE).

Domain Independent Method with Domain Dependent Facts. Novice searches for a generalization from among the definitions and the relationships among the extensional definitions. For example, in the house domain, the values for CHAIRS for the guest bedroom (2) and the master bedroom (1) are not equal but both meet the definition for a COUNTING-MEASURE, so that is the generalized value. The relationships among the extensional knowledge need not describe a tree — it may be any directed acyclic graph (DAG).

Multiple Generalizations. When the values for an aspect of an attribute generalize in more than one way, the generalizations might be combined into descriptions in one of two different ways: (1) as multiple, separate descriptions, or (2) as a conjunction of the different values. In the first method, new nodes are created for each alternative. In the second, all the alternative generalizations are combined into a single new value for this attribute-aspect and that is the value used in the generated description. This second method is only used when one of the alternative generalizations is a "poor" generalization.

Sometimes there are known "poor" generalizations that can be made. The epistemological engineer specifies a list of poor generalizations, meaning that these generalizations are not as trustworthy as others for statistical or other reasons. For example, every human is either male or female. We would not want to be hypothesizing that being male (or female) is important to a class (*e.g.*, homicidal maniacs) based on only a few exemplars since the probability of getting all males in a subset by chance alone is still high. If any generalization that is on this list of poor generalizations (values specified for a particular aspect of an attribute) is found among the alternatives suggested, then, instead of using the alternative generalizations to make multiple descriptions, the alternatives are combined as a single value for this attribute-aspect.

Filtering the Hierarchies

After all the subsets have been generated, the collection of generated descriptions is filtered. Novice merges equal descriptions, ensures that only the most specific generalizations are kept, handles descriptions that did not generalize, finds identical descriptions appearing in the old and new layers, and picks up any additional nodes meeting a new node's description.

After the hierarchy growing phase has finished (it either exceeded limits set within Novice for the size of the new layer, the new layer and the old layer are equivalent, or there were no new nodes generated), the root nodes are inspected for possible additions or pruning (including all descendants uniquely below this root). There are three operations applied here. First, the epistemological engineer can specify certain attribute-aspect-values that should not appear at the top of the network. This will be further described below. Also, the epistemological engineer can specify certain values as mutually exclusive for a particular attribute-aspect. Here, if one of the values appears as a root and the other values do not appear as roots, then new nodes are added for the other values. This occurred in the bacteremia example described below, when GRAM-POSITIVE was discovered as a root. Novice noticed that it was missing roots for GRAM-NEGATIVE and ACID-FAST, the other two mutually exclusive values. Novice then tries to add nodes for these roots. At this time, it finds out that there are no descriptions with the value ACID-FAST, so only a new root for GRAM-NEGATIVE is added. Finally, Novice ensures that there is more than one root.

One way to view the roots is as screening questions, the first pieces of information you would like to know when faced with a new case. Often there are data that are either too dangerous or expensive to obtain, or that are otherwise inappropriate as screening questions. The epistemological engineer can specify these values and Novice will remove roots that use these values and their respective subtrees.

The House Example Hierarchies

Figure 3 shows the complete network developed from the house example. The resulting hierarchies show some nice properties. The bedrooms are all grouped together in Interior-Node-36, these are rooms with a bed, chairs, dressers, and in which we sleep and dress. Of course, Novice does not recognize this as a description of a *bedroom*, only we do that, but the description *is* of a

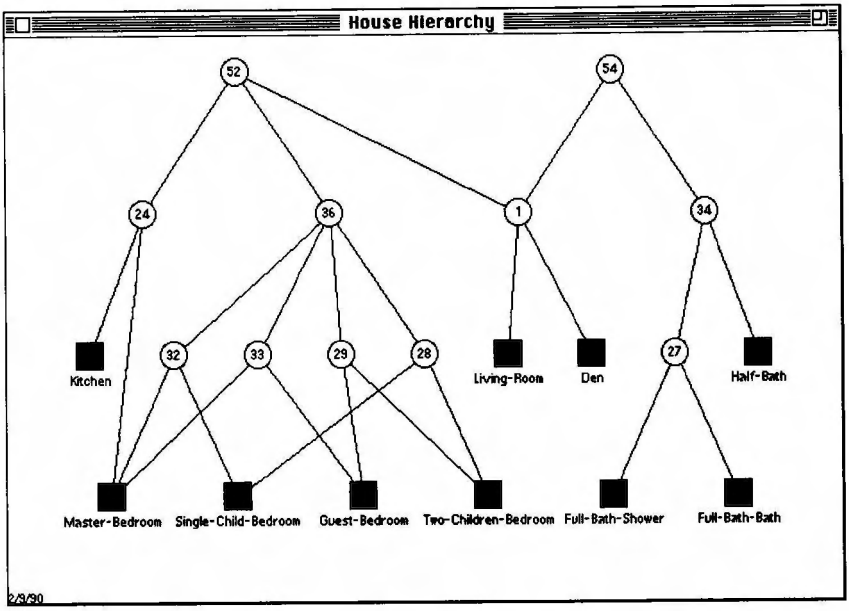


Figure 3: The Resulting Network from the House Example (a screen dump).

bedroom. There is a layer between this node and the various bedroom prototypes in which similarities among various subsets of bedrooms are emphasized (*i.e.*, `Interior-Node-28`, `Interior-Node-29`, `Interior-Node-32` and `Interior-Node-33`). It is only after this layer that Novice recognizes *bedrooms* as a category.

With bathrooms, Novice first groups together the two *full bathrooms* in `Interior-Node-27` (*i.e.*, those with either a shower or a bathtub) and then adds the `Half-Bath` (*i.e.*, a bathroom with only a toilet and no shower or bathtub) in `Interior-Node-34` to create a *bathroom* node.

Novice even creates a *TV room* category (`Interior-Node-24`). This category includes both the `Kitchen` and `Master-Bedroom`. In reality it is more specific than a TV room (which also should include the `Den`), it is really a TV room with a telephone and chairs (which is why the `Den` was excluded).

Finally, the top level descriptions are rooms in which we read (`Interior-Node-54`) and rooms with chairs (`Interior-Node-52`).

Individuals might quibble with details of the prototypes or details of the resulting network, but that illustrates the power of the technique. By laying out all the connections explicitly, it is very easy for you (or for the expert) to offer a critique of the information. The system has provided the context that helps make the critique easy, even without a specific problem.

The Rule Extraction Stage

After organizing the prototypes Novice can extract problem solving rules based on the network and the task of heuristic classification. These include rules to navigate the structure of the domain (*i.e.*, the network) and trigger rules based on the importance of a datum or of a particular class. Rules may span a single link or multiple links of the network, jump to a particular node (description) or exclude a prototype. Since the rules are based on abstract class descriptions, they are too general and need to be made more specific in the next learning phase. The network provides the context for learning rules that discriminate among the diseases as they might present themselves in the clinic. Providing context to the rule generating mechanism is key to limiting the number of rules suggested.

Two annotations on the learned rules are important for future learning. Both `certain` and `probable` rules are learned. Even though most problem solving knowledge is probabilistic, rules learned from the hierarchies are labeled as certain — they are certain in the context of the idealized

prototypes. Other learned rules, from such aspects as **rare** or **common**, represent probabilistic knowledge even in the context of the idealized prototypes.

Finally, all the rules are stored on the nodes of the network. In essence, a node and the path to it through the network serve as additional antecedents of each rule, *e.g.*, "If you are at this node, and some value is true for this attribute under this aspect, then go to a particular node."

There are four different types of rules that are extracted from the network.

- *Pathognomonic* rules, which link data unique to a particular prototype to that prototype.
- *Exclusionary* rules, which rule out prototypes based on necessary or definitely excluded values for the prototype.
- *Splitting attribute* rules, which follow any values for a single attribute of the prototype to a unique descendent.
- *Multiple level* rules, which find unique appearances of values in the subtrees rooted at a particular node and jump to the node where that value appears. These values do not appear in the node itself (that is the difference between a multiple level rule and a splitting attribute rule).

Both the pathognomonic and exclusionary rules are treated as daemons (data triggers), since they are based on context-free information and allow immediate decisions to be made that will either focus the diagnostic search or increase the efficiency of the search by removing prototypes from consideration. Let us first describe the different rule types and then illustrate them with the house example.

Splitting Attribute Rules

These rules focus on the values for the attributes that appear at a particular node. The rules represent unique paths out of the node for an attribute-aspect-value triplet, usually to a child but possibly to another descendant. These rules split the descendants by singling out an individual descendant. Splitting attribute rules are the work horses of the problem solving system.

Multiple Level Rules

These are a context-sensitive generalization of pathognomonic rules — the rules represent data that appear uniquely *within a subtree* of the network, but not in the root of the subtree. Novice looks at the values appearing below a specific node (but not in that node's description) to see if any of them lead to a single node. If so, a rule is created linking this value to that node. These rules increase the efficiency of problem solving by encouraging multiple level inferences. They compensate for deficiencies in the hierarchy by skipping over extraneous layers in the hierarchy. The same rule might include many ancestor nodes leading to the same descendant, if the chain is long.

Pathognomonic Rules

A pathognomonic datum is a datum that is unique to a single prototype — it occurs in only a single solution class. Given that datum, you know the solution. For instance, in discussing animals, flying is pathognomonic of birds.[7] This information is context-free; whenever we find this datum, irregardless of the current node we are evaluating, a solution immediately suggests itself. A pathognomonic rule changes the focus of the search; given the datum Novice immediately tries to verify the solution it suggests. These rules are treated as data triggers (*i.e.*, daemons) to alter the focus of the problem solver.

The data are also used in splitting attribute rules and multiple level rules. Pathognomonic rules only operate with information that is volunteered by the user at the beginning of a problem solving session (since this is the only opportunity the user has to volunteer information about the case). To ensure that the information is asked for when it is needed, if it has not already been volunteered, these data are also used in splitting attribute and multiple level rules. This also ensures that the information is stored at all the nodes where it serves to discriminate among paths, so that it might later be used to make more complex rules: probabilistic rules with more than one part to their antecedents (the pathognomonic data might only be unique in a probabilistic sense). The same data appearing in a splitting attribute or multiple level rules will lead to the same node as the pathognomonic rule would conclude. The difference is that the splitting attribute and multiple level rules are asked by the system when the information is needed, while the pathognomonic rules only use volunteered information.

[7] OK, bats fly. This just shows the probabilistic nature of prototypical reasoning.

Exclusionary Rules

Exclusionary rules make use of the aspects **always** (**common**) and **never** (**rare**) and the data for a particular case to increase the efficiency of problem solving by ruling out potential solutions. Knowing a value does not appear in a case will rule out any prototype for which that value is mandatory. The presence of a value rules out any prototype for which that value is excluded from the prototype. In probabilistic inferences our certainty in a particular solution is reduced instead of ruled out.

This information is context-free, depending only on the prototypes and the case's data. So, like the pathognomonic rules, these rules are data triggered, removing nodes from consideration whenever applicable. While it sounds like we are unconditionally excluding prototypes and might be wrong,[8] the final problem solver will handle the probabilistic nature of diagnosis and not unconditionally rule out these solutions. Exclusionary rules increase efficiency of problem solving by removing prototypes from consideration and increase the accuracy of diagnosis by reducing our belief in solutions that are contraindicated.

The House Example Rules

The house example also demonstrates the rule types. Having a **SHOWER** is pathognomonic of **Full-Bathroom-Shower**, while having an **OVEN** is pathognomonic of the **Kitchen**. These facts create pathognomonic rules.

Novice is told that we **never EAT** in the **Living-Room**. So, if one of the activities for the room we are trying to classify is eating, then we know that we need not consider the **Living-Room**. This is captured in an exclusionary rule.

To demonstrate the splitting rules and multiple level rules, consider the *bedroom* node (**Interior-Node-36**) and its children (**Interior-Node-28**, **Interior-Node-29**, **Interior-Node-32** and **Interior-Node-33**; the descriptions are reproduced in Figure 4). There are five splitting attribute rules: two to **Interior-Node-28** and one to each of the other three children.

- If **DESK** has **values (COUNTING-MEASURE)** then go to **Interior-Node-28**.
- If **ACTIVITIES** has **values (PLAY)** then go to **Interior-Node-28**.
- If **CHAIRS** has **values (2)** then go to **Interior-Node-29**.
- If **CHAIRS** has **values (1)** then go to **Interior-Node-32**.
- If **BED** has **values (KING)** then go to **Interior-Node-33**.

There are seven multiple level rules associated with **Interior-Node-36**: three leading to the **Master-Bedroom**, two to the **Single-Child-Bedroom** (there are also corresponding pathognomonic rules) and two to the **Two-Children-Bedroom** (with corresponding pathognomonic rules).

- If **ACTIVITIES** has **values (WATCH-TV)** then go to **Master-Bedroom**.
- If **TELEPHONE** has **values (T)** then go to **Master-Bedroom**.
- If **TV** has **values (T)** then go to **Master-Bedroom**.
- If **DESK** has **values (1)** then go to **Single-Child-Bedroom**.
- If **DESK** has **values (2)** then go to **Two-Children-Bedroom**.
- If **BED** has **values (TWIN)** then go to **Single-Child-Bedroom**.
- If **BED** has **values (BUNK)** then go to **Two-Children-Bedroom**.

Novice created 130 rules for the house example: 61 splitting attribute rules, 44 multiple level rules (36 of these involve data that is also found in pathognomonic rules), 5 exclusionary rules and 20 pathognomonic rules.

Novice Evaluation

To test the adequacy of the system, we compared the rules Novice developed to those found in MYCIN [41, 7] and tested Novice on a simple case from the MYCIN literature. Finally, we examined Novice's generality by testing it in a nonmedical domain.

[8] I know I have eaten in the living room on occasion.

```
Next Layer
==========

Name:       Interior-Node-36
Parents:    (Interior-Node-52)
Children:   (Interior-Node-28 Interior-Node-29 Interior-Node-33 Interior-Node-32)
ACTIVITIES: (values DRESS SLEEP)
BED:        (values BED)
CHAIRS:     (values COUNTING-NUMBER)
DRESSERS:   (values T)

Next Layer
==========

Name:       Interior-Node-32
Parents:    (Interior-Node-36)
Children:   (Single-Child-Bedroom-1 Master-Bedroom-1)
ACTIVITIES: (values DRESS SLEEP)
BED:        (values BED)
CHAIRS:     (values 1)
DRESSERS:   (values T)

Name:       Interior-Node-33
Parents:    (Interior-Node-36)
Children:   (Guest-Bedroom-1 Master-Bedroom-1)
ACTIVITIES: (values DRESS SLEEP)
BED:        (values KING)
CHAIRS:     (values COUNTING-NUMBER)
DRESSERS:   (values T)

Name:       Interior-Node-29
Parents:    (Interior-Node-36)
Children:   (Two-Children-Bedroom-1 Guest-Bedroom-1)
ACTIVITIES: (values DRESS SLEEP)
BED:        (values BED)
CHAIRS:     (values 2)
DRESSERS:   (values T)

Name:       Interior-Node-28
Parents:    (Interior-Node-36)
Children:   (Two-Children-Bedroom-1 Single-Child-Bedroom-1)
ACTIVITIES: (values DRESS PLAY SLEEP)
BED:        (values BED)
CHAIRS:     (values COUNTING-NUMBER)
DESK:       (values COUNTING-NUMBER)
DRESSERS:   (values T)
```

Figure 4: Bedroom Subtree of the House Example

Rule Comparison

Novice was given prototypes for bacteremia (blood infections) and background knowledge from a standard medical text [38], augmented with other teaching material. Most of MYCIN's background knowledge can be found in texts, although it is not always obvious what knowledge is relevant. Prototypes and their inter-relationships incorporate much of MYCIN's judgemental knowledge used in diagnosing the most common classes. Missing are special cases of these classes and some rare classes, as expected. Some (but not all) diagnostic (or clinical) knowledge, knowledge about how to reach a conclusion in the domain and reasoning about the data's validity, can be found in the text, although not in the prototypes themselves. MYCIN rules missing from Novice include information to exclude a diagnosis, temporal reasoning (something Hamatmid does not even attempt to address), common sense knowledge (the need for some of this knowledge is obvious enough that it can be included by the epistemological engineer when entering the background knowledge, but the general problem of common sense reasoning is not addressed by Hamatmid), and reasoning about the laboratory data needed to derive the concepts that the prototypes and background knowledge mention. Much of this knowledge can be added in later stages when learning is based on problem solving. The important question of whether Novice's rules can be easily modified to approximate an expert's representation remains largely unexplored.

Case Comparison

To compare the problem solving ability of the two systems, we used the case that Shortliffe used to demonstrate MYCIN [41, pp. 48–55]. The expectation is that Novice would include all MYCIN's

MYCIN	Novice
Pseudomonas	Pseudomonas aeruginosa (67%)
Klebsiella	Klebsiella (80%)
Enterobacter	Enterobacter (80%)
Escherichia coli	Escherichia coli (60%)
Proteus (unknown)	Proteus (75%)
—	Acinetobacter (75%)
—	Meningococci (28.6%)

Table 1: A comparison of MYCIN's and Novice's problem solving behavior. The percentages are the percent of attributes that matched the prototype.

suggestions plus a few additional ones. These additional diagnoses result from the over generality of Novice's rules. There should not be too many additional suggestions though. (Novice included only 20 possible solutions, while MYCIN has 51 possible solutions.)

Table 1 compares MYCIN's and Novice's problem solving ability. Novice used a bare bones problem solving system to traverse the network; this is not the final problem solving system for the system. The problem solving system also reported dead ends, *i.e.*, nodes where no rule matched. Two dead ends were reported. The bacteria under them are: Enterobacter, Klebsiella and Serratia; and Bacteriodes, Enterobacter, Escherichia coli, Haemphilus influenzae, Klebsiella, Pseudomonas aeruginosa, Pseudomonas pseudomallei and Serratia. These may represent possible solutions, but more importantly, they represent learning opportunities: nodes whose rules must be made more specific or places where exclusionary information might be important.

One example of Novice's over generality is demonstrated in the diagnosis of Meningococci. Meningococci was included as a solution because of a rule asking about age greater than 25 years. A combination of this rule and another rule that asked whether the nasopharynx was the portal of entry (instead of asking about age by itself) would have been enough to prevent this bacterium from being considered.

Generality

The method, as implemented in Novice, should be generally applicable to any domain where (1) the task is classification (2) the solution classes are presented as prototypes (idealized cases), (3) there is additional background knowledge to supplement these solutions (optional) and (4) there is a well developed repository of educational material to use (a domain where all the knowledge is still transferred between the generations by apprenticeship is not a good candidate domain). Examples of such a domain are geology, zoology and device repair. Even areas like business and law, where case studies are the basis of the educational system, are potential domains for this method after first extracting idealized prototype from the case studies.

In a test of general applicability we used a simple domain: the repair of a home humidifier. This was chosen to ease the task of knowledge acquisition and to allow some experimentation with negative (facts that do not appear in the class) and exclusionary (facts that can be used to rule out a class) knowledge. The knowledge came from a table stating whether every feature is present or absent for each diagnostic class (*i.e.*, necessary and sufficient conditions). Novice's rules with the problem solving system correctly selected classes for the cases given it, although it often suggested extraneous solutions.[9] Negative information increased the specificity of the resulting diagnosis by providing more data on which to discriminate the solutions. Exclusionary information increased specificity by removing solutions from consideration. Finally, the combination of both negative and exclusionary information allowed the problem solving system to diagnose correctly every case.

[9]For the two cases where incorrect suggestions were made (as classes under a dead-end that did not contain the correct solution), the problem solving system warned that these were learning opportunities to be exploited. These occurred with the inclusion of negative information but no exclusionary information. The underlying problem here was a successful partial match on the negative data on the nodes though there was contradictory positive information

Comparison to Existing Systems

Hamatmid builds on current machine learning and knowledge acquisition techniques to pattern its development of expertise after humans. Novice uses similarity based learning techniques focused by the abstraction [15] inherent in textbook presentations. The hierarchy building phase is an example of conceptual clustering using background knowledge [43]. Basing the learning on abstract prototypes in a bottom-up approach and the heuristics used in the hierarchy building phase are what sets Novice apart from the other approaches. The rule extraction phase is unique in its learning of pathognomonic, trigger rules from the prototypes. Future work on Hamatmid will draw from supervised learning while problem solving (*e.g.*, [16, 24]), weaning rules from the network (*e.g.*, [39]), learning exceptions in a network (*e.g.*, [23]), strengthening weights of rules and links (*e.g.*, [1, pp. 249–255] or [23, pp. 70–75]), and compilation methods (*e.g.*, [26, 18, 44]), among other techniques. In each phase current machine learning research will serve as a basis, to be extended as necessary for Hamatmid.

This method is yet another technique to provide a framework for knowledge elicitation. It is based on the developmental path of experts — not on methods to elicit the expert's knowledge representation (*e.g.*, [6, 4, 35]). It shares many goals with PROTOS [2] but is committed to a rule-based representation rather than a case-based representation (for the benefits rules offer in incremental learning and explanation), although neither system is based on a single representation.

Conclusions

Hamatmid outlines a phased knowledge acquisition system for the development of computer expertise. The method parallels an idealized picture of medical education: the assimilation of textbook knowledge, problem solving simple cases with a teacher, then handling more difficult cases with feedback about the correct outcome (and someone to answer system initiated questions) and finally developing expertise on its own. Novice, the first stage of Hamatmid, includes many necessary features for heuristic classification systems: structural knowledge and support knowledge based on introductory material in the domain. Its strategic knowledge is built into the problem solving system itself (an obvious topic for future research). Novice develops problem solving rules from these sources after imposing additional structure on the descriptions. Other task specific problem solving rules must be added by the epistemological engineer. Future work will both try to improve Novice and implement the remaining stages of Hamatmid. The method seems ideal for domains where knowledge is presented as abstracted cases and the features describing the cases are related to the solutions through multiple, tangled hierarchies. Medical diagnosis is certainly such a domain. Human knowledge appears replete with many others also.

References

[1] J. R. Anderson. *The Architecture of Cognition.* Harvard University Press, Cambridge, 1983.

[2] R. Bareiss, B. W. Porter, and K. S. Murray. Supporting start-to-finish development of knowledge bases. *Machine Learning*, 4:259–283, 1989.

[3] H. S. Barrows and P. J. Feltovich. The clinical reasoning process. *Medical Education*, 21:86–91, 1987.

[4] J. S. Bennett. ROGET: A knowledge-based system for acquiring the conceptual structure of a diagnostic expert system. *Journal of Automated Reasoning*, 1:49–74, 1985.

[5] F. Bergadano, A. Giordana, and S. Ponsero. Deduction in top-down inductive learning. In *Proceedings of the Sixth International Conference on Machine Learning*, pages 305–317, San Mateo, CA, 1989. Morgan Kaufmann Publishers, Inc.

[6] J. H. Boose. A knowledge acquisition program for expert systems based on personal construct theory. *International Journal of Man-Machine Studies*, 23:495–525, 1985.

[7] B. G. Buchanan and E. H. Shortliffe, editors. *Rule-Based Expert Systems.* Addison-Wesley, Reading, MA, 1984.

[8] W. G. Chase and H. A. Simon. The mind's eye in chess. In W. G. Chase, editor, *Visual Information Processing*, pages 215–281. Academic Press, Inc., New York, 1973.

[9] M. T. H. Chi, R. Glaser, and E. Rees. Expertise in problem solving. In R. J. Sternberg, editor, *Advances in the Psychology of Human Intelligence*, chapter 1, pages 7–75. Lawrence Erlbaum Associates, Hillsdale, NJ, 1982.

[10] W. J. Clancey. The epistemology of a rule-based expert system — a framework for explanation. *Artificial Intelligence*, 20:215–251, 1983.

[11] W. J. Clancey. Heuristic classification. *Artificial Intelligence*, 27:289 –350, 1985.

[12] W. J. Clancey. The knowledge level reinterpreted: Modeling how systems interact. *Machine Learning*, 4:285–291, 1989.

[13] P. Cohen, D. Day, J. Delisio, M. Greenberg, R. Kjeldsen, D. Suthers, and P. Berman. Management of uncertainty in medicine. In *Proceedings of the IEEE Conference on Computers and Communications*, pages 501–506, 1987.

[14] P. R. Cohen. *Heuristic Reasoning about Uncertainty: An Artificial Intelligence Approach.* Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1985.

[15] L. Darden. Viewing the history of science as compiled hindsight. *AI magazine*, 8(2):33–41, 1987.

[16] R. Davis. Interactive transfer of expertise: Acquisition of new inference rules. *Artificial Intelligence*, 12:121–157, 1979. A shortened version appears in [7].

[17] R. M. Dawes, D. Faust, and P. E. Meehl. Clinical versus actuarial judgement. *Science*, 243:1668–1674, 1989.

[18] G. DeJong. An introduction to explanation-based learning. In Howard E. Shrobe and the American Association for Artificial Intelligence, editors, *Exploring Artificial Intelligence*, pages 45–81. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1988.

[19] A. S. Elstein, L. S. Shulman, and S. A. Sprafka. *Medical Problem Solving.* Harvard University Press, 1978.

[20] P. J. Feltovich, P. E. Johnson, J. H. Moller, and D. B. Swanson. LCS: The role and development of medical knowledge in diagnostic expertise. In W. J. Clancey and E. H. Shortliffe, editors, *Readings in Medical Artificial Intelligence: The First Decade*, pages 275–319. Addison-Wesley, Reading, MA, 1984.

[21] D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2):139–172, 1987.

[22] S. J. Hanson and M. Bauer. Conceptual clustering, categorization, and polymorphy. *Machine Learning*, 3(4):343–372, 1989.

[23] J. H. Holland, K. J. Holyoak, R. E. Nisbett, and P. R. Thagard. *Induction: Processes of Inference, Learning and Discovery.* The MIT Press, Cambridge, MA, 1986.

[24] G. Kahn, S. Nowlan, and J. McDermott. Strategies for knowledge acquisition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-7:511–522, September 1985.

[25] B. Kuipers and J. P. Kassirer. Causal reasoning in medicine: Analysis of a protocol. *Cognitive Science*, 8:363–385, 1984.

[26] J. E. Laird, P. S. Rosenbloom, and A. Newell. Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, 1(1):11–46, 1986.

[27] D. L. Medin, W. D. Wattenmaker, and S. E. Hampson. Family resemblance, conceptual cohesiveness, and category construction. *Cognitive Psychology*, 19:242–279, 1987.

[28] R. S. Michalski. Knowledge acquisition through conceptual clustering: A theoretical framework and an algorithm for partitioning data into conjunctive concepts. *International Journal of Policy Analysis and Information Systems*, 4(3):219–244, 1980.

[29] R. S. Michalski. A theory and methodolgy of inductive learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 83–129. Tioga Publishing Company, Palo Alto, CA, 1983.

[30] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors. *Machine Learning: An Artificial Intelligence Approach*, volume 2. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1986.

[31] G. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63(2):81–97, 1956.

[32] M. Minsky. A framework for representing knowledge. In P. Winston, editor, *The Psychology of Computer Vision*, pages 211–277. McGraw-Hill, Inc., New York, 1975.

[33] S. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 339–352, San Mateo, CA, 1988. Morgan Kaufmann Publishers, Inc.

[34] G. L. Murphy and D. L. Medin. The role of theories in conceptual dependence. *Psychological Review*, 92(3):289–316, July 1985.

[35] M. A. Musen. Automated support for building and extending expert models. *Machine Learning*, 4:347–375, 1989.

[36] V. L. Patel and G. J. Groen. Knowledge based solution strategies in medical reasoning. *Cognitive Science*, 10:91–116, 1986.

[37] R. S. Patil. Artificial intelligence techniques for diagnostic reasoning in medicine. In H. E. Shrobe and the American Association for Artificial Intelligence, editors, *Exploring Artificial Intelligence*, pages 347–379. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1988.

[38] R. G. Petersdorf, R. D. Adams, E. Braunwald, K. J. Isselbacher, J. B. Martin, and J. D. Wilson, editors. *Harrison's Principles of Internal Medicine.* McGraw-Hill, Inc., New York, 10th edition, 1983.

[39] J. R. Quinlan. Generating production rules from decision trees. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 304–307, San Mateo, CA, 1987. Morgan Kaufmann Publishers, Inc.

[40] R. C. Schank, G. C. Collins, and L. E. Hunter. Transcending inductive category formation in learning. *Behavioral and Brain Sciences*, 9:639–686, 1986.

[41] E. H. Shortliffe. *Computer-Based Medical Consultations: MYCIN.* Elsevier, New York, 1976.

[42] E. E. Smith and D. L. Medin. *Categories and Concepts.* Harvard University Press, Cambridge, MA, 1981.

[43] R. E. Stepp III and R. S. Michalski. Conceptual clustering: Inventing goal-oriented classifications of structured objects. In Michalski et al. [30], pages 471–498.

[44] G. G. Towell, J. W. Shavlik, and M. O. Noordewier. Refinement of approximate domain theories by knowledge-based neural netwroks. In *Proceedings of the Eigth National Conference on Artificial Intelligence*, pages 861–866, Menlo Park, CA, 1990. AAAI Press.

A New Approach to Classification Knowledge Refinement

Ken-ichi Hagiwara, Manager
System Development Department No.1
FUJIFACOM CORPORATION
Hino-shi Tokyo Japan

Abstract

The 'choice component' with the 'include-option' is one of powerful representations for classification knowledge. In the conventional construction of a choice component type KB man must prepare the necessary and sufficient findings and case data beforehand, think of all the cases of each goal concept and decide exactly all the clauses. But this conventional work is often difficult for field experts to do. In order to refine a choice component clause step by step based on the case data which may contain insufficient examples or incorrect examples a new algorism is proposed here. The algorism introduces the "primary choice component cube" in the "base findings space" and reduces the amount of calculation. The algorism is applied further to choice component clauses conjunction refinement. The efficiency of the algorism in the field engineering is discussed. And the algorism is compared with other methods which induce rules from case data.

1.Classification Knowledge Representations

Classification knowledge of a goal concept C decides whether an object with some observations belongs to the concept C or not.
Classification knowledge is used mainly in order to judge the situation in the applications of diagnosis, scheduling, design, control and so on.
In general classification knowledge is represented as a set of following rules:

if C1,C2,--- then C.

Here C1, C2, --- are logical clauses and if all C1, C2, --- are satisfied then a goal concept C is concluded. C1,C2,--- are often arranged in the order of confirmation cost so that classification can be done as low cost as possible. If there are several cases of a goal concept, they are represented in different rules with the same goal. The type of a logical clause Ci is generally a finding and the classification knowledge is represented as a discrimination tree.
The 'choice component' with the 'include-option' is one of logical clause representations and has the following form:

Ci: mi of (A,B,---) [inc ni of (X,Y,---)]
where A,B,---,X,Y,--- are findings,
$0 < mi <=$ the elements number of (A,B,---),
$0 <= ni <=$ the elements number of (X,Y,---) and
$(X,Y,---) \subset (A,B,---)$.

Here (A,B,---), (X,Y,---), mi, and ni are called "base findings list", "included findings list", "choice number" and "including number" of the choice component clause respectively.
This notation means 'if mi or more findings in (A,B,---) are observed and ni or more findings in (X,Y,---) are included in the observations, then the choice component Ci is satisfied.' The include-option "inc ni of (X,Y,---)" enables a restriction representation but may be omitted.
The choice component is based on the addition of findings. Therefore the choice component is efficient to acquire and maintain classification knowledge when the relation of goal concepts and observations is unknown, the set of the findings is not yet fixed and so the knowledge-base is constructed with cut and try.
In addition the choice component is tolerant of unknown findings in an inference phase. The choice component is often used in medical and mechanical diagnosis systems. For example the rheumatology consultant system[1] uses the criteria table representation containing the choice component.

2. The Aim of This Research

2.1. Conventional Construction of a Choice Component type KB

A choice component type KB is constructed conventionally in the following steps [Figure 1]:

1)Definition of the base findings lists
Man arranges the relationship between goal concepts and observations. And he/she defines the base findings lists for each choice component clause.

2)Preparation of the case-base
Man collects examples of the real operation in the field. And he/she arranges the examples to extract pairs of observations and experts conclusion for the observations. These pairs are stored in the case-base.

3)Definition of each choice component clause
Man defines all the choice numbers, the includind numbers and the included findings lists of each choice component clause. Matrixes of goal concepts and observations, trouble shooting charts and so on are often used to acquire knowledge in this step.

4)Tuning of the choice numbers
Man compares the conclusion of the reasoning with the KB and experts conclusion in the case-base. If the KB makes a mistake, he/she tunes choice numbers in the KB to have the same conclusion as experts one.
SEEK[2] and SEEK2[3] are proposed as a refinement tool of choice component clauses without include-option. These systems find 'false positive' and 'false negative' clauses and select the target clauses for 'specialization' and 'generalization' respectively based on the 'hill climbing' algorism. These systems are strongly influenced from the

property of the case-base which stores pairs of a set of observations and experts conclusion for the observations. And a failure of generalization / specialization or a repetition of generalization and specialization may happen, because these systems don't estimate the effect of the current generalization and specialization.

Therefore the KB should be well defined and case-base of good quality should be prepared before the tuning step by SEEK/SEEK2. In addition the tuning process should be done over again when a part of the base findings lists is modified.

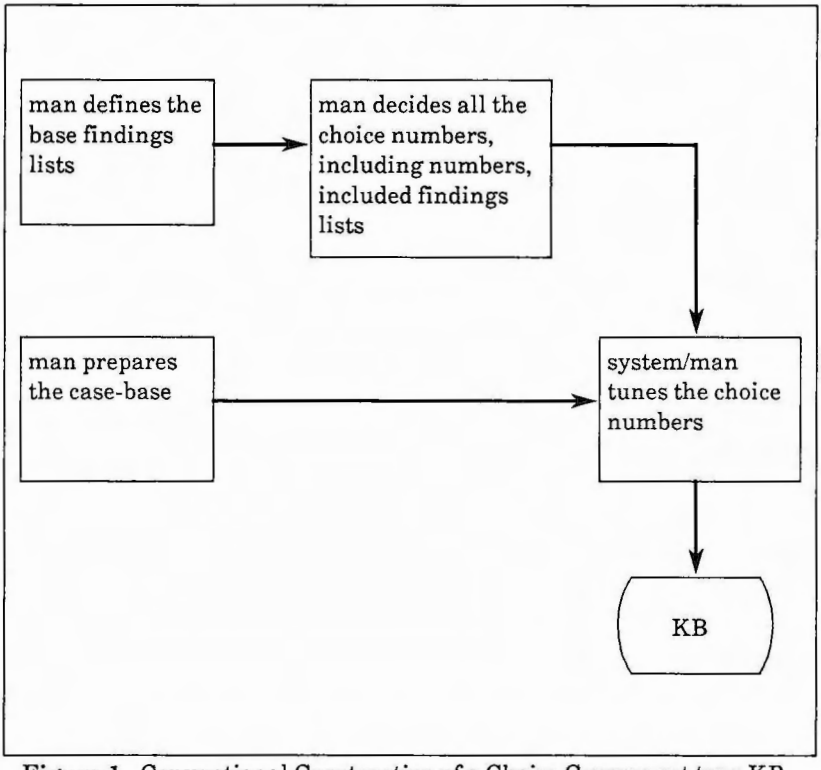


Figure 1 Conventional Construction of a Choice Component type KB

There are some difficulties concerned to knowledge acquisition in the conventional engineering. It is difficult for field experts

* to define all the necessary and sufficient findings beforehand,
* to think of all the cases of each goal concept,
* to decide exactly all the clauses (each choice number, including number and included findings list) at a time and
* to prepare the necessary and sufficient case-base.

Therefore the modification of findings lists, choice component clauses and classification rules is often necessary and the steps are repeated.

2.2. A New Approach to Choice Component type KB Construction

As mentioned above it is required in the choice component type KB construction that the definition of the clauses is supported based on the case-base which may contain insufficient examples or incorrect examples and that the modification of the KB is efficiently performed. The aim of this research is to build up such a support system [Figure 2]. And in this paper the basic algorism of the system is proposed.

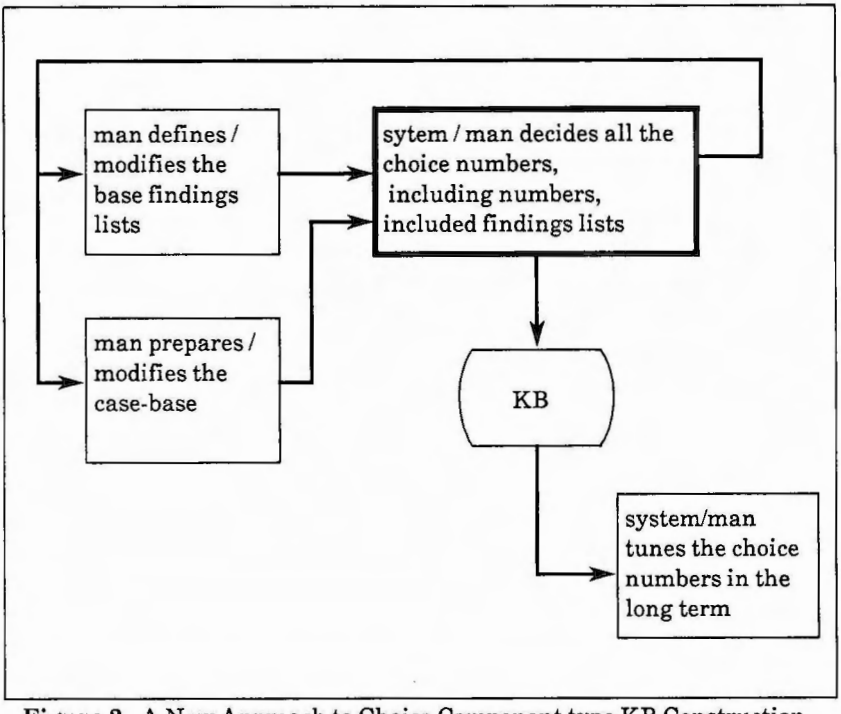


Figure 2 A New Approach to Choice Component type KB Construction

3. A New Algorism of Choice Component Clauses Refinement

3.1. Preparation

[Base findings space]
Definition :

The base findings space (S_B) of a choice component clause (C) is a space which is made up with the base findings in L_B :

C : m of L_B inc n of Li

where L_B = (F1, F2, ---, FN)

S_B : [F1, F2, ---, FN]

[**Choice component cube**]
Definition :

The choice component cube (Cu) corresponding to a choice component clause (C) is a super cube which covers all the area in the base findings space where the clause is satisfied [Figure 3].

C : m of L_B inc n of Li

where L_B = (F1, F2, ---, FN)

Cu : [F1, F2, ---, FN | m of L_B inc n of Li]

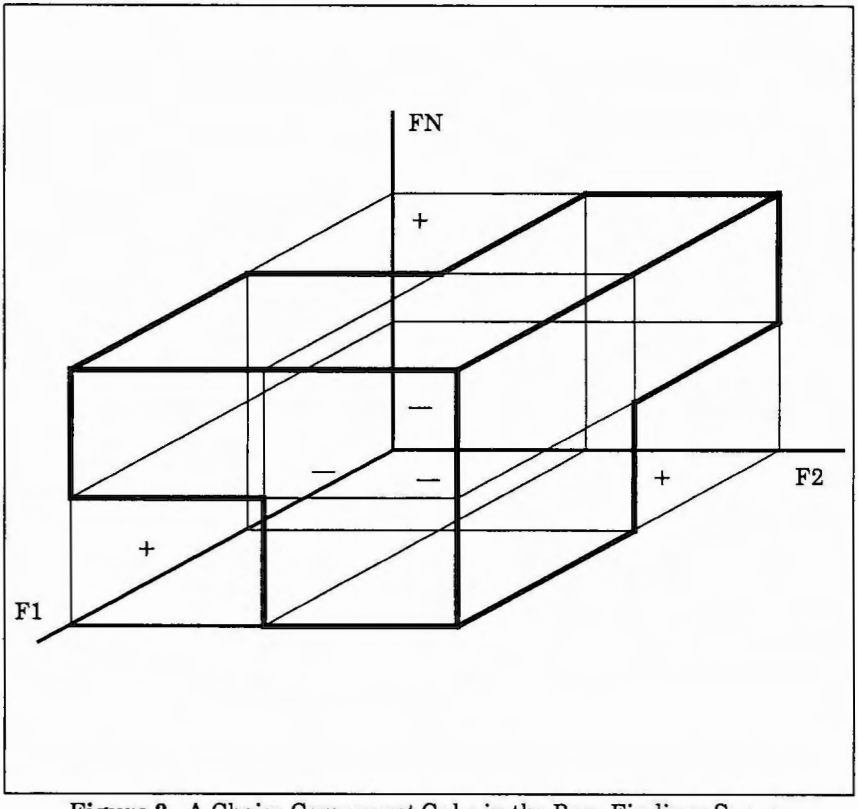


Figure 3 A Choice Component Cube in the Base Findings Space

[**Primary choice component cube**]
Definition :

The primary choice component cube (PCu) is a choice component cube corresponding to a choice component clause whose including number (n) equals to the elements number of the included findings list (Li).

PCu : [F1, F2, ---, FN | m of L_B inc n of Li]

where L_B = (F1, F2, ---, FN),

n = the elements number of Li

Theorem :

Any choice component cube can be divided into primary choice component cubes.

Cu : [F1, F2, ---, FN | m of L_B inc n of Li]

where L_B = (F1, F2, ---, FN),

n = < m.

Cu = $\cup_{k=1,K}$ PCuk

where K = $_N$Cn,

PCuk : [F1, F2, ---, FN | m of L_B inc n of Lik],

Lik $\subseteq$ Li ,

the elements number of Lik = n

[Implication]

Definition :

A primary choice component cube PCu1 = [F1, F2, ---, FN | m1 of L_B inc n1 of Li1] implies another primary choice component cube PCu2 = [F1, F2, ---, FN | m2 of L_B inc n2 of Li2] (PCu1 $\vdash$ PCu2), if and only if

1) m1 = m2, n1 < = n2, Li1 $\subseteq$ Li2 or

2) m1 < = m2, n1 = n2, Li1 = Li2

If PCu1 implies PCu2, PCu1 is always satisfied when PCu2 is satisfied.

The primary choice component cubes are partialy ordered in this implication.

3.2. A New Algorism of a Choice Component Clause Refinement

At first a new refinement algorism of a choice component clause with the include-option is proposed. Next application of the algorism to choice component clauses conjunction refinement is introduced.

Man defines a base findings list for a choice component clause. He/she prepares also the case-base for the choice component clause which stores positive / negative examples of the goal concept. Each example is a pair of observations and experts conclusion (+ for the goal concept / - for other than the goal concept).

The problem to be solved is

1) to plot all the positive / negative examples of the case-base in the base findings space,
2) to find the super cube in the space which contains all the positive examples but any negative examples,
3) to divide the super cube into choice component cubes and
4) to generate choice component clauses from the cubes.

The algorism to solve the above problem is shown in the Figure 4.

The algorism has the input parameters (L_B: given base findings list, CB: given case-base) and the output parameters (CC: calculated choice component clauses, CB_R: residue case-base which contains examples which are conflicted to each other and are not used in the calculation).

```
Ref_a_ccc (L_B: in, CB: in, CC: out, CB_R: out) :
    /* Arrangement of the case-base */
    Sort    {example ∈ CB | example = (observations, +/-),
            o_number(example) = number of {observation ⊂ example | observation
            belongs to L_B, observation = + }}
            by the o_number into CB*
    Move    {(example1, example2) | observations(example1) = observations(example2) on
            L_B, positive_example(example1), negative_example(example2) }
            from CB* to CB_R
    /* Search of the super cube */
    Repeat until    {example ∈ CB* | positive_example(example)} = ∅
        m* = minimum of { o_number(example) | example ∈ CB*,
                                positive_example(example) }
        PCu+(m*) = {PCu = [m of L_B inc n of Li] | m = n = m*,
                        Li = (observation ∈ L_B | observation ⊂ example,
                        observation = +, positive_example(example),
                        o_number(example) = m*) }
        PCu-(m*) =      {PCu = [m of L_B inc n of Li] | m = n = m*, Li = (observation ∈ L_B |
                        observation ⊂ example, observation = +,
                        negative_example(example), o_number(example) = m*) } ∪
                        {PCu = [m of L_B inc n of Li] | m = n = m*, length of Li = n, Li ⊂
                        ( observation ∈ L_B | observation ⊂ example, observation = +,
                        negative_example(example), o_number(example) > m*) }
        PCu--(m*) = {PCu = [m of L_B inc n of Li] | m = m*, n < m*,
                        length of Li = n, PCu ⊢ PCu-(m*)}
        PCu*(m*) =      {PCu = [m of L_B inc n of Li] | m = m*, n < = m*,
                        length of Li = n, PCu ∉ PCu--(m*),
                        ¬(∃PCu' = [m' of L_B inc n' of Li'] | m' = m*, n' < m*, length of
                        Li' = n', PCu' ∉ PCu--(m*), PCu' ⊢ PCu) }
        /* Generation of the clauses */
        Repeat until    PCu*(m*) = ∅
            n* = minimum of { n | PCu = [m of L_B inc n of Li], PCu⊂PCu*(m*)}
            Lis(n*) = { Li | Li ⊂ L_B, ¬(∃combination | combination ⊂ Li,
                        length of combination = n*, combination ∈ PCu--(m*)) }
            Li*(n*) = { Li | Li ∈ Lis, ¬(∃Li' | Li' ∈ Lis, Li' ⊃ Li) }
            Cu*(m*,n*) = {Cu = [m of L_B inc n of Li] | m = m*, n = n*,
                                Li ∈ Li*(n*) }
            Add Cu*(m*,n*) to CC
            Subtract    {PCu | PCu involved in Cu*(m*,n*) in the base findings space}
                        from PCu*(m*)
        End
        Subtract    {example | example contained by ccc in the base findings space, ccc ∈
                    CC, positive_example(example) } from CB*
    End
```

Figure 4 New Algorism of a Choice Component Clause Refinement

The algorism is made up of three parts (arrangement of the case-base, search of the super cube and generation of clauses).

At the first part examples in the case-base are sorted by the number of positive observations which belong to the given base findings list. If there are examples which have the same observations on the given base findings list but some of which are classified in positive examples and other of which are in

negative examples, they are removed from the case-base and moved into the residue case-base CB_R.

The second part searches the target super cube in the following way:

1) search the primary choice component cubes (PCu*) with the choice number which is equal to the minimum number of positive observations in the positive examples
2) if positive examples remain and are not contained in the above cube, search the next primary choice component cubes (PCu*) with the choice number which is equal to the minimum number of positive observations in the remaining positive examples
3) repeat the above process until all the positive examples are contained in the union of the cubes.

The third part generates the target choice component clauses. This part is invoked in the second part for each PCu* and utilizes the theorem mentioned in 3.1 in order to unify the above PCu* and generate compact choice component clauses (Cu*).

3.3. Application to Choice Component Clauses Conjunction Refinement

Here such application of the above algorism is introduced that refines a choice component clauses conjunction as follows :

IF C1, C2 THEN C

where C1 and C2 are choice component clauses.

Man defines the base findings lists and divides them according to confirmation cost for example. He/she prepares also the case-base for the goal concept (C).The algorism is utilized as shown in the Figure 5.

```
Ref_2_ccc ( (LB1,LB2): in, CB: in, CC: out, CBR: out) :

   /* On the first base findings list */
   Ref_a_ccc(LB1, CB, CC, CBR1)
   CB1+ = {example | example ∈ CBR1, positive_example(example) }
   Ref_a_ccc(LB1, CB1+, CC1, void)
   Repeat for each c ∈ CC1
      CB2+ = {example | example ∈ CBR1,
                example involved by c in the base findings space of LB1 }
      Ref_a_ccc(LB2, CB2+, CC2, CBR2+)
      Add {(c1, c2) | c1 = c, c2 ∈ CC2 } to CC
      Add CBR2+ to CBR
   End

   /* On the second base findings list */
   [the same procedure exchanged the sufix 1 and 2 above]
```

Figure 5 Application of the Algorism to Choice Component Clauses Conjunction Refinement

Choice component clauses based on the first base findings list are searched at first. If there are such examples that conflict to each other on the first base findings list, they are refined with another base findings list and conjunctions of the choice component clauses are obtained.
The same procedure is performed for the second base findings list.

4. Discussion

4.1. Amount of Calculation

Here the amount of calculation of the algorism is estimated.
Suppose that the target choice component clause is the following :

m of L_B inc n of Li

where L_B = (F1, F2, ---, FN).

If all the possible choice component clauses are enumerated, the number of them is

$\Sigma_{m=1,N} \Sigma_{n=0,m}$ {case number of Li whose length $<= N$ } =
$\Sigma_{m=1,N} (1 + \Sigma_{n=1,m} \Sigma_{i=n,N} {}_N C_i)$ =
$N(3N+1)2^{N-3} + N$.

In this algorism "primary choice component cubes" are introduced and the amount of calculation is reduced as follows :

$\Sigma_{m=1,N} \Sigma_{n=0,m}$ {case number of Li whose length = N } =
$\Sigma_{m=1,N} (1 + \Sigma_{n=1,m} {}_N C_n)$ =
$(N+2)2^{N-1} - 1$.

As shown above the algorism has the possibility of combination explosion. In order to decrease the amount of calculation the algorism searches the target clauses in such space that the choice number is minimum at first and increases the choice number until all the positive examples are involved in the generated clauses. Therefore the amount of calculation is able to be controled by means of setting a limit of the choice number to be searched for example.
A statistical analysis is also effective to select important findings and reduce the number of the base findings.

4.2. Efficiency in the field engineering

As mentioned in 2.1 definition of the findings and preparation of the case-base are often difficult for field experts to do. In addition it is important in the field that the knowledge-base is improved based on the current KB.
The algorism is evaluated from the view point of the field engineering here.

1) Definition / modification of the base findings lists

When field experts define the initial version of the base findings lists(L_B), unnecessary findings may be involved or necessary findings may be missing. If an unnecessary finding(Fu) is involved in L_B=(F1, F2, ---, FN),

the algorism generates such clauses that the choice number is small so that the influence of Fu is weakened or the included findings lists don't contain Fu. Therefore man may find out that Fu is not so important for the goal concept. If a necessary finding is missing, the conflict of positive examples and negative examples may increase and be catched in the residue case-base(CB_R).

When the generated clauses are not so well, field experts try to add some important findings to the base findings lists(L_B) or remove unnecessary findings from L_B. The algorism is performed on the partialy ordered set of the primary choice component cubes. The partialy ordered set is influenced by the addition/deletion of a finding only in the part which correspons to the finding and some elements of the negative cubes set (PCu--) are added/removed respectively. Therefore the algorism generates the clauses which are comparable with ones before modification.

Findings are often arranged and grouped for the purpose of classification efficiency or cost. The algorism is applicable to the grouping of the base findings lists shown in 3.3.

2) Preparation / modification of the case-base

Necessary and sufficient case data is not always prepared in the field. And the quality of case data is often poor.

The algorism works well even if some observations in an example are missing, because the algorism is based on the number of the positive observations.

Even if the amount of positive examples prepared in the case-base is not sufficient, the algorism works. The reason is that the algorism searches the super cubes in the base findings space which involve not only positive examples but the possibilities which are not denied explicitly by the negative examples. But in the case of a conjunction refinement insufficient amount of positive examples may cause the target clause missing.

On the other hand when the amount of negative examples prepared in the case-base is not sufficient, the algorism generates too generalized clauses.

If an observation in the case-base has such error that positive observation is recorded as negative observation or in the reverse way, the algorism generates too generalized clauses. Some of the errors may be catched in the residue case-base(CB_R).

If the classification of examples in the case-base is incorrect, the algorism does'nt search the target clauses. It is required that the statistical analysis removes the classification error beforehand.

3) Tuning

The algorism can find such an or-type concept that is represented in different clauses with the same goal, modifies the results according to modification of the base findings lists and collects such findings into the included findings lists that may have some meaning togather.

Therefore tuning of the knowledge-base generated by the algorism is easy for field experts.

4.3. Comparison with other algorisms

Version space [4] and ID3 [5] are well known as the algorisms of rule induction from case data.
Here the algorism is compared with them especially from the view point of field engineering [Table 1].

	Version Space	The Algorism	ID3
Definition / Modification of the base findings lists *addition / deletion of findings	current result is avoided and different result is obtained	current result is modified continuously	current result is avoided and different result is obtained
*grouping of findings	not acceptable	acceptable	not acceptable
Preparation / Modification of the case-base *lack of observations in an example	not acceptable	acceptable	not acceptable
Amount of Calculation	exponential order	exponential order	polynominal order
Result *type	a conjunction of findings	choice component clauses	conjunctions of findings (tree)
*or-type concept	not acceptable	acceptable	acceptable
*tuning	impossible	by the base findings lists	impossible

Table 1 Comparison with Other Algorisms

5. Future Research

The following research is to be done in the future :

1) Combination with statistical methods
   * extraction of important findings
   * weighing of examples by occurrence frequency
2) Cooperation with man
   * elimination of unnecessary clauses in early phase
   * filling up the lack of negative examples
3) Application in the fields
   * diagnosis
   * multi-criteria selection.

References

[1] Lindberg,D. et al, "Computer-Based Rheumatology Consultant", Proc. of the third World Conf. on Medical Informatics, North-Holland, pp.1311-1315, 1980.

[2] Politakis,P. and Weiss,S., "Using Empirical Analysis to Refine Expert System Knowledge Bases", Artificial Intelligence, 22, pp.23-84, 1984.

[3] Ginsberg,A., Weiss,S. and Politakis,P., "SEEK2: A Generalized Approach to Automatic Knowledge Base Refinement", Proc. of IJCAI-85, pp.367-374, 1985.

[4] Mitchell,T.M., "Version Spaces: A Candidate Elimination Approach to Rule Learning", Proc. of 5th IJCAI, pp.305-310, 1977.

[5] Quinlan,J.R., "Induction of Decision Trees", Machine Learning 1(1), Kluwer, pp.81-106, 1986.

Rule optimization and theory optimization: Heuristic search strategies for data driven machine learning

Geoffrey I Webb, Lecturer,

Department of Computing and Mathematics
Deakin University
Geelong 3217
Australia

Previous implementations of the Aq algorithm have used rule optimization search strategies to attempt to develop optimal classification procedures. These strategies involve generating successive characteristic descriptions each of which is individually of maximal value. This is contrasted with theory optimization search strategies which, instead, generate successive complete classification procedures from which those with the maximal value are selected. These two strategies have been applied to the domain of the diagnosis of Immunoglobulin A Nephropathy disease. The theory optimization strategy was observed to out perform the rule optimization strategy.

1. Introduction

The two key components of a data driven machine learning system are its *classification language* and its *search algorithm*. The classification language is a language in which classification procedures can be expressed. This defines the *classification procedures* that the system can produce. Typical examples of classification languages are predicate logic [1] and decision trees [2].

The search algorithm takes as input a *training set*, a set of examples of correct classifications of instances from a domain. The training set is used to select one of the classification procedures that can be described by the classification language. The algorithm attempts to select a classification procedure that will correctly classify all instances from the domain from which the training set is drawn.

A search algorithm usually attempts to maximize two measures—

- the number of instances from the training set that are correctly classified by the classification procedure; and
- the intrinsic value of the classification procedure.

Intrinsic value is primarily based on the simplicity of the classification procedure. It is generally accepted that the more simple classification procedure is preferable to the more complex [3].

The relative weights of these two measures reflect the degree of noise that is expected in the data. If no noise is expected then intrinsic value is usually

used as a secondary measure to select between theories that classify the example set equally well.

Measures of the complexity of a classification procedure are not self-evident. For example, is a predicate logic statement in conjunctive normal form containing a two disjuncts that each contain three conjuncts more complex than a statement containing three disjuncts each of which contains two conjuncts? What are the relative complexities of two predicates (such as male(X) and olderthan(X, 25))?

Criteria for assigning the intrinsic value of a classification procedure and for weighing the intrinsic and the accuracy with relation to the training set are called *classification procedure evaluation criteria.*

It might be thought that, given a classification procedure evaluation criteria, the rest is trivial—a system need only search the space of classification procedures to find the procedure that maximizes the criteria. This is indeed possible for attribute-value machine learning where all attributes are discrete [4]. It is also possible where no disjunction is required [5]. However, where disjunction is required and ordinal attributes or structural relations are allowed by the language, the number of potential theories usually precludes exhaustive search. As a result, heuristic search strategies are usually employed.

The Aq algorithm [6] is a very successful data driven machine learning algorithm. It can be described as follows. The input is a set of instances divided into POS (instances belonging to the class of interest) and NEG (instances not belonging to the class of interest.) The output is a set of characteristic descriptions for the class.

Algorithm: Aq

Input: *POS* (a training set of instances belonging to the class of interest)
NEG (a training set of instances not belonging to the class of interest)
criteria (classification procedure evaluation criteria)

Output: *R* (a set of characteristic descriptions for the class.)

initialize *R* to empty.

while *POS* is not empty

randomly select an instance *i* from *POS*.

select a characteristic description *c* that covers *i* and maximizes *criteria*.

remove from *POS* all instances covered by *c*.

add *c* to *R*.

end while.

simplify *R* using both general and domain specific rules.

To obtain a complete classification procedure for a domain, the above algorithm is applied to each class in the domain, with *POS* and *NEG* set accordingly. Any new instance covered by a characteristic description so formed is deemed to belong to the class for which that characteristic description was developed.

It should be noted that the above is not a strict expression of the Aq algorithm which specifies that all most general characteristic descriptions that cover *c*

and cover no instances in *NEG* should be generated at step 2, and that step 6 should select the subset of the characteristic descriptions so generated that maximizes the evaluation criteria. However, as noted above, an exhaustive search, which is required by this latter approach, is not computationally feasible for many domains. As a result, the exhaustive search is replaced in practice by the above algorithm with a heuristic search employed at step 2 that attempts to find the same characteristic description as would be found by an exhaustive search.

The heuristic search usually employed is a *beam search*. A beam search maintains a set of a maximum size, n, which contains the best n characteristic descriptions discovered to date. This set is called the beam. Successive cycles of the search attempt to refine the candidates in the beam, replacing current candidates with new candidates as better characteristic descriptions are discovered.

Two general strategies are usually employed. The beam may be seeded by the set of most general characteristic descriptions that cover the selected example. The search then considers successive specializations of candidates in the beam [7]. Alternatively, the beam may be seeded with the most specialized characteristic description that covers the selected example. In this case the search considers successive generalizations of candidates in the beam [8].

A version of the latter strategy is the ROA algorithm (below.) Note that this algorithm constrains the search by only considering generalizations that expand the number of instances covered. This is achieved by only generalizing a rule by a *least generalization* [9] that causes the characteristic description to cover a new positive instance.

It can be seen that Aq attempts to construct the best theory by attempting to select successive characteristic descriptions each of which individually maximizes the given evaluation criteria. This is a *rule optimization* search strategy. This type of strategy is subject to the flaw that individual characteristic descriptions that rate highly will not necessarily form a classification procedure that rates highly.

An alternative strategy is to generate successive complete classification procedures and to select that which maximizes the evaluation criteria. This is a *theory optimization* search strategy. The TOA algorithm (below) is a modification of the ROA algorithm that utilizes theory optimization.

It should be noted that the strict Aq algorithm, which generates all most general characteristic descriptions that cover each case examined, is a theory optimization strategy, as an optimal theory is selected from the set of characteristic descriptions that are generated. The strict Aq algorithm differs from the TOA algorithm, however, in that the TOA algorithm generates successive complete theories and selects between these. This strategy proves to be computationally efficient even with respect to continuous attributes.

Note that when $width = 0$, both the ROA and TOA algorithms perform identically.

Algorithm: ROA (Rule optimization algorithm)

Input: *POS* (a training set of instances belonging to the class of interest)
NEG (a training set of instances not belonging to the class of interest)
criteria (an evaluation criteria)
width (an integer specifying the maximum width of the beam)

Output: *R* (a set of *characteristic descriptions* for the class.)

initialize *R* to empty.

while *POS* is not empty

randomly select an instance *i* from *POS*.

initialize *beam* with the most specific characteristic description that covers *i*.

for *x* = each instance in *POS* other than *i*, in random succession

initialize *beam** to an empty list.

for *y* = each characteristic description in *beam* in succession

add the least generalization of *y* that covers *x* to *beam**.

end for.

add *beam** to *beam*.

if *beam* contains more than *width* characteristic descriptions remove those that rate lowest according to criteria until only *width* characteristic descriptions remain.

end for.

remove from *POS* all instances covered by *c*.

add *c* to *R*.

end while.

simplify *R* using both general and domain specific rules.

2. Evaluation

To compare the algorithms they were implemented on a Sun 4/60 Sparcstation 1 and applied to the diagnosis of Immunoglobulin A Nephropathy disease. This domain was selected because it poses a difficult data driven machine learning task. Each instance is described by 37 attributes specifying relevant clinical information. For only 17 of these attributes are the values known for all instances. Only 276 instances are available. For no instance are all attribute values known. Of the 276 instances, 57 are positive and 219 are negative.

The data includes all patients for whom biopsies were conducted for renal conditions at Geelong Hospital from 1979 to 1989 inclusive. Clinical evaluation of a biopsy results in a definitive and objective diagnosis. Due to the objective nature of both the diagnosis and the attributes, the absence of noise from the data can be guarantied.

Algorithm: TOA (Theory optimization algorithm)

Input: *POS* (a training set of instances belonging to the class of interest)
NEG (a training set of instances not belonging to the class of interest)
criteria (an evaluation criteria)
width (an integer specifying the maximum number of theories to examine)

Output: *r* (a set of *characteristic descriptions* for the class)

initialize r to empty.

repeat *width*times

  initialize r^* to empty.

  while *POS* is not empty

    randomly select an instance *i* from *POS*.

    initialize *c* to the most specific characteristic description that covers *i*.

    for *x* = each instance in *POS* other than *i*, in random succession

      set c^* to the least generalization of *c* that covers *x*.

      if the value *criteria* assigns to c^* is greater than the value *criteria* assigns to *c* set *c* to c^*.

    end for.

    remove from *POS* all instances covered by *c*.

    add *c* to r^*.

  end while.

  restore *POS* to its initial value.

  if r^* rates higher than *r* according to *criteria* set *r* to r^*.

end repeat.

simplify *r* using both general and domain specific rules.

Note that the algorithms as presented do not specify the evaluation criteria that are to be employed within a search. In order to test the algorithms, they were executed using simple evaluation criteria. The intended objective of these criteria is to develop the simplest possible classification procedure that correctly classifies all instances in the training set. Simplicity is measured by the number of characteristic descriptions in the classification procedure.

The algorithm used to implement these criteria is listed below. Essentially, of any two classification procedures, one with no negative cover is preferred to one with negative cover. If neither has negative cover then the one with the highest positive cover is preferred. Of any two classification procedures with no negative cover and identical positive cover, the one with the fewest characteristic descriptions is preferred. Note, $|C|$ denotes the number of characteristic descriptions in C.

Algorithm: CPEA (Classification procedure evaluation algorithm)

Input: C (a set of characteristic descriptions)

Output: v (a value)

if any characteristic description in C covers an instance in *NEG* then set v to 0

else set v to the number of instances in *POS* covered by characteristic descriptions in C.

add $1 - \frac{1}{|C|}$ to v.

As it does not relate to the difference between the two algorithms, the final step of ROA and TOA algorithms, refinement by general and domain specific rules, has been omitted.

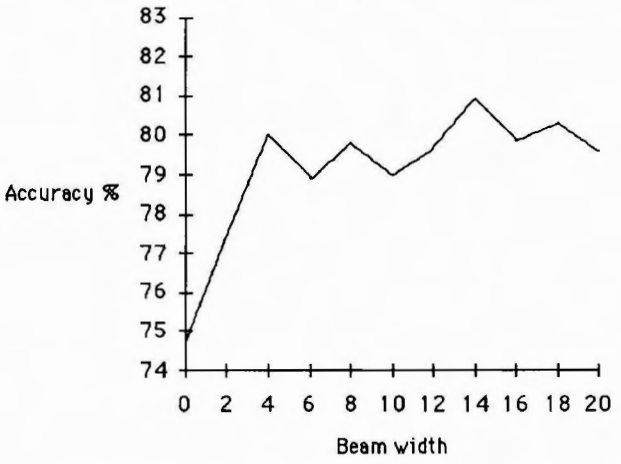
The ROA and TOA algorithms were evaluated by application to a subset of the data and then evaluating the performance of the resulting classification procedure against the remnant of the data. In each test, instances were individually randomly selected for the training set with a 75% chance of inclusion. As both algorithms are sensitive to the order of instances in the training set, the data was randomly shuffled before each use. The algorithms were applied 100 times for each even setting of *width* from 0 to 20.

To generate a complete classification procedure, the respective algorithms were applied twice to each training set, once with *POS* set to contain those instances diagnosed with Immunoglobulin A Nephropathy disease and once with *POS* set to contain all other instances. The characteristic descriptions formed were expressed as production rules and sorted in order of the number of positive cases that they covered from highest to lowest. During evaluation, the first rule to apply to a case was selected.

The language used to express characteristic descriptions supported conjunctions of conditions each relating to a single attribute. For nominal attributes a condition tests whether the value for an attribute falls within a specified set of values. A missing value is treated as a distinct value. For ordinal attributes a condition either tests whether the value of an attribute falls within a specified range or requires that it is missing.

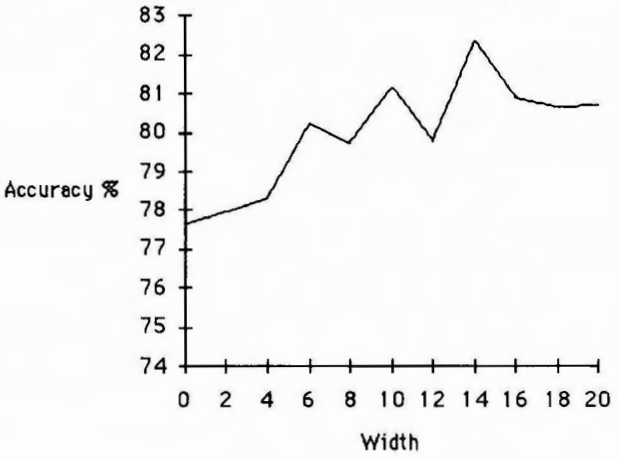
The Appendix shows the output of a run of the TOA algorithm with *width* set to 20.

Figures 1 and 2 chart the accuracies of the two algorithms at successive settings of *width*. As can be seen, as the setting of width increases, both algorithms show an initial increase in accuracy which appears to rapidly level out. The average accuracy over all runs of ROA was 79.1% and of TOA was 79.9%.



Each point represents the average accuracy over 100 runs at the specified setting of *width*.

Figure 1: Rule optimization accuracy

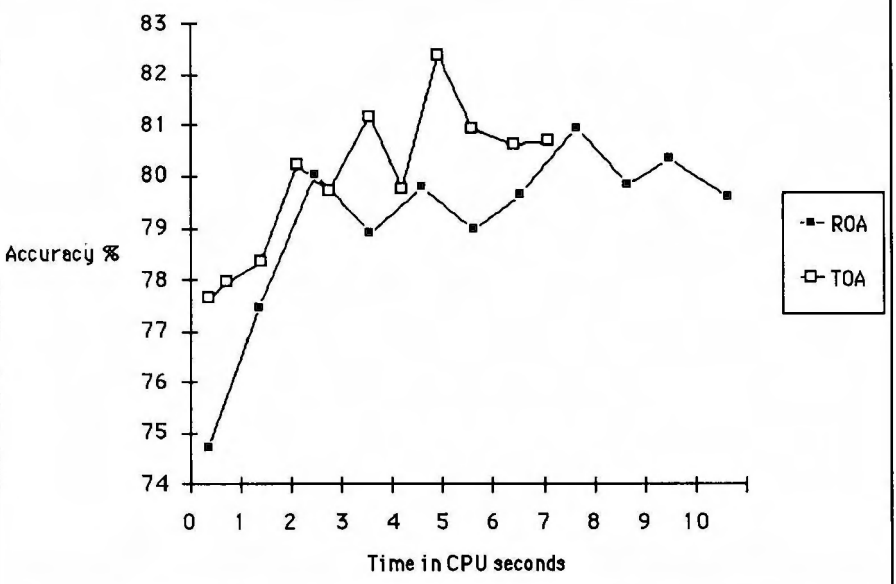


Each point represents the average accuracy over 100 runs at the specified setting of *width*.

Figure 2: Theory optimization accuracy

Although these results suggest a modest advantage for TOA over ROA, this fails to take account of the fact that identical settings of *width* are not commensurate across the two algorithms. Rather than attempting to compare performance on the basis of performance for given setting of width, a more relevant measure is accuracy against computation time. Figure 3 compares the performance of the two algorithms on this basis. This

comparison shows TOA consistently providing higher accuracy than ROA for equivalent equivalent expenditures of computation time.

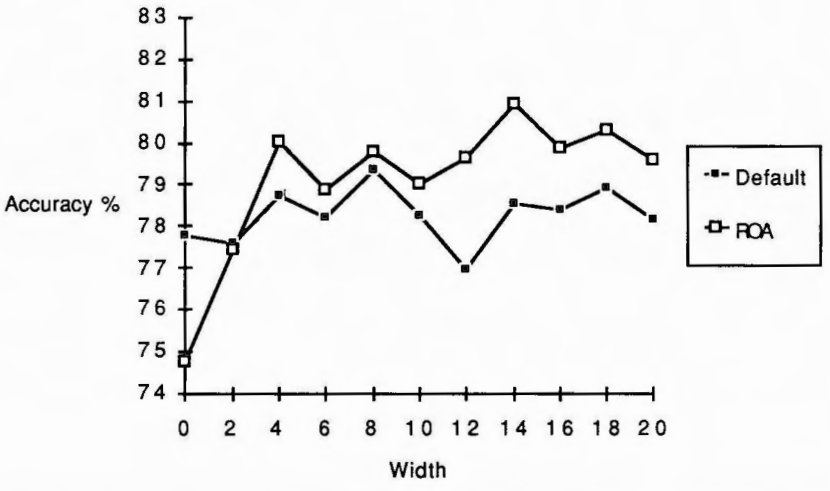


Each point represents the average accuracy and average computation time in CPU seconds over 100 runs at a single setting of *width*.

Figure 3: Comparative performance of ROA and TOA plotting accuracy against computation time.

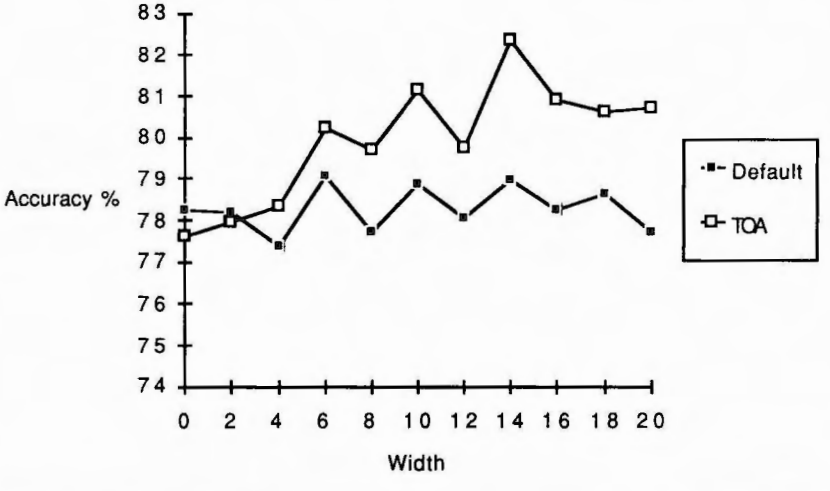
One measure of the power of an induction algorithm is to compare its performance against chance performance (the accuracy obtained by making random classifications.) With two classes, random performance would be expected to provide 50% accuracy. Both algorithms perform considerably better then this.

Another interesting measure of the power of an induction algorithm is to compare its performance against the *Default Classification Strategy* (always classifying an instance as belonging to the most common class.) This extremely simple strategy gives an indication of the difficulty of obtaining high accuracies for a domain. For each of the experimental runs of the system, the accuracy obtained by the Default Classification Strategy was measured. Figure 4 compares the performance of the Default Classification Strategy and ROA and Figure 5 compares the performance of the Default Classification Strategy and TOA. These results show that both algorithms perform consistently better than the Default Classification Strategy with settings of *width* above two.



Each point represents the average accuracy over 100 runs at the specified setting of *width*.

Figure 4: Comparative performance of ROA and the Default Classification Strategy.

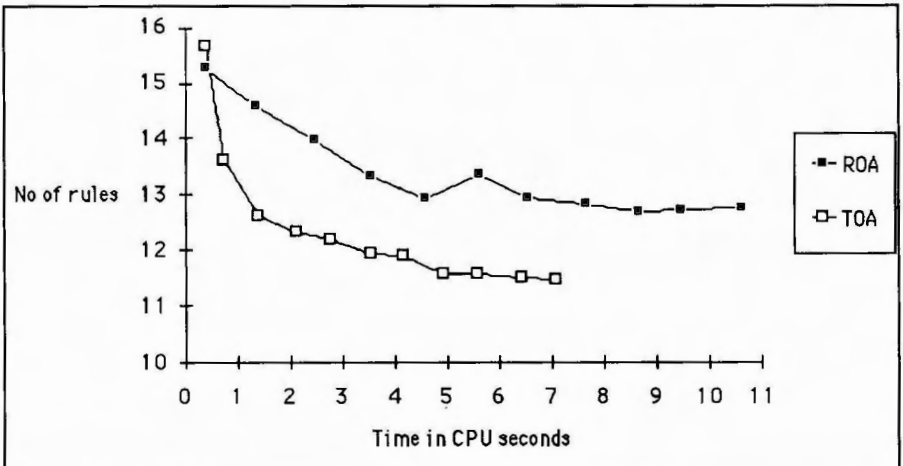


Each point represents the average accuracy over 100 runs at the specified setting of *width*.

Figure 5: Comparative performance of TOA and the Default Classification Strategy.

The accuracy of the Default Classification Strategy equals the proportion of the data withheld for evaluation which belongs to the most common class. The variations observable in Figures 4 and 5 in the performance of the strategy reflect random variations in the proportion of the most common class in the withheld data. Correspondences between variations in the proportion of the most common class in the withheld data and variations in performance of ROA and TOA that are apparent in Figures 4 and 5 suggest that this is one of the factors affecting performance of both ROA and TOA. This accounts to some extent for the ragged nature of the performance curves for the two algorithms observable in Figures 1 to 5.

A final comparison to draw between the two strategies is the number of characteristic descriptions that are produced. The evaluation criteria have been tailored to produce as few characteristic descriptions as possible on the assumption that this is a good measure of the simplicity of a classification procedure and that the simpler classification procedure is more likely to be accurate. Figure 6 plots the number of characteristic descriptions produced against computation time. As can be seen, TOA is considerably more successful than ROA in producing smaller sets of characteristic descriptions.



Each point plots the average number of characteristic descriptions produced against the average computation time in CPU seconds over 100 runs at a single setting of *width*.

Figure 6: Comparative performance of ROA and TOA plotting number of characteristic descriptions produced against computation time.

It is worthy of note that the large reduction in the number of characteristic descriptions produced by TOA has not produced such a dramatic improvement in performance. The average accuracy of all runs of TOA was only 0.7% greater than the average accuracy of all runs of ROA. By contrast, the average number of rules produced by all runs of TOA was 8.0% smaller than the average number of rules produced by all runs of ROA. Further, comparing runs that produced comparable numbers of rules (such as TOA with width two and ROA with width four), ROA produces higher

accuracy. This suggests that although TOA is considerably better at developing sets of characteristic descriptions that maximize the evaluation criteria, those criteria do not provide an ideal measure of the value of a set of classification procedure. TOA can be expected to provide even greater improvements in accuracy over ROA when the evaluation criteria employed more adequately measure the relative worth of different classification procedures.

3. Conclusion.

Previous approaches to the generation of classification procedures based on characteristic descriptions have utilized search heuristics that attempt to maximize the value of the entire classification procedure by finding a succession of characteristic descriptions each of which individually is of maximal value. This rule optimization is achieved by generating multiple candidate rules and selecting the rule with the highest value.

This strategy has been contrasted with a strategy that generates and selects between multiple candidate classification procedures. The theory optimization strategy has the advantage that the final object of interest, the classification procedure, is the direct object of evaluation. However, the possibility remains that the single candidate classification procedure generation algorithm employed will not produce sufficient high value classification procedures for the strategy to be successful.

This study has demonstrated that this possibility is not realized, at least for the domain of the diagnosis of Immunoglobulin A Nephropathy disease. For an extremely small set of data with high levels of missing values theory optimization has out performed rule optimization.

Acknowledgements

I would like to thank John Agar and Geelong Hospital for providing the data used in this study.

Bibliography

[1] Winston, P. H. Learning structural descriptions from examples. In P. H. Winston (Ed.), *The Psychology of Computer Vision*, McGraw-Hill, New York, 1975, pp. 157–209.

[2] Quinlan, J. R. Induction of decision trees. *Machine Learning*, **1**:81–106(1986).

[3] Quinlan, J. R. Simplifying decision trees. *IJMMS*, **27**:221:234(1987).

[4] Webb, G. Techniques for efficient empirical induction. *Proceedings of the Second Australian Joint Conference on Artificial Intelligence*, Adelaide, 1988, pp. 218–232.

[5] Mitchell, T. M. *Version Spaces: A Candidate Elimination Approach to Concept Learning*. PhD thesis, Stanford University, 1979.

[6] Michalski, R. S. Synthesis of optimal and quasi-optimal variable-valued logic formulas. *Proceedings of the 1975 International Symposium on Multiple Valued Logic*, Bloomington, Ind., 1975, pp 76-87.

[7] Michalski, R. S. A theory and methodology of inductive learning. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*, Springer-Verlag, Berlin, 1984, pp. 83–129.

[8] Larson, J. B. *Inductive Inference in the Variable Valued Predicate Logic System VL21: Methodology and Computer Implementation.* PhD thesis, Computer Science Department, University of Illinois, Urbana, Ill., 1977.

[9] Plotkin, G. D. A note on inductive generalization. In B Meltzer & D. Michie (Eds) *Machine Intelligence 5*, Edinburgh University Press, Edinburgh, 1970, pp. 153-163.

Appendix

The following is output generated by the TOA algorithm with *width* set to 20. Each characteristic description is expressed as a production rule. The number of positive cases covered by each rule is listed in brackets at the end of the rule.

IF 2<=AGE<=77
100<=SYST_BP<=230
60<=DIAST_BP<=120
DYSPNOEA is T
800<=U.RBC'S<=500000
0.10<=UTP<=23.13
0.10<=CRCL<=10.50
30.00<=S.CR<=680.00
3.0<=S.UREA<=30.6
0.20<=C3<=1.96
0.03<=C4<=1.21
5.4<=HB<=16.6
3.8<=WCC<=31.8
100<=PL'TS<=648
5<=ESR<=142
ANF is unknown or NEG or POS
ANTI-GBM is unknown or NEG or POS
THEN DIAGNOSIS=other [45]

IF 2<=AGE<=68
90<=SYST_BP<=160
50<=DIAST_BP<=100
DYSPNOEA is F
H'PTYSIS is F
0<=U.RBC'S<=500000
0.20<=UTP<=11.00
40.00<=S.CR<=112.00
IGG is unknown
IGM is unknown
ANF is unknown or NEG or POS
ANTI-GBM is unknown
THEN DIAGNOSIS=other [28]

IF 15<=AGE<=72
115<=SYST_BP<=210
60<=DIAST_BP<=115
H'PTYSIS is F
BOWEL_SX is F
0<=U.RBC'S<=33000
0.43<=UTP<=16.25
0.18<=CRCL<=3.23
70.00<=S.CR<=561.00
3.2<=S.UREA<=43.8
42<=TOT.PROT<=89
28<=S.ALB<=52
7.4<=HB<=20.4
4.3<=WCC<=11.0
140<=PL'TS<=280
ANF is unknown or NEG or POS
ANTI-DNA is unknown
ANTI-GBM is unknown or NEG
THEN DIAGNOSIS=other [25]

IF 15<=AGE<=66
110<=SYST_BP<=230
65<=DIAST_BP<=160
H'PTYSIS is F
RASH is F
WT_LOSS is F
1000<=U.RBC'S<=640000
0.10<=UTP<=6.79
0.50<=CRCL<=2.76
60.00<=S.CR<=1188.00
3.5<=S.UREA<=18.9
2.60<=IGA<=5.40
11.7<=HB<=17.3
3.7<=WCC<=9.8
166<=PL'TS<=299
ANF is unknown or NEG or POS
ANTI-DNA is unknown
ANTI-GBM is unknown
THEN DIAGNOSIS=IGA_NX [24]

```
IF      2<=AGE<=79
        100<=SYST_BP<=260
        60<=DIAST_BP<=150
        ANOREXIA is T
        8000<=U.RBC'S<=500000
        0.62<=UTP<=22.69
        65.00<=S.CR<=1000.00
        3.1<=S.UREA<=78.6
        0<=TOT.PROT<=74
        0<=S.ALB<=51
        8.0<=HB<=15.1
        ANF is unknown or NEG or POS
        ANTI-GBM is unknown or NEG or POS
THEN DIAGNOSIS=other [24]

IF      4<=AGE<=78
        80<=SYST_BP<=210
        50<=DIAST_BP<=120
        LOIN_PAIN is F
        ANOREXIA is F
        WT_LOSS is F
        0<=U.RBC'S<=500000
        2.44<=UTP<=28.97
        30.00<=S.CR<=938.00
        3.7<=S.UREA<=32.3
        9.5<=HB<=16.6
        4.6<=WCC<=13.6
        187<=PL'TS<=590
        ANF is unknown or NEG or POS
        ANTI-GBM is unknown or NEG
THEN DIAGNOSIS=other [21]

IF      11<=AGE<=76
        105<=SYST_BP<=200
        70<=DIAST_BP<=115
        H'PTYSIS is F
        BOWEL_SX is F
        5000<=U.RBC'S<=500000
        0.31<=UTP<=2.56
        0.52<=CRCL<=2.12
        50.00<=S.CR<=200.00
        63<=TOT.PROT<=76
        29<=S.ALB<=47
        S.GLUCOSE is unknown
        11.3<=HB<=17.5
        7.0<=WCC<=12.5
        164<=PL'TS<=436
        1<=ESR<=118
        ANF is unknown or NEG
        ANTI-GBM is unknown or NEG
THEN DIAGNOSIS=IGA_NX [17]

IF      8<=AGE<=58
        120<=SYST_BP<=180
        65<=DIAST_BP<=100
        OEDEMA is F
        DYSPNOEA is F
        H'PTYSIS is F
        22000<=U.RBC'S<=500000
        0.15<=UTP<=1.47
        0.60<=CRCL<=3.58
        47.00<=S.CR<=138.00
        2.5<=S.UREA<=7.9
        7.90<=IGG<=14.40
        0.90<=IGA<=3.20
        0.34<=IGM<=2.60
        S.CHOL is unknown
        S.TRIGLY is unknown
        11.2<=HB<=16.3
        4.4<=WCC<=16.0
        174<=PL'TS<=550
        2<=ESR<=41
        ANF is unknown or NEG
        ANTI-GBM is unknown or NEG
THEN DIAGNOSIS=other [13]

IF      13<=AGE<=55
        110<=SYST_BP<=180
        70<=DIAST_BP<=100
        HEADACHE is F
        RASH is F
        ANOREXIA is F
        BOWEL_SX is F
        112000<=U.RBC'S<=500000
        0.13<=UTP<=2.00
        0.85<=CRCL<=3.32
        52.00<=S.CR<=270.00
        3.9<=S.UREA<=12.0
        S.CHOL is unknown
        S.TRIGLY is unknown
        S.GLUCOSE is unknown
        11.0<=HB<=15.7
        5.6<=WCC<=8.8
        244<=PL'TS<=467
        3<=ESR<=58
        ANF is unknown or NEG or POS
        ANTI-GBM is unknown
THEN DIAGNOSIS=IGA_NX [8]
```

```
IF      6<=AGE<=72
        SEX is M
        90<=SYST_BP<=200
        55<=DIAST_BP<=120
        RASH is F
        BOWEL_SX is F
        5000<=U.RBC'S<=114000
        0.22<=UTP<=1.60
        50.00<=S.CR<=1083.00
        3.4<=S.UREA<=52.1
        ASOT is unknown
        S.CHOL is unknown
        S.TRIGLY is unknown
        S.GLUCOSE is unknown
        8.1<=HB<=13.5
        3.2<=WCC<=24.6
        100<=PL'TS<=433
        ANF is unknown or NEG
        ANTI-DNA is unknown
        ANTI-GBM is unknown or NEG
THEN DIAGNOSIS=other [7]

IF      AGE=11
        SEX is M
        110<=SYST_BP<=240
        55<=DIAST_BP<=130
        HEADACHE is T
        DYSPNOEA is F
        H'PTYSIS is F
        U.R.T.I. is T
        ANOREXIA is T
        BOWEL_SX is T
        300000<=U.RBC'S<=500000
        0.30<=UTP<=0.50
        0.27<=CRCL<=1.47
        100.00<=S.CR<=320.00
        17.8<=S.UREA<=24.0
        8.80<=IGG<=22.00
        1.70<=IGA<=2.85
        1.10<=IGM<=1.22
        0.20<=C3<=2.08
        0.18<=C4<=0.60
        S.CHOL is unknown
        S.TRIGLY is unknown
        S.GLUCOSE is unknown
        10.5<=HB<=11.0
        7.4<=WCC<=8.4
        460<=PL'TS<=608
        36<=ESR<=55
        ANF is NEG
        ANTI-DNA is unknown
        ANTI-GBM is unknown
THEN DIAGNOSIS=other [2]
```

A COMBINED APPROACH TO TEXT RETRIEVAL USING CONCEPT NETWORKS LINKED TO HYPERTEXT

Pádraig Cunningham, Hitachi Dublin Laboratory
Hiromichi Fujisawa, Hitachi Central Research Laboratory, Tokyo
Lucy Hederman, Trinity College Dublin
Fred Cummins, Trinity College Dublin

Hitachi Dublin Laboratory
Trinity College
College Green
Dublin 2
Ireland

Knowledge based approaches to information retrieval from full text databases require the generation of a concept network reflecting the information in the document database. A robust solution to this knowledge acquisition task is probably beyond the scope of current natural language processing and machine learning techniques. This paper presents an assessment of this knowledge acquisition task and proposes a knowledge based text retrieval system incorporating a concept network linked to a hypertext representation of the documents. The argument is that giving the user access to the text through the links to hypertext allow the user to use his own tacit knowledge to resolve problems of inference about the text. The concept network and the reference links in hypertext will be produced by bottom up linguistic analysis and top-down contextual analysis.

1. INTRODUCTION

Classical information retrieval systems [1] typically respond to users queries with a set of references to documents which are of potential interest to the user. The system will have stored the documents and indexed them. This involves identifying the important terms in the document manually or possibly automatically (usually with statistical methods). User queries then specify the index terms which interest them and the system attempts to match the term in the query with documents it has indexed. Such systems are well established. Research continues on automating the indexing process and improving the match between a user's query and the retrieved documents.

Other systems, coming from the AI and natural language understanding areas, attempt to understand most of the incoming material and to represent what it has understood in a knowledge base which can be queried directly. For example the SCISOR project [2], [3] aims to partially parse and understand short newspaper stories in the domain of corporate takeovers. It extracts key pieces of information from these stories and answers questions about this information.

The first kind of system typically suffers from inflexibility: the set of index terms is crucial to retrieval, and if the user cannot identify the terms which have been used to describe his area of interest he fails. Also typically these systems do not allow a user to learn about the domain during the retrieval process by discovering relationships in the knowledge base.

The second kind of information retrieval system requires the generation of a concept network reflecting the information in the document database. Research on the task of generating concept networks to reflect information in textual databases has been under way for some time (see [4], [5] and [6] for instance). It is evident that generating this concept network by hand is not a realistic option for a document database of any size. On the other hand the current state of natural language processing (NLP) techniques is not adequate to the task of automatically generating the concept network from text. A compromise is the interactive concept indexing approach presented in [7] and [6] where the machine facilitates the indexing process and allows for human supervision of the text analysis and indexing .

With the approach described here, information from documents in the system is presented to the user in a number of different ways, each of them useful, and each supplementing the other. A reasonable amount of automatic natural language understanding (NLU) is required. This results in a certain amount of structured information (knowledge) which can be browsed. There is also a hypertext representation of the documents that has been generated in conjunction with the generation of the concept network. These two networks are interlinked. The idea is that the user will be able to retrieve the appropriate text using a basic concept network coupled with a hypertext representation of the document; this will take advantage of the tacit knowledge of the user to resolve the more complex NLU problems in the text.

In this paper we will examine the problems of natural language understanding (NLU) associated with the automated generation of a concept network. Our discussion is based on an analysis carried out on product reviews from various computer magazines and on patent documents. These texts are relatively straightforward but nevertheless present some formidable NLU problems. We will consider how the prospect of retrieving information from hypertext rather than from text reduces the demands on the concept indexing process. The central argument is that the user has the required tacit knowledge to resolve the more difficult NLU problems and the inclusion of the hypertext in the knowledge based text retrieval system will allow the user access to the relevant text.

There is still the significant knowledge acquisition problem of building an adequate concept network. There is also the auxiliary problem of constructing a hypertext representation of the documents and creating meaningful reference links in this hypertext and the all important links between the concept network and the hypertext. The knowledge acquisition process for producing the concept network and the reference links in hypertext is a bottom-up linguistic analysis coupled with a top-down conceptual analysis (cf. [2]).

2. THE ROLE FOR TACIT KNOWLEDGE IN AI

Central to our approach is that the general problem of designing effective text management systems is greatly relieved by drawing on the tacit knowledge of the end user. This is true in general for AI systems. The case for tacit knowledge in expert systems was made in [8] and was presented using the following metaphor.

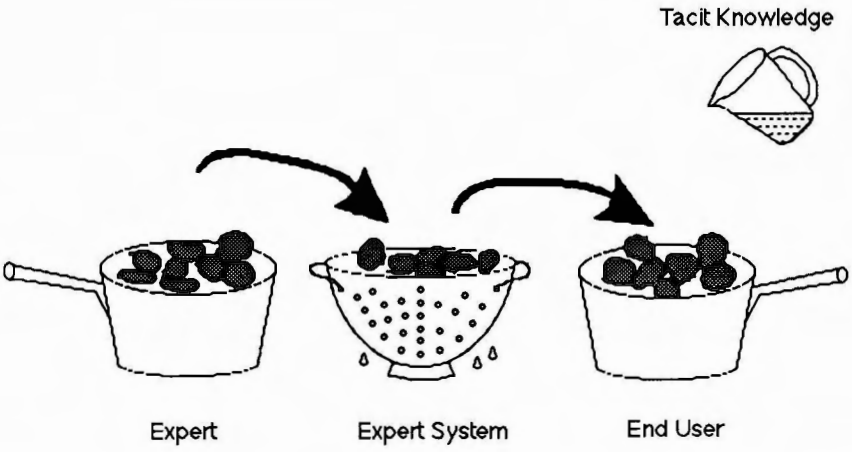


Fig. 1 The chicken soup metaphor for tacit knowledge in expert systems.

The knowledge that a human expert has is made up of factual knowledge about the problem domain and also tacit or implicit knowledge about practices, interrelationships etc. The whole thing is like a pot of chicken and dumpling soup; the dumplings representing the factual knowledge, and the tacit knowledge being the soup (see Fig. 1). The expert system is a colander that can transport this factual knowledge but the tacit knowledge cannot be contained. The whole process works because the end user supplies this tacit knowledge. The argument made by Collins et al. was that effort should not be invested in developing systems for capturing this tacit knowledge. Instead systems should be designed to facilitate this coupling of the factual knowledge with the tacit knowledge of the end user.

This is also true for intelligent text management systems where a lot of effort can be invested in inferencing mechanisms or methods for understanding metaphor. A simple alternative is to provide a structure to draw on the tacit knowledge of the user to resolve these NLU problems that are difficult for the machine. Our contention is that if documents are represented as hypertext in a concept network based text management system then the text will be accessible to the user who can make the correct inferences provided there are useful links between the concept network and the hypertext.

3. KNOWLEDGE ACQUISITION

The process of knowledge acquisition (KA) for text retrieval attempts to produce conceptual structures representing the information contained in the text. For our purposes the conceptual representation is a concept network the nodes of which are frames linked by (canonical) relationships, and a hypertext network in which the nodes are chunks of natural language text and the links are structural and referential.

The knowledge acquisition process can be divided into two stages. The first, *natural language understanding* (NLU), uses linguistic (syntactic and semantic) analysis to produce a deep structure representation from text.

The second stage, which can be called *concept indexing*, produces conceptual structure (i.e. extends the concept network) from a collection of deep structure predicates. In general,

many deep structure predicates might result in a small number of additional concepts and conceptual relationships.

As an example of the concept indexing stage consider the following sentence from a product review of the Apollo 2500:

> "The minimum configuration includes a serial communications port, but no parallel port."

The deep structure for this sentence would be something like:-

```
(INCLUDE    SUBJECT =  min configuration [of Apollo 2500]

            PART    =  <serial communications port,
                        ¬parallel port>)
```

The concept indexing stage would add "serial communications port, ¬parallel port" to the "has-components" slot of the "Apollo 2500 - min. configuration" concept frame. This is relatively straightforward if it is assumed that "includes" maps onto the generic relationship "has-components".

Another example, from the domain of chemical patent documents is the following passage, from Irish patent # 19647 :

> "In accordance with the present invention a process is provided of producing an antibiotic substance defined as *aureomycin* ..."

This passage contains three facts, for our purposes - that the invention includes a process, that this process produces aureomycin, and that aureomycin is an antibiotic substance. The knowledge acquisition process would add two new objects to the concept network's subtype hierarchy - "aureomycin" as a subtype of "antibiotic" and "process #19647" as a subtype of the generic "process" concept. Furthermore, "process #19647" is inserted in the "has-part" slot (attribute) of the concept frame for "invention #19647". (The invention concerns both a substance and a process; the sentence we are discussing here concerns only the process.) Finally, the "produces" slot of the frame for the "process #19647" concept is given the value "aureomycin".

The effectiveness of the knowledge acquisition process can be improved by making domain knowledge available to the process. Consider the following sentence from the product review of the Apollo 2500:

> "The system can hold two internal 104 megabyte SCSI hard disk drives"

A mechanism that recognises SCSI hard disk drives might place this information in a slot labeled "storage". The best weaker translation or inferencing methods can be expected to produce is perhaps a "has-option" relationship. It is evident that the strong methods produce the better representation - a representation closer to the handcrafted representation of a human expert. However the weak methods would be more general and would not require the generation of 'shell concept networks' for each domain.

In the examples above we have looked at relatively straightforward text and seen what kind of canonical conceptual relationships can be extracted from them using currently available techniques. In the next section we will investigate some of the things which present difficulties for current NLU techniques.

In the product reviews analysed during the process of this research it appears that, using relatively straightforward techniques of translating predicates to canonical relationships, enough of the text could be translated to a concept network to support useful browsing.

This is because much of the useful information in such texts is contained in simple declarative sentences. It is also true that much of the text in reviews of this type is preamble or padding and contains little information that should be included in the concept network. eg.:-

"Like fine wine some Macintosh software gets better with age"

3.1 Problems for NLU

The examples of the previous section reflect situations where facts are stated simply and explicitly in the text so the correct conceptual structures can be easily created. Problems occur when the language is not simple or when the statement is not explicit. Two recognised problem areas in NLU are resolving inference and understanding metaphor.

Inference is the process by which the reader comes to believe an assertion as a result of reading a text but which was not previously believed by the reader and was not stated explicitly in the text [9]. Take the following two statements for example:-

"For the base price of about $3900, you get a 20-MHz Motorola 68030 processor with a 20-MHz 68882 math coprocessor."

"The minimum configuration also includes a serial communication port but no parallel port."

We have already decided that the minimum configuration is a sub-class of the Apollo 2500 but it is by inference that we decide that the first statement refers to the same configuration. This inference is not straightforward because it involves equating the 'base price' with the 'minimum configuration'.*

Metaphors also convey information implicitly rather than explicitly. Even non-literary texts, such as product reviews, technical reports, etc, are often liberally peppered with metaphors. Here are some examples from recent product reviews:

"Clearly, the RasterPro requires less babysitting than a pen plotter."

BYTE June 1990, "DaVinci does it Penlessly"

". . . be prepared to watch PostScript files fly. The StellaPS uses Intel's i960KB RISC chip, an embedded microprocessor that eats PostScript commands for breakfast."

BYTE June 1990, "FastFonts"

". . . and it is slow as molasses compared to the StellaPS."

BYTE June 1990, "FastFonts"

Techniques for understanding metaphors are still a subject of research ([10]). We have found, however, that restriction of our domain to the processing of patents has, to some

---

* It is worth mentioning that writing style makes this problem pervasive. Rather than use the same term repeatedly in a passage authors search for alternative equivalent terms or idioms for the sake of variety. Thesauri are often used to discover equivalent terms and perhaps could also be used to resolve references of this type ('minimum' appears as an equivalent for 'base' in WordFinder from Microlytics).

extent, side-stepped this problem, as full-blown metaphor rarely occurs in the particular style of language there prevalent.

Many of the problems encountered in natural language understanding will not fit readily into the categories of inference and metaphor but are due to the general complexity of the language. Two examples are :

> "Apple has finally entered the portable computer sweepstakes with its introduction of the Macintosh Portable, a hefty wedge of computing power that is all Macintosh"
>
> "Working with a 2500 isn't quite like anything else"

What we propose is to "give up" on creating concepts for difficult passages, but to give the user access to the text directly and allow him to use his own tacit knowledge to understand them. The text of all documents in the system will be provided in a hypertext network. Links from the concept network to the hypertext and reference links within the hypertext will bring the user to the passage from which the concept was extracted, and to related passages.

4. HYPERTEXT

Hypertext is a generic name for the non-linear representation of text by computer. This is in contrast with the linear or sequential format imposed by the printed media, and allows more flexible organisation during both document creation and retrieval. Printed documents contain the rudiments of non-linear organisation in the use of such conventions as footnotes, references, parentheses etc.

An overview of a number of hypertext implementations is given in [11], and common to all these forms is the use of nodes, interconnected by links. The nodes typically contain blocks of text or structural information about the document, while the links are manifestations of different logical and structural connections between these blocks.*

As an organisational tool during document creation, hypertext offers the author a rich environment in which each node can be used to contain a single concept or argument, and which can be linked arbitrarily to other nodes. This may facilitate the thinking process and help to structure the argument in the document, and thus provide a problem solving tool (see the claims for IBIS in [12] for instance).

Of more relevance to our work, however, is the opportunity provided by hypertext as a document browsing tool, and, correspondingly, as an aid in the retrieval of information. Its usefulness depends critically both on the richness of the links within the document and on the quality of the graphical interface to the system. Some feel for document browsing using hypertext can be gained from Fig. 2, where the text is taken from the description of an invention in a patent specification, the structure of which is shown in Fig. 3.

---

* Many commentators talk of hypermedia rather than hypertext. However there is no conceptual difference. Catering for other media in hypertext means that some of the nodes contain graphics or pictures rather than text. There will be a difference for purposes of I/O, but if these different nodes are represented as objects, as is likely, then they can take care of their own I/O. For this reason the term 'hypermedia' is rarely used and 'hypertext' is considered to include media other than text.

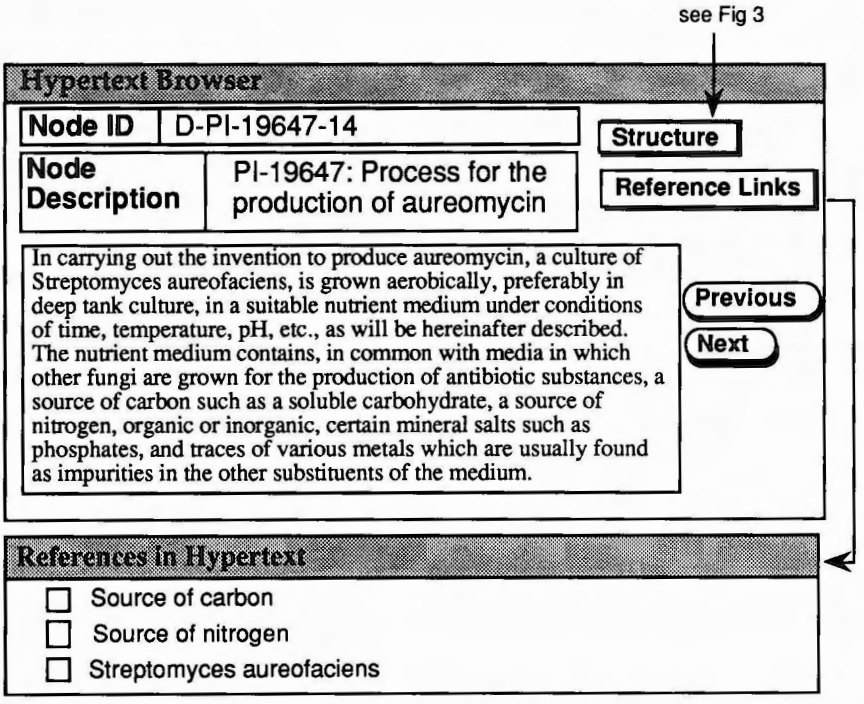


Fig. 2 An example of a node browser in hypertext.

This node illustrates the two main types of links within hypertext. Organisational links are exemplified by the previous and next node buttons, and skimming through the document using these is exactly equivalent to linear browsing, as with a printed document. In this case the next node button will bring up the next paragraph in the document. A menu of reference links from this node is found by clicking on the appropriate button, and this gives access to the non-hierarchical links to other nodes containing salient information on selected topics mentioned in this source paragraph. A suitable environment would simultaneously provide a representation of the document structure as an orientation guide.

4.1 Hypertext nodes

The most common form of node typically contains a section of text, a diagram or a picture. Important design decisions about the granularity of the text nodes need to be made before retrospective construction of a hypertext representation of a document is undertaken. The decomposition of a document into meaningful units for representation as nodes is not a straightforward task; one initial approach may be to use paragraphs or other immediately obvious structural units as nodes. This may be augmented by the provision of composite nodes containing subnodes. For example, one logical node at the end of many documents is a 'references' section, in which each reference may also suitably be a node in its own right. When working within the domain of documents with a large amount of predictable internal structure, such as patents, legal contracts, etc, heuristics may be developed to identify structural units which should suitably constitute nodes.

A partial hypertext representation of the structure of the document of Fig. 2 is shown below. As with many documents of fixed format (technical reports, contracts, statutes etc), a patent document contains a great deal of internal structure which naturally lends itself to this form of presentation, and which can be availed of in browsing through the network.

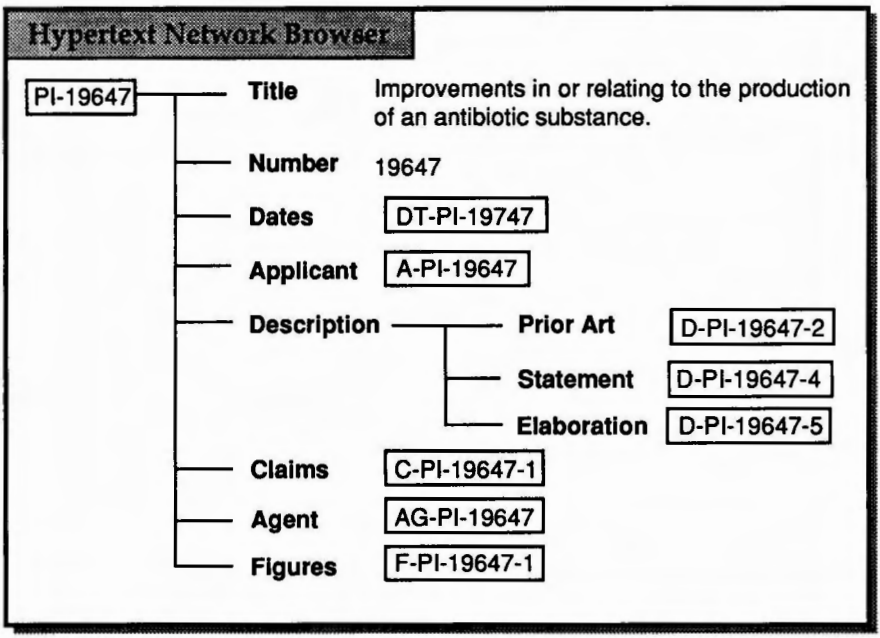


Fig. 3 The structure of a patent document

4.2 Hypertext links

Different hypertext systems provide different link types, but most will fall into one or other of the following categories:-

Organisational links: Most commonly, these are hierarchical links, similar to the 'is-a' relationships of semantic nets, or they show partonomic structure. Organisational links are based on document structure, and the tree structure formed by hierarchical links is a minimal requirement for hypertext systems. The links shown in Fig. 3 are all hierarchical, but further structural links (e.g. between the description of specific examples and figures showing those examples) may easily be established for particular types of document. Organisational links may also include inter-document links within a database, for example from the references section to the documents cited therein.

Reference links: The power of hypertext as a browsing and retrieval tool depends to a large extent on the reference links, such as are shown in Fig. 2. The node browser should ideally be able to navigate forward and backward along these links. Machine creation of meaningful reference links is a similar task to the more general problem of concept identification and indexing, and it is envisaged that, in our system,

the generation of the concept network and the creation of reference links in the hypertext should be done in parallel when the hypertext is generated from a source document. Reference links will probably only connect nodes within one particular document, in contrast to the concept network, in which the network of links is essentially independent of the document database.

Semantic links: A semantic link is like a reference link but the link itself has semantic content. The simplest semantic links will be *support* and *refutation* links indicating nodes that support or refute an argument. Links like *responds-to, questions, supports, objects-to, specializes, generalizes, refers-to, replaces*, may be useful. Manual analysis of technical reports, patent documents and magazine articles indicates that semantic links are difficult to identify *post hoc*. They are of use principally in the authoring of texts and the use of hypertext as an problem solving tool. For this reason our focus of attention has rather been on reference links.

4.3 Retrospective generation of hypertext

An outline architecture for an intelligent filing system incorporating hypertext is shown in Fig. 4. This is based on the system architecture described in [13] so the overall details will not be described again here. Instead this description concentrates on the impact of the additional hypertext representation on the overall architecture.

1. The first stage in the filing process is the image capture.

2. Before the image is stored it is analysed to identify the regions containing pictures, graphics and text. This is necessary before the image is stored, because different compaction methods are appropriate for the different types of region. Optical character recognition will be performed on the text so that it can be stored as ASCII. (See [5] and [14] for more details on document structure analysis)

3. Once the logical structure of the document has been identified a rudimentary hypertext structure for the document can be created. This structure will initially contain hierarchical links only and the nodes will be paragraphs of text, chapter headings, etc. as well as diagrams, pictures, and tables.

4. Once the text is machine readable an index is created and stored.

5. The knowledge acquisition process will be performed on the textual nodes in the hypertext. The main task for the knowledge acquisition process will be to create the concept network. The acquired knowledge will also be used to enrich the hypertext by creating new reference links.

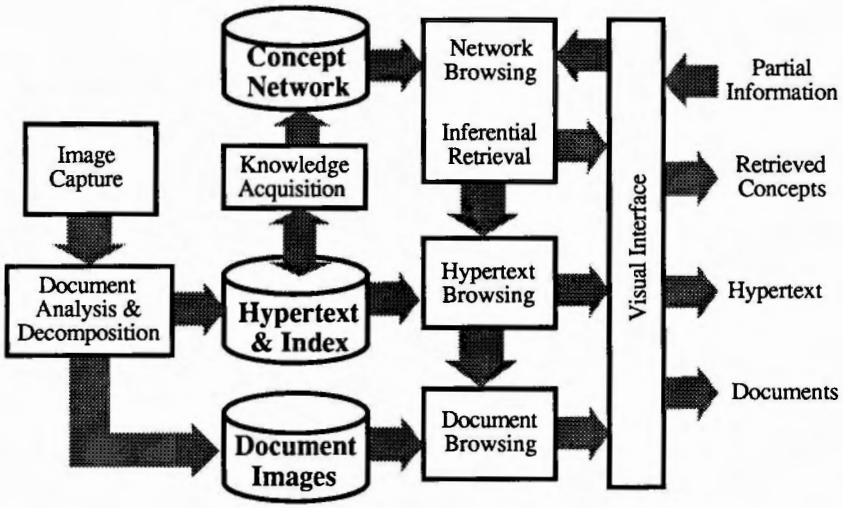


Fig. 4 A complete intelligent filing and retrieval system incorporating hypertext

This retrospective generation of a comprehensive hypertext with hierarchical and reference links is a key feature. This hypertext is an important component of the information retrieval process. Information retrieval is based on two networks: the hypertext, and the concept network as shown in Fig. 5. The concept network contains information that spans many documents while the hypertext is a non linear representation of each document with occasional references to other documents. There are some parallels between the networks as both networks contain substantial information on the nodes and also the links between nodes are hierarchical links or reference links. This means that the same network browser and node browser tools can be used to display both networks.

It is important to emphasise the conceptual differences between the hypertext network and the concept network. The nodes in the hypertext contain either structural information about a document, or text, diagrams, etc. from that document. The nodes of the concept network, however, are frames, or similar structures, representing concepts, abstract or concrete, without being associated with a particular document.

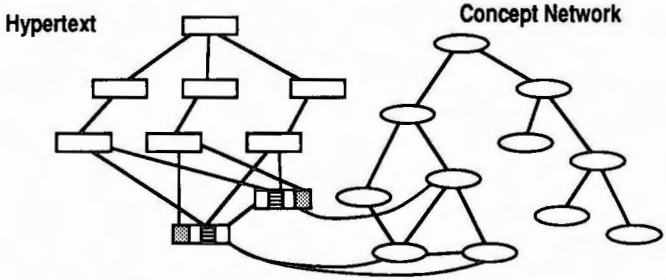


Fig. 5 The hypertext network and the concept network.

4.4 Problems solved by hypertext

The interlinking of hypertext with the concept network will allow the user more easily to identify relevant documents and document parts, and thus help to avoid many of the inherent problems associated with the construction of an elaborate representation of the conceptual structure of a document. In constructing the concept network, we are thus restricting our goal to a process of concept indexing, while we regard the natural language of the hypertext as the underlying form of knowledge representation.

5. AN EXAMPLE OF THE RETRIEVAL ENVIRONMENT

Fig. 6 shows a screen from our Concept Network/Hypertext text retrieval system. In this example the text being retrieved is from a patent of a process for producing an antibiotic. In this system the two windows on the top show information from the concept network and the two on the bottom show the hypertext. The other small window shows a list of references in the hypertext for a selected concept. The example starts with the Concept Network Browser in the top left corner. Each node represents a concept and the links shown are sub-class or instance links. Other relationships like 'has-part' or 'produced-by' are best traced using the frame browser (in the top right hand corner).

In the Concept Network Browser the user is browsing through a taxonomic hierarchy using the 'Display Children' option. This option causes the sub-classes and instances of the selected node to be displayed. The interesting concept AUREOMYCIN is discovered and the user selects the 'Display Concept' option. The system displays the frame for AUREOMYCIN in the Frame Browser window. The user can follow other links in the concept network by mousing on concepts shown in this window - mousing on GRAM-POSITIVE would bring up the frame for the GRAM-POSITIVE concept in the Frame Browser window. If AUREOMYCIN is a concept of interest to the user then the user can request a list of hypertext references that will be shown in the small References window. When one of these references is selected the text is displayed in the Hypertext Browser window.

Retrieving the text as hypertext has two major advantages for the user. The user can request the document structure to be displayed in the Hypertext Network Browser window and use that as a map to jump to another portion of the document - the Claims section for instance. Alternatively the user can navigate through the document by following either the structural links (previous and next nodes) or the reference links. If the user selects 'Reference Links' in the Hypertext Browser a list of nodes that elaborate on issues from this paragraph will be displayed in the References window.

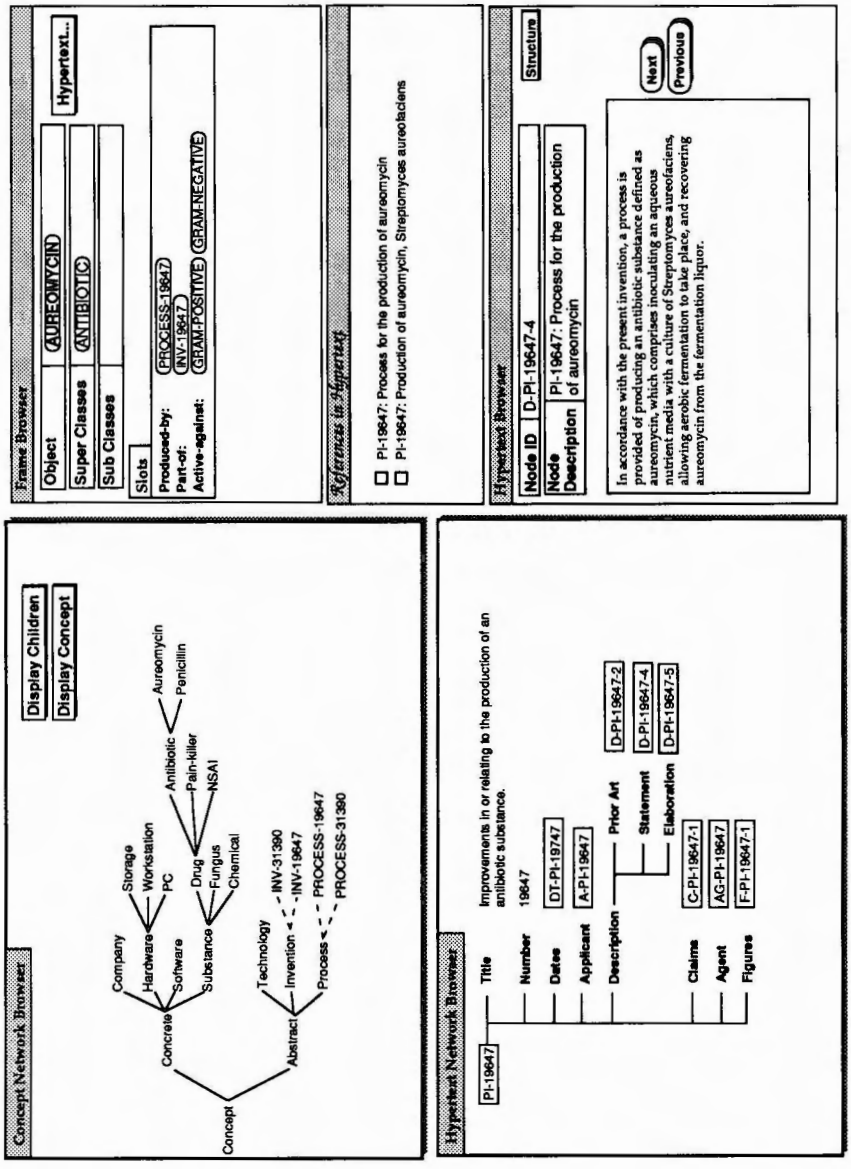


Fig. 6 A sample screen from the Hypertext/Concept Network for text retrieval.

6. PROBLEMS AND CONCLUSIONS

The main argument of this report is that the problem of knowledge acquisition associated with knowledge based text retrieval can be relieved by using concept networks linked to hypertext. The main motivation of this linking to hypertext is to allow the user access to text on which NLU or knowledge acquisition techniques fail.

Two problems that are evident with this approach at this stage are; the retrospective generation of hypertext, and the need to recognise that conceptual relationships cannot/should not be generated for a particular passage of text. In the first case, we have seen problems associated with determining suitable units of text to be used as nodes. Also, the difficulties inherent in the construction of reference links are similar to those of the more general concept indexing task.

The second problem concerns the idea that an adequate concept index can be produced by successful parsing and analysis of only a portion of the text in a document. This requires that the processing is concentrated on the salient passages of text, on the identification of the important concepts in that text and that effort is not wasted on text where the system is making no progress. These associated problems of salience and termination are relieved somewhat by the rigid structure of patents, the documents we are currently considering. A patent usually starts with a statement of the field of invention. It also has a section containing a statement of invention and a claims section listing the particular claims to novelty of the invention. Processing can be concentrated in these areas where the salient concepts can be identified. Unfortunately this is a particular advantage of patents and many document types will not contain these structural clues to salient information.

Our immediate objective is the evaluation of the effectiveness of these techniques using patents documents and in the future we will assess the appropriateness of these techniques in other domains.

Acknowledgement

We would like to thank Nobuo Hataoka for his comments on this paper and his contributions during the course of the research.

7. REFERENCES

[1] Salton, G., McGill, M.J., Introduction to Modern Information Retrieval, McGraw-Hill, 1983

[2] Rau, L.F., "Information extraction and text summarization using linguistic knowledge acquisition", Information Processing & Management, Vol.25, No.4, pp419-428, 1989.

[3] Jacobs P. S., Rau L.S., " The GE NLToolset: A software foundation for intelligent text processing", COLING-90 , 1990

[4] Sowa J.F., Conceptual Structures, Addison Wesley.

[5] Fujisawa H., et al., "Document analysis and decomposition method for multimedia contents retrieval", Proceedings of the Second International Symposium on Interoperable Information Systems (INTAP), pp231-238, 1988.

[6] Parsaye K., Chignell M., Khoshafian S., Wong H., "Intelligent Databases" Wiley, 1989.

[7] Humphery S.M., "A knowledge-based expert system for computer-assisted indexing, IEEE Expert, pp25-38, Fall 1989.

[8] Collins H.M., Green.R.H., Draper R.C., "Where's the expertise? Expert systems as a medium of knowledge transfer", in Expert Systems 85, ed. Martin Merry, Cambridge University Press, 1985.

[9] Norvig P., " Marker passing as a weak method for text inferencing", Cognitive Science, pp569-620, Vol. 13, No.4, 1989.

[10] Martin J.H., "Understanding new metaphors", IJCAI '87, pp137-139, 1987.

[11] Conklin J., "Hypertext: An introduction and survey", IEEE Computer, pp17-41, Sept. 1987.

[12] Ritter H., Webber M., "Dilemmas in the general theory of planning", Policy Sciences, Vol.4, 1973.

[13] Fujisawa H., Hatakeyama A., "Intelligent filing system with knowledge-base", Hitachi Review, pp323-328, Vol. 37, No.5, 1988.

[14] Yashiro H., et al., "A new method of document structure extraction using generic layout knowledge", International workshop on industrial applications of machine intelligence and vision (MIV-89), pp282-287, 1989.

Knowledge Acquisition with a special Hypertext System

H.Langendörfer U.Schreiweis M.Hofmann

TU Braunschweig
Institut für Betriebssysteme & Rechnerverbund
Bültenweg 74-75
D-3300 Braunschweig
Fed. Rep. Germany

Tel.: [+49] 531 391 3293
UUCP: ..mcsun!unido!tubsibr!schrei.uucp

Knowledge acquisition is a crucial issue in developing knowledge-based systems. In the approach described below, we are trying to support a knowledge-based application by the special features of our hypertext system CONCORDE. Beyond these features are typed links, node classes, local user contexts, and constraints. We are convinced that our integrated approach of collecting knowledge, managing sources of this knowledge, supporting its structuring, and the facility to maintain the knowledge by hypertext links is an important step to the integration of databases, expert systems, and hypermedia systems.

1 Introduction

The process of structuring the knowledge of experts is often seen as the most critical issue of the development of *expert systems*. We are convinced that the new type of information system called hypertext system can fill an existing gap of today's knowledge engineering. This gap exists between the very early work of a knowledge engineer (e.g. reading basic sources) and the final structuring of the expertise.

Hypertext resp. *hypermedia systems* are able to relate information of different data types in a flexible manner [8]. Information is divided into isolated portions; these portions are called *nodes*. Individual relationships between node instances are expressed by *links* [6]. A hypertext system is not only a system storing and managing this kind of information; a real important part of any system is the *presentation* of nodes and links [16].

This paper sketches a framework supporting the knowledge engineering process based on the hypertext system CONCORDE[1], a research prototype with special features of user-support.

[1] Connected Card-oriented Entities

2 Knowledge Acquisition

Knowledge acquisition is a crucial stage in the development of an expert system. As a process, it involves eliciting, analyzing and interpreting the knowledge that a human expert uses when solving a particular problem and transforming this knowledge into a suitable machine representation. Knowledge acquisition is crucial since the power and utility of the resulting expert system depend on the qualitity of the underlying representation of expert knowledge.

The traditional approach is characterized by the term *rapid prototyping* [15]. Here, the knowledge engineer implements a prototype system as quick as possible. Then, the initial system is successively refined as the knowledge engineer gets comments from the expert. At some stage in this cyclic process the system is deemed to be adequate for use according to the relevant assessment criteria.

But there are several problems with this approach. In the past ten years, many expert systems have been constructed in a wide variety of domains, but the number of fully operational systems is very small. This probably indicates that the art of knowledge engineering isn't well established yet. Knowledge engineering, like any engineering activity, is a form of problem solving. From the literature two heuristics for effective problem solving are known. First a problem is easier to solve if partitioned into particular subtasks and second a problem should be analyzed completely before solution methods are applied.

Up to now various knowledge engineering methodologies have been proposed to overcome this *knowledge acquisition bottleneck*. The basic idea behind is to view the process of knowledge-based system development as a *modelling* activity [20]. Given this distinction, knowledge engineering can be seen as going through a number of phases (see fig. 1).

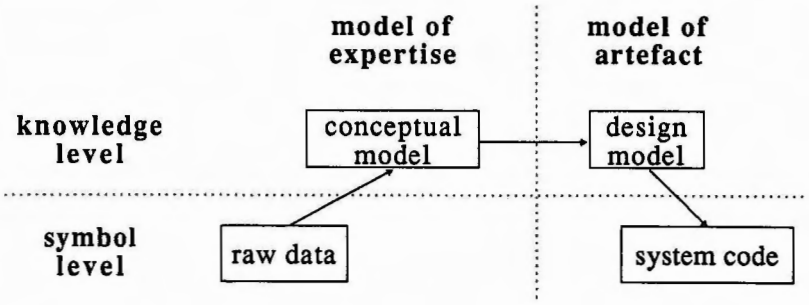


Figure 1: Models in knowledge-based system development

Raw data in the figure refers to the input data from the real world: text books, interviews, thinking aloud protocols, etc. The *conceptual model* is real-world oriented in the sense that it only describes the competence in expert problem solving. The *design model* is a model at the same level of abstraction. It is a high-level logic system design.

The major difference between the conceptual model and the design model thus is that the design model takes into account external requirements.

Therefore knowledge acquisition is divided into three major phases:

1. knowledge elicitation
2. analysis and interpretation,
   where the structure of the problem solving process and the relevant types of knowledge are identified from the elicited data, in order to build an epistemological model of expertise.
3. operationalization of the interpreted and analysed data.

In essence the developing of an expert system involves *modelling* of expertise, i.e., the structuring of the required knowledge, inference competence and *flexible* use of knowledge.

3 The Hypertext System CONCORDE

This section describes the hypertext system CONCORDE, a research prototype suited for active hypertext applications, e.g. decision support, authoring, or knowledge structuring. A first prototype of CONCORDE is implemented in Smalltalk-80 on a SUN-3/60.

Two parts of the system have to be differentiated: a global hypertext and local contexts. The *global hypertext* contains the information that is common to all users. Any information item of the global hypertext is accessible to any user (read access). The information items are stored in a global database and managed by a global object manager.

A *local context* supports individual user views of the global hypertext. Any local context contains a subset of information items copied from the global hypertext, a set of tools for manipulating these items, a local object manager to handle the information, and a local storage. Manipulation of the information items can only take place in local contexts not in the global hypertext. The distinction between global hypertext and local context is also supported by the *client-server-configuration* of CONCORDE. While the global hypertext is stored at the server, any user environment is placed at a client (see fig. 2).

Basic object of the hypertext CONCORDE's is the *card*[2]. A card is made of a unique surrogate, a content of datatype text, graphics, or bitmap[3], a list of system attributes, a list of application attributes, and a set of links.

System attributes are properties independent of an application. Examples for such attributes are the access rights of a card (reading access is always guaranteed), the

[2]We are dedicated to the current fashion to call the basic information nodes cards. A card is a good symbol for a unit whose content is presented as a whole. In addition, the use of cards in hypertext systems is analogous to the usage of notecards in office environments.

[3]A scanner enables CONCORDE to include drawings, notes etc.

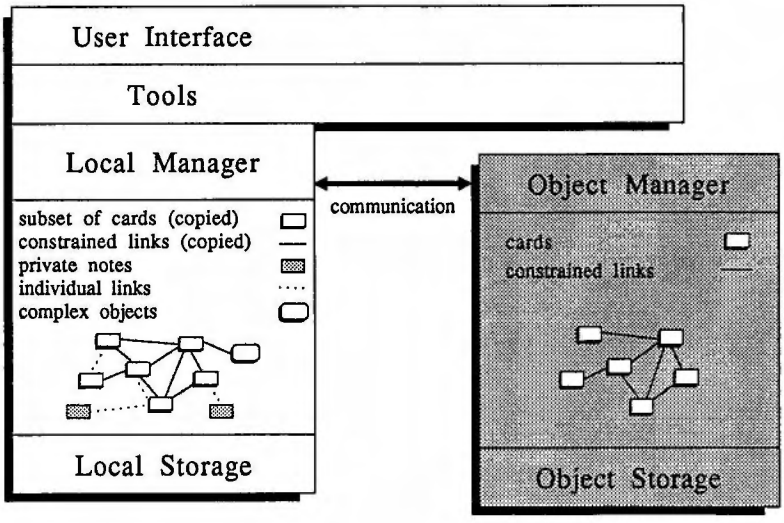


Figure 2: Software-architecture of CONCORDE

mode (cards can be archived or editable), the class of the card etc. Other attributes are application-dependent. Usually, their presentation differs from the presentation of the system attributes. Application attributes are presented by default while system attributes are only shown by demand.

A *link*, instantiating a relation between two cards, is the second basic object of CONCORDE. We differentiate two categories of links: individual links and constrained links.

Constrained links are typed links defined in the global hypertext (see fig. 2). For any link type, two sets of card classes are defined. One set contains the classes whose instances are valid link sources, the other set contains the classes whose instances are valid to be link destinations. In addition, constraints of the link types are defined as a set of predicates concerning operations such as adding a link, deleting a link, and renaming a link.

Therefore, constrained link types are well suited for an adaption to a special application. For instance, the *gIBIS*-system offers a limited set of link types to support a decision support application [7]. The given types of links serve as a restricting structure for the decision process. The placement of links can be controlled. We take advantage of such a link typing. The types and their constraints are tailored to support the knowledge engineering process.

Fig. 3 shows an example of an application of a constraint.

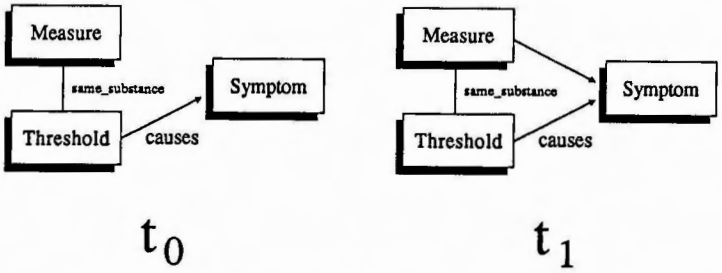


Figure 3: Example of Relationships between Cards defined by Constraints

The constraint shown in fig. 3 could be expressed by the following formula:

$< causes > < insert_link > < \exists\ N_m \in [Measure],\ \exists\ N_t \in [Threshold],$
$\exists\ N_s \in [Symptom], \exists\ L_s \in [same_substance] :\ L_s(N_m, N_t) >$
$< insert_link(causes(N_m, N_s)) \longrightarrow \exists\ L_c \in [causes] : L_c(N_t, N_s) >$

The constraint describes a relationship in a knowledge base containing symptoms, measures causing symptoms, thresholds of these measures, etc. The relationship expresses that a measure causes a symptom since a threshold is exceeded. So, a relationship between the measure and the symptom exists as well as the relation between a threshold and the symptom (here, we neglect the relationship between a measure and its threshold). If the entities of the knowledge base are expressed by card instances of CONCORDE, the insertion of a *causes*-link from a *measure*-card N_m to a *symptom*-card N_s only is allowed if a *causes*-link L_c exists from a *threshold*-card N_t to the same *symptom*-instance N_s. If a user wants to insert such a link, the link only becomes valid if the second link already exists. Then the predicate will evaluate true.

Individual links are untyped. They can be named but cannot be controlled. Links of this category are unconstrained. In opposite to constrained links of the global hypertext, they exist only in local contexts. They are kept in a local storage (see fig. 2), even if the cards they relate to are written back to the global hypertext ("check_in"). We see the following advantages in this approach.

Firstly, we are able to protect private links against the access of other users. We can guarantee the privacy of information. CONCORDE offers the special card class *PrivateNote* in every local context; instances of this class are never inserted into the global hypertext but kept locally. So, not only private link structures can be created, also private remarks are possible. Secondly, a lot of links not relevant to any other user are kept locally. This makes our global hypertext "more readable". Especially, the danger of the effects of disorientation will be reduced. These advantages overcome the disadvantage of the management overhead.

Beside individual links and private cards, local contexts show a third additional structure of information. Cards can be aggregated to *complex objects*; their structure is manipulated with the help of structure cards. All links of the separate cards are still visible in complex objects made of these cards. So, the characteristics of CONCORDE

are:

- A global hypertext common to all users of the system. It contains cards and constrained links, such enabling an adaption of the hypertext to an application.
- Local contexts for building user-environments. They contain a set of cards and constrained links, both copied from the global hypertext, and add privacy (by individual links), personal information (by private cards) as well as structuring support (complex objects).
- Object-oriented design and implementation [17].
- Dedicated tools, e.g. a graphical browser [18].

4 Knowledge Engineering Support by CONCORDE

At various places, attempts are being made to integrate the idea of hypertext with expert systems. Hypertext is used to relate the network of objects, rules, and properties (see e.g. [22]). The result is just a graphical presentation of relationships. Existing shells like Nexpert Object already have a kind of presentation facility like this without calling it a hypertext. Other applications (beyond them knowledge-based systems) use a hypertext network as help system.

In this section, we describe an approach using hypertext for more than simple presentation and help tasks. We are convinced that this tool will bridge the gap between existing expert systems shells and current knowledge acquisition research prototypes. It will enable the knowledge engineer to develop at last *maintainable* expert systems.

Taking advantage of CONCORDE's concepts the three main phases of the knowledge acquisition process are supported by the following functionalities:

1. Any source used in knowledge engineering and accessible in machine-readable form can be made to a hypertext node. E.g., references of the expert to a source in form of an image become a link to this source. So, for the first time the knowledge engineer is able to manage all existing different knowledge sources in an *integrated environment.*
2. Some knowledge elicitation methods are easily integrated. The structure formation techniques like *Networking*, *Heidelberger Structure Formation Technique* or *Card-Sort* methods are based on the fact of being able to present knowledge-structures graphically or visually. These techniques make available an exactly specified visual system for representing the structure of a particular knowledge-domain [1].

   The networking technique was developed to present text-passages graphically, but it is also used to elicit ideas about special personal constructs.

   The Heidelberger Structure Formation Technique was introduced as a technique for elicitating so called *subjective theories.* Background is the assumption that

one can only understand the actions of a human, if one take his knowledge and his assumptions about the physical and social world as reference system.

The Card-Sort method is used to discover an initial structuring of the main domain concepts and to analyze the principles about the structuring process.

3. In a more common fashion, the knowledge engineer is able to create his set of relation types (e.g. links) specially adapted to the domain. This naturally involves some background understanding of the particular expertise.

   In case none of the presented methods seems adequate to the exepert, he is capable of using the underlying hypertext system. He selects information or sources of his area of expertise, defines fitting link types, and relates the information written on cards by links of these types. A knowledge engineer can get an idea of the relationships of the application by looking at the created hypertext.

   Also the expert is able to select subsets of the global hypertext and to *criticize* the relations placed by the knowledge engineer by creating personal remarks and individual links. Individual links are also used to correct errors of the knowledge engineer.

   A related kind of method is the following: Using constrained links, the knowledge engineer presents "exercises". These "exercises" are made of relations the knowledge engineer inferred from the elicited knowledge. If the relation is valid, the expert creates a complex object containing the cards and links of the "exercise". A *valid*-link is placed with the complex object as its destination. If something is wrong, a better solution can be proposed by use of (new) cards and (new) links.

   These procedures result in a temporal independence of expert and knowledge engineer but take for granted a common access to *one* local context.

   The upper part of fig. 4 shows such an exercise prepared by the knowledge engineer: a symptom and four hypotheses set up by it have been inferred by the knowledge engineer. The lower part of the figure shows the structure completed by the expert: a fifth hypothesis has been added and the content of one hypothesis has been changed.

4. Up to now one of the most important methods for knowledge elicitation is *interviewing*. While modelling the "universe of discourse" of an application the knowledge engineer is able to mark ambiguous points with *question-cards*.

   The knowledge engineer is immediately able to write down his intended question in the right context. The question-card is then related to the card containing the questionable content by a *query* link. Later on a *query-modul* accumulates these cards: starting at a given card, travelling along specified links and collecting the question-cards until an appropriated condition is realized.

   This ordered specification will support the knowledge engineer to *pre*structure the interview. Throughout the specified links the question will be arranged in a natural, logical consecutive way.

5. Expert systems require extensive maintenance. Maintenance problems are not so much attributable to the fact that in many domains of expertise knowledge may

change, but rather that systems are so opaque and unstructured that it is hard to tell where updates and modifications should be applied [3]. Simply adding a new rule may affect the way other rules interact.

The knowledge engineer builds a hypertext-model of the entire domain. So the hypertext contains formal expert knowledge as well as knowledge components only important for the knowledge engineer himself. The hypertext includes also the knowledge the rules consist of. So, if a rule is modified the knowledge engineer is aware of the piece of knowledge this appropriated rule models.

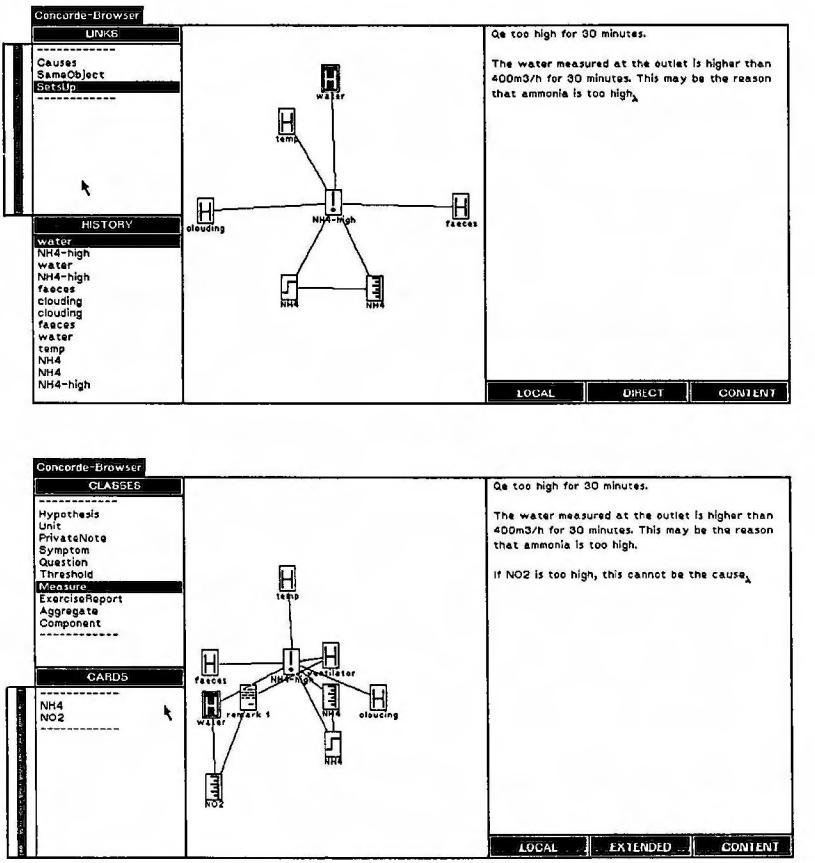


Figure 4: Completion of Structure by *Exercises*

5 Concluding remarks

In this paper, we introduced methods of knowledge engineering (section 2). Also, we described a prototypical hypertext system that is able to manage relationships between individual nodes containing information (section 3). We showed some possibilities to take advantage of the properties of our hypertext systems to support knowledge structuring (section 4).

It hasn't been our goal to introduce a new knowledge engineering methodology nor to present a knowledge-acquisition workbench for expert systems either. But we are convinced that our *integrated approach* of collecting knowledge, managing sources of this knowledge, supporting its structuring, and the facility to maintain the knowledge by hypertext links is an important step to the integration of databases, expert systems, and hypermedia systems.

In the next future, we want to extend our prototype. At the moment, the manager of the global hypertext checks only the static constraints of the typed links. We plan to "activate" this module of CONCORDE. It must be able to interpret the structures of the hypertext. For instance, it should look for forbidden cycles of an argumentation. But we will not make an inferencing module of it. Another task for this module would be the automatic leading of a user to open questions marked by the concerning link.

References

[1] Bonato, M., Knowledge elicitation with structure formation techniques, in: [10]

[2] Boose, J., Gaines, B. and Linster, M., Proceedings of the European Knowledge Acquisition Workshop (EKAW'88), GMD-Studien Nr. 143, 1988

[3] Breuker, J. and Wielinga, B., Use of Models in the Interpretation of Verbal Data, in: [19]

[4] Cercone, N. and McCalla, G., The Knowledge Frontier: Essays in the Representation of Knowledge, Springer-Verlag, 1987

[5] Collier, G.H., Thoth–II: Hypertext with Explicit Semantics, Proceedings Hypertext'87, Chapel Hill (NC), 1987, pp.269–289

[6] Conklin, J., Hypertext: An Introduction and Survey, IEEE Computer, Vol.20, No.9, 1987, pp.17–41

[7] Conklin, J. and Begeman, M., gIBIS: A Hypertext Tool for Exploratory Policy Discussion, ACM ToOIS, Vol6, No.4, 1988, pp.303–331

[8] Cordes, R., Hofmann, M. and Langendörfer, H., Layered Object-oriented Techniques Supporting Hypermedia and Multimedia Applications, Proceedings WOODMAN'89, Rennes, 1989, pp.286–296

[9] Diaper, D., Knowledge Elicitation: Principles, Techniques and Applications, Ellis Horwood Limited, 1989

[10] Diederich, J. and Uthmann, T., Knowledge Acquisition for Expert Systems, Arbeitspapiere der GMD 281, 1987

[11] Gaines, B. and Boose, J., Knowledge acquisition for knowledge-based systems, Academic Press, 1988

[12] Göranzon, B. and Josefson, I., Knowledge, Skill and Artificial Intelligence, Springer-Verlag, 1988

[13] Hahn, U., Jarke, M., Kreplin, K., Farusi, M. and Pimpinelli, F., CoAuthor: A Hypermedia Group Authoring Environment, Tech. Report MIP-8926, Univ. Passau, 1989

[14] Halasz, F.G., Moran, T.P. and Randall, R.N., NoteCards in a Nutshell, Proceedings ACM Conference CHI/GI'87, Toronto, 1987, pp.45–52

[15] Hart, A., Knowledge Acquisition for Expert Systems, in: [12]

[16] Hofmann, M., Cordes, R. and Langendörfer, H., Hypertext/Hypermedia, Informatik–Spektrum, Vol.12, No.4, 1989, pp.218–220

[17] Hofmann, M. and Langendörfer, H., Konzept eines Informationssystems zur Schaffung einer individuellen Arbeitsumgebung durch spezielle Hypertext–Verweise, Notizen zu Interaktiven Systemen, Vol.18, 1989, pp.113–132

[18] Hofmann, M. and Langendörfer, H., Browsing as Incremental Access of Information in the Hypertext System CONCORDE, Proceedings Interactive Communication, Paris, 1990

[19] Kidd, A.L., Knowledge acquisition for expert systems, Plenum Press, 1987

[20] Schreiber, G., Breuker, J., Bredeweg, B. and Wielinga, B., Modelling in KBS Development, in: [2]

[21] Streitz, N.A., Hannemann, J. and Thüring, M., From Ideas and Arguments to Hyperdocuments: Travelling through Activity Spaces, Proceedings Hypertext'89, Pittsburgh, 1989, pp.343–364

[22] SynergTech, ADEPT User Manual, 1989

Knowledge Acquisition from Machine Adjustment Cases Using Causal Model and Operationality Criteria

Kotaro NAKAMURA
JAPAN TOBACCO INC. *

Shigenobu KOBAYASHI
TOKYO INSTI. OF TECHNOLOGY †

Abstract

This paper presents a method of knowledge acquisition from machine adjustment cases based on the explanation-based learning(EBL). This method uses adjustment cases of domain experts as training examples, the causal model as domain theory and observability and strategic importance as operationality criteria respectively. Based on this method, a knowledge acquisition system has been developed.

It consists of generalization, reduction and transformation of explanation structure. By applying it to 50 actual adjustment cases in cigarette making machine, 79 diagnostic rules have been acquired. Comparing these rules with the knowledge base in the practical expert system, the effectiveness of the proposed method has been proven in both knowledge acquisition and knowledge refinement.

1 Introduction

One of the solution to the burden of knowledge acquisition of expert systems is reducing the dependency on knowledge acquisition from domain experts. Various possibilities, such as the model-based reasoning [1] (MBR), the case-based reasoning [2](CBR), and the machine learning [3] have been researched, until now.

MBR uses domain models of structural and functional knowledge about the target object. MBR is capable of processing unknown situations, which knowledge-based reasoning(KBR) does not cover, and presenting justification for its solution. MBR's drawback is the necessity of a lot of measurement or testing for finding solutions because of combinational ambiguity, while in KBR such a problem is not critical because of dependency on compiled heuristic knowledge. [10]

CBR uses past cases of problem solving, which reflect the domain expert's experience with success/failure results. CBR utilizes explicitly raw information implied in expert's problem solving efforts, which have not been transformed yet to generalized knowledge. Also, CBR uses domain knowledge for case modification and repair.[11]

*Engineering Research Laboratory, 1-31 Kurobeoka Hiratsuka Kanagawa 254 JAPAN
†Graduate School of Science and Engineering, Yokohama 227

2.1 Causal model in machine adjustment

A mechanical system consists of many mechanical components (system components) and mechanical connections among them (system structure). Each component is featured by some mechanical properties. If a change of one component's properties affects any other component's properties, there exists a causal relation between the two. A set of such causal relations is here called a causal model.

By varying an abstraction level of modelling of the target machine, various causal models can be derived. Our study identified a causal model at one abstraction level, which domain experts seem to use in real adjustment work, by referring to documents about system structures and component functions of the target machine and gathering some information about causes, faults and symptoms from domain experts. The mechanism of cigarette making machine is shown in Fig.1. Figure 2 shows a part of causal model of it.

The causal model is represented by a directed graph that has nodes representing every mechanical properties of the machine and arcs representing connections among nodes. Figure 2 shows many paths connecting possible causal hypotheses, as cigarette-rod-stopping (F11 illustrated in Fig.2) is recognized as a fault. Hence, in order to identify more possible paths, it is necessary to use more detailed information about causes and symptoms included in machine adjustment cases.

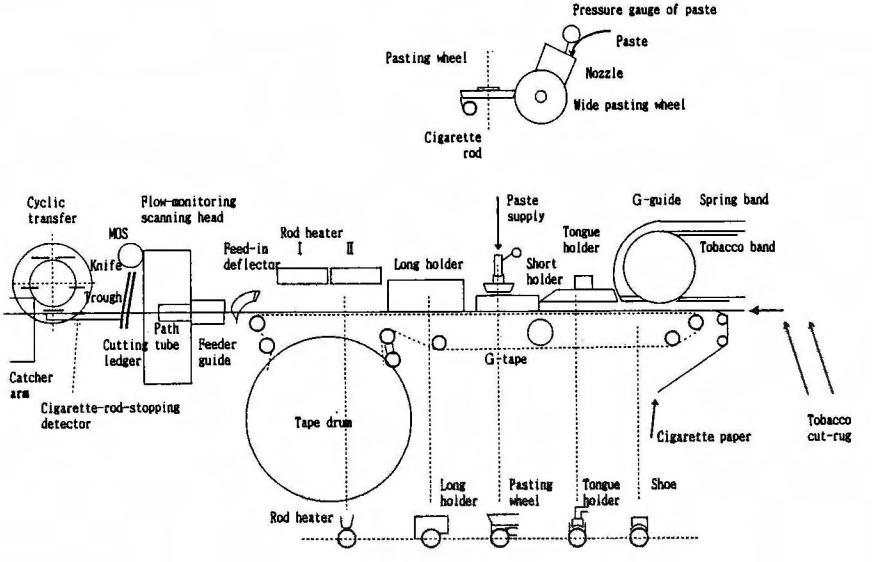


Figure 1: Mechanism about cigarette making machine(partly illustrated)

EBL(explanation-based learning)[4] ,one of deductive learning methods, has been recently investigated by many researchers. EBL is a new framework for generating a macro rule from a training example, and has favorable properties for knowledge refinement in KBR systems. EBL techniques provide justification by explanation of a given example as an instance of a goal concept, using a domain theory, and generate refined macro rules by generalization of explanation structures under operationality criteria[5]. EBL has been applied to domains of fault diagnosis[6][7] ,LSI design[8] and so on.

One of the authors has been involved in the development of expert systems to support adjustment of cigarette making machines. From such an experience, we have reached to have the following view about the expert model in the machine adjustment : "In the situation where perfect heuristic knowledge is available, the experts solve problems by using it. In the situation where heuristic knowledge is imperfect or lacking, they solve problems by using both deep knowledge and past cases. That is, they retrieve a similar past case, then modify it by using causal models, representing the physical flows and relations among the system element's properties concerning the part of the target machine. Then if needed, these cases are generalized, stored as informal know-how and reused to similar problems. Repetitive problem solving seems to promote the transformation of this informal know-how to formal knowledge."

This view suggests the need of methods for acquiring more effective and justified diagnostic knowledge or adjustment knowledge from a lot of machine adjustment cases. Our research adopts an framework of EBL to transform machine adjustment cases acquired from domain experts to generalized knowledge. This paper presents a method of acquiring effective diagnostic knowledge from machine adjustment cases, based on causal models and operationality criteria, and shows its usefulness. Barletta,et al.[6]have tried to apply EBL to the indexing problem of cases in CBR system for domains of fault recovery. Koseki[7] has shown MBR diagnostic system architecture for communication system, one which learns from experience using EBL. While, our research is aimed to use EBL to acquire fault diagnostic knowledge for KBR system.

This paper is organized as follows. Section 2 analyzes features of the use of the causal model and cases in actual machine adjustment. Sections 3 and 4 present a method and procedures of generalization of machine adjustment cases by the EBL technique. Sections 5 and 6 describe the realization of prototype systems based on this method, and show several examples and an evaluation of the presented method.

2 Use of the Causal Model and Cases in the Machine Adjustment Field

In machine adjustment, aiming to rectify faults in machine operation and product quality, related causes are identified to determine an adjustment method. This section discusses the usefulness of the causal model and cases in machine adjustment for problem solving.

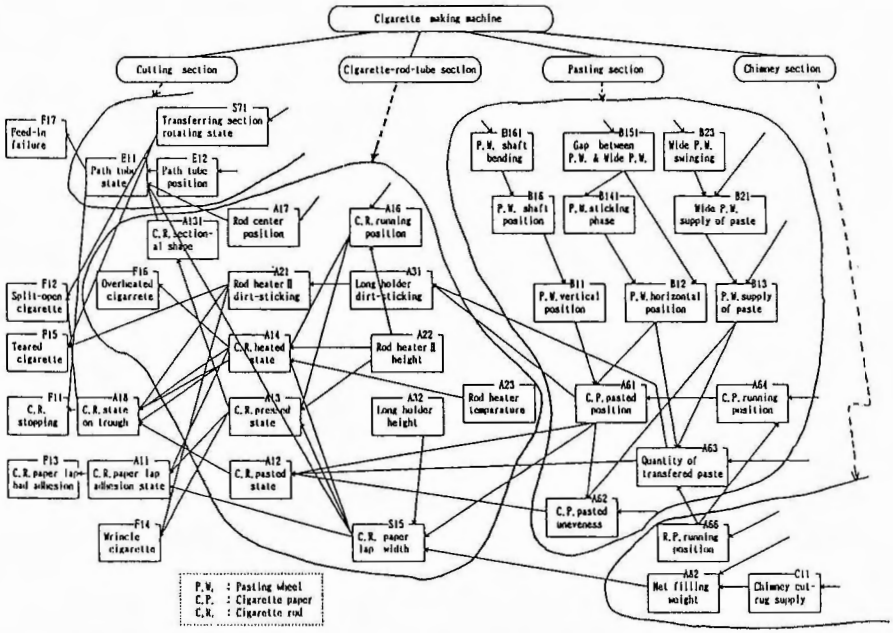


Figure 2: Causal model about cigarette making machine(partly illustrated)

2.2 Machine adjustment cases

A case in the machine adjustment is defined to be a series of information that one fault in the target machine is recognized, some possible causal hypotheses are identified by diagnostic work, an appropriate adjustment method is applied, and its result is evaluated. Now, a machine adjustment case is defined as a form of the following 6 terms.

$$Case = \{F, C, S, M, R, E\} \tag{1}$$

$$; F = Fault, C = Cause, S = Symptom, M = Method, R = Result, E = Evaluation$$

Here, a symptom, which is observable property, is a node on causal paths connecting faults with causes. For example,"pasted position is swing" is a positive symptom and "rod heater has no problem" is a negative one.

As documentation about adjustment cases, production management reports and adjustment standards used informally in the real expert system, are available. In areas where no documentation exist, it is necessary to extract cases directly from domain experts. However, this work is not as difficult as knowledge acquisition, because domain experts can remember cases in form of expression (1) as a kind of episode easily. An example of adjustment case extracted from one expert is shown in Figure 3.

There are many variations in adjustment cases, from performing a general adjustment under typical situations to performing a special one under special ones. Moreover, under the same situations, methods of recognition and adjustment vary with domain

Case 2-3

Fault : cigarette-rod-stopping and bad adhesion of cigarette-rod-paper-lap (F11, F13)

Cause : shaft bending of pasting wheel (B161)

Symptom : cigarette paper position and rod heater are no problem (not(B64), not(A21), not(A22), not(A23))

M.A.method : exchange of shaft in paste wheel

M.A.result : cigarette-rod-stopping and bad adhension of cigarette-rod-paper-lap were resolved

Evaluation : success

M.A. : Machine adjustment

Figure 3: Example of machine adjustment case

experts. Therefore, to acquire knowledge from cases, it is necessary to prove justifications of adjustment cases by using causal models as described in Section 2.1.

2.3 Case explanation based on the causal model

From the above discussion, a complementary relationship between causal model and case exists in contribution to problem solving. Cases are backed by expert's experiences, but not justified. So, we attempt to explain cases by causal model and to generalize them according to operationality criteria under the framework of EBL. This approach makes it possible to resolve combinational ambiguity in model-based reasoning, by using effectively information about not only faults and causes, but also symptoms in real cases.

From the viewpoint of knowledge acquisition, the generalization of cases means that macro rules are generated from causal paths triggered by information included in cases. We think that it is proper to leave the user to determine which part of triggered causal paths should be transformed to macro rules.

2.4 Use of measuring/testing knowledge and diagnostic strategy

By using measuring/testing and diagnostic strategy in addition to the causal model, it is possible to generate knowledge from cases more operational so that problem solving performance would be enhanced.

The measuring/testing knowledge provides the degree of observability concerning each property item in the causal model. The diagnostic strategy provides the degree of strategic importance to identify causes in an efficient way. This knowledge suggests which property items largely contribute to discriminating possible causes. To evaluate explanation paths in the causal model based on both knowledges makes it possible to generalize cases taking into account the operationality such as diagnostic efficiency.

3 Generalization of Cases by EBL

This section proposes a generalization method of cases. In the following, a case is assumed to be a form of triple of cause, faults and symptoms.

3.1 Generalization method of cases by EBL

Figure 4 shows an application of EBL to machine adjustment in the following;

a)goal concept ⇐= machine adjustment goal,
b)example ⇐= machine adjustment case,
c)domain theory ⇐= causal model,
d)operationality criteria ⇐= observability and strategic importance,
e)generalization procedure ⇐= structural generalization,
f)macro rules ⇐= diagnostic rules.

This method consists of generation of explanation structure, reduction of explanation structure, and transformation to rules. First, explanation structure is generated by explaining adjustment case using the causal model under machine adjustment goal. Second, this explanation structure is generalized by structural reduction using operationality criteria. Finally it is transformed into operational rules.

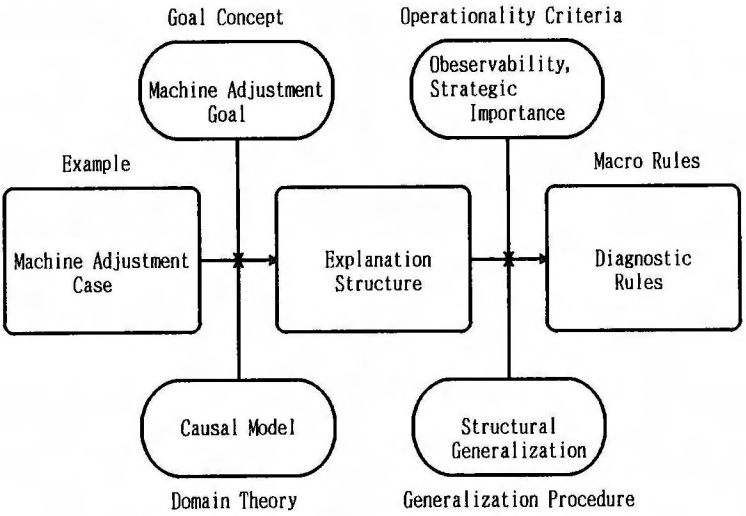


Figure 4: Application of EBL to machine adjustment

3.2 Operationality criteria

We consider how operationality criteria are defined by using measuring/testing and diagnostic strategy knowledge.

The measuring/testing knowledge shows what observation methods each property of the causal model is related to. This knowledge can be acquired from machine

adjustment experts. Observability is classified in 10 ranks, with combination of 2 ranks of easiness of observation and 5 ranks of condition of observation as follows:

Observability:

- easiness of observation:
  - easy (observable by watching or touching),
  - difficult (observable by measuring or exact measuring)
- condition of observation:
  - no (observable at real operation),
  - weak (observable at no-load operation),
  - medium (observable at shutdown),
  - strong (observable at shutdown and after removing mechanical parts),
  - very strong (observable at shutdown and after resolving mechanical parts)

The diagnostic strategy shows which properties are important to efficiently identify possible causes in case of trouble-shooting. Machine adjustment experts seem to have a mental model of mechanical systems, which are separated into several number of subsystems from structural and functional viewpoints. They determine an observational order among mechanical properties taking into account diagnostic efficiency and focussing important subsystems. This knowledge is defined as the strategic importance of each property in the causal model. The strategic importance is a relative measure of the contribution of observing each property in identifying a true cause. The strategic importance is identified by weighing the properties of the causal model in cooperation with experts. This is determined by a combination of the following items with $3 \times 2 = 6$ ranks.

Strategic importance:

- strength of diagnostic effect : large,medium,small
- coverage of diagnostic effect : broad ,narrow

4 Procedures of Knowledge Acquisition from Cases

This section describes detail procedures of knowledge acquisition from cases, which consist of generation of explanation structure, reduction of explanation structure and transformation to rules.

4.1 Generation of explanation structure

Explanation structure is generated by proving that a given case is an instance of adjustment goal by using the causal model. This is equivalent to identify some causal paths connecting nodes on the causal model so that all the information of that case are included. There exist multiple causal paths in one explanation structure, in general. To generate explanation structure, the following 3 steps are performed.

(a) Generation of possible causal paths triggered by cause in a case
By triggering the causal model by cause in the current case, all possible faults to be observed are identified. This search results in generation of causal paths connecting to cause and candidate faults.

(b) Elimination of causal paths by faults
Causal paths which do not include the same faults in the case are eliminated.

(c) Elimination of causal paths by symptoms
Causal paths which include nodes corresponding to negative symptoms in the case are eliminated.

From the above processing, the explanation structure has only a few causal paths.

4.2 Reduction of explanation structure

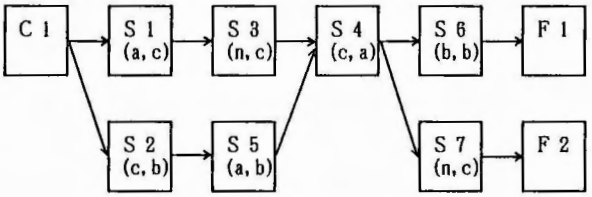
The explanation structure must be generalized. We consider only structural generalization. It means reduction of the explanation structure. The reduction is performed considering operationality criteria. This processing is necessary to transform causal knowledge to compiled operational knowledge. The structural generalization consists of the following 3 steps.

(a) Assignment of evaluation values reflecting operationality criteria
For each node in the explanation structure, evaluation values, which reflect Operationality criteria are assigned.

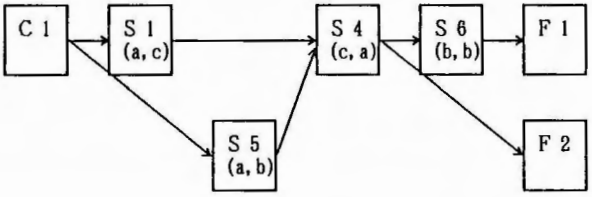
(b) Reduction based on evaluation value(No.1)
By using evaluation values of observability, unobservable nodes in the explanation structure are removed.

(c) Reduction based on evaluation value(No.2)
Arcs having no branch are found for unifying nodes in the link. The unified node is selected by considering the relations of trade-off between observability and strategic importance of each arc in the link. Figure 5 shows an example of the reduction of explanation structure.

The purpose of reduction of explanation structure is to generate operational knowledge. This processing is essentially possible to bring about overgeneralization of knowledge. However,if such knowledge is generated, the problem of misleading solution to wrong conclusion can be escaped by flexibly utilizing it in the knowlege base.



a)Assignment of evaluation values of operatinality criteria to explanation structure



b)Reduced explanation structure after structural generalization

: Node
; C:Cause, F:Fault, S:Symptom

Links : Causal relations

(x , y) : Evaluation value of operationality criteria
;x :Evaluation value of observability
;y :Evaluation value of strategic importance

Evaluation value is higher according to alphabetical order (n : not observable)

Figure 5: Illustration of reduction of explanation structure

4.3 Transformation to rules

A causal path on the reduced explanation structure is called primitive rule. We consider a composition of closely related primitive rules. First, we must extract primitive rules. Second, redundancy among rules must be excluded. Finally, an appropriate granularity of rules should be generated. The transformation to rules is carried out by the following procedure.

(a) Extraction of primitive rules
Possible primitive rules are extracted by tracing all causal paths on the reduced explanation structure. A primitive rule is composed of fault element(F), multiple symptom elements(Si ;i=1,2..), and cause element(C). It is represented in the following forms.

$$F\&S1\&S2..... \rightarrow C \qquad (2)$$

(b) Exclusion of redundant rules
If an inclusive relation exists among conditional parts of primitive rules, more special rules are excluded.

(c) Disjunctive connection of rules
Taking into account of the maintenance and extension of knowledge base, multiple primitive rules should be aggregated into a disjunctive form to make an appropriate granularity of rule.

(d) Display in sentence expression
The aggregated rules are translated into a sentence expression in order to help easy understanding by the expert.

5 Realization of knowledge acquisition system

The procedure of knowledge acquisition from cases described in Section 4 has been implemented. This section describes the system configuration and execution example.

5.1 System configuration

Figure 6 illustrates the system configuration, which consists of seven functions of designation of case, explanation of case, reduction of case, transformation to rules, adjustment case base, causal model, and data of observability and strategic importance. This system was built using the ESP language. Adjustment case base is a place where stores adjustment cases. A case is represented by a frame structure.

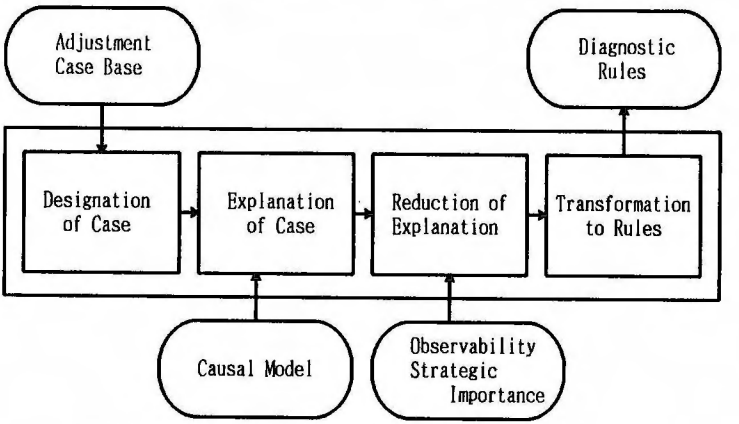


Figure 6: System structure of knowledge acquisition system

Actual machine adjustment cases has been acquired from domain experts in 4 cigarette manufacturing factories. Currently, 50 cases are stored in the case base. Causal model is represented by a predicate form. It is developmented for cigarette making machine by topological analysis of its mechanism, and includes 120 nodes consisting of 38 causal nodes, 10 fault nodes, 57 symptom nodes, and 15 unobservable

intermediate nodes. Data of observability and strategic importance are stored as evaluation values for each mechanical property.

5.2 Execution example

The execution example of this system is shown here. In real processing, the ESP multiwindows system was used.

1. Designation of case

   The next expression is followed by form (1) in Section 2.2.

   Adjustment case =

   { "cigarette-rod-stopping, bad adhesion of cigarette- rod-paper-lap (F11,F13)" ,

   "rolling paper position and rod heater are no problem

   (not(S64),not(S21),not(S22),not(S23))" ,

   "shaft bending of paste wheel(C161)" ,

   "exchange of shaft in paste wheel" ,

   "cigarette-rod-stopping and bad adhesion of cigarette-rod-paper-lap were resolved" ,

   "success" } .

2. Generation of explanation structure

   (a) Generation of possible causal paths triggered by cause $\longrightarrow$ 25 paths

   (b) Elimination of causal paths by faults $\longrightarrow$ 12 paths

   (c) Elimination of causal paths by symptoms $\longrightarrow$ 11 paths

   Figure 7 illustrates the generated explanation structure.

3. Reduction of explanation structure

   (a) Assignment of evaluation values reflecting operationality criteria

   Figure 8 shows an explanation structure with evaluation values of the operationality criteria. Both of evaluation values about observability and strategic importance are expressed in a form of a ranking symbol below the symbol of each node. The ranking symbol means higher rank of evaluation according to alphabetical order.

   Symbol 'n' means to be not observable.

   (b) Reduction based on evaluation value

   Figure 9 shows the reduced explanation structure. Based on the operationality criteria, the only nodes enclosed by the thick rectangles as described in Fig.8 are selected.

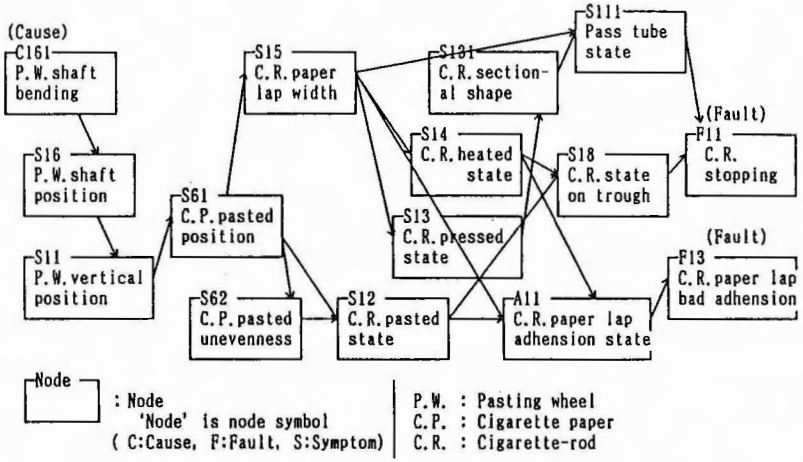


Figure 7: Explanation structure of adjustment case 2-3

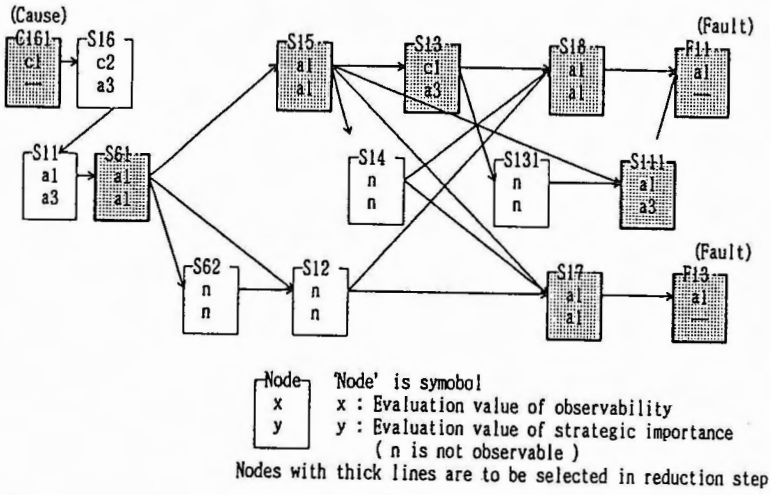


Figure 8: Explanation structure with evaluation values of operationality criteria

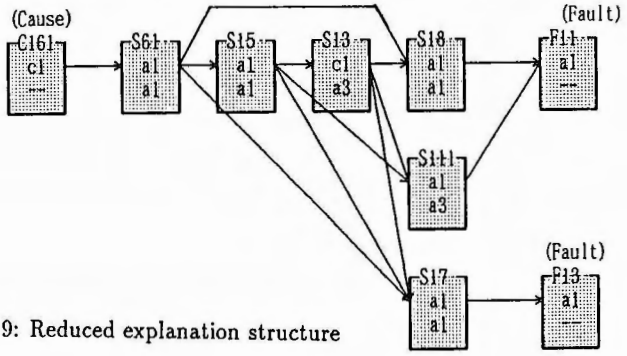


Figure 9: Reduced explanation structure

4. Transformation to rules

   (a) Extraction of primitive rules

   Figure 10 shows primitive diagnostic rules.

   (b) Exclusion of redundant rules

   R1, R3, R5 and R6 in Fig.10 are redundant rules. So, they are excluded. A set of effective rules are shown in Fig.11.

   (c) Disjunctive connection of rules

   This case is not applicable.

   (d) Display in sentence expression

   Here, rule R2 is expressed in a natural language sentence form as follows:

   "If " cigarette-rod-stopping(F11)" as a fault is found, and "cigarette-rod-splitting-open(S18) and bad-paper-rolling-position(S61)" as the symptoms are observed, then "shaft bending of paste wheel(C161)" as one of causal hypotheses is inferred."

By confirming R4 and R7 too in the same way, these 3 rules are acquired as the final diagnostic rules as shown in Fig.11.

```
R1 : F11 & S18  & S13 & S15 & S61 ---> C161
R2 : F11 & S18              & S61 ---> C161
R3:  F11 & S111 & S13 & S15 & S61 ---> C161
R4 : F11 & S111       & S15 & S61 ---> C161
R5 : F13 & S11  & S13 & S15 & S61 ---> C161
R6 : F13 & S11        & S15 & S61 ---> C161
R7 : F13 & S11              & S61 ---> C161
```

Figure 10: Primitive diagnostic rules

```
R2 : F11 & S18 & S61 ---> C161
R4 : F11 & S111 & S15 & S61 ---> C161
R7 : F13 & S11 & S61 ---> C161
```

Figure 11: Final diagnostic rules

5.3 Application to actual cases

This system was applied to actual 50 adjustment cases according to the procedure described in Section 5.2. It is conformed that the system performed the knowledge acquisition of 79 rules by generalizing 47 cases. Concerning the remaining 3 cases, the system failed in their generalization. The details are discussed in Section 6.2.

6 Evaluation

This section evaluates the effectiveness of the proposed knowledge acquisition system based on reviews by domain experts, comparing with the corresponding rules in the existing expert system and the incompleteness of the causal model.

6.1 Reviews by domain experts

To obtain the domain experts' opinion on this method, the method with its execution examples were shown to domain experts experienced in building practical expert systems. Their opinions were that : a)This method is similar to their method of making rules by using the fault analysis trees as a foothold made by the causal knowledge about the applied machine. b)This method makes it easy to obtain knowledge at almost the same level of generalization as from plural experts; hence, it is useful to support knowledge acquisition work since experts depend upon their own judgement to transform their experiences to a proper level of generalization.

We would like to point out the following reasons for the opinions described above. First, there is a great similarity between the EBL method and experts' way of acquiring knowledge based on the utilization of fault analysis trees for knowledge completeness. Second, experts recognize that experimentally useful knowledge is still required to indicate justification or operationality of the knowledge, and find that the proposed method can contribute to getting these indications.

6.2 Comparison with the existing knowledge base

Expert systems in the same domain as shown in this paper have been already put into practical use. So as to evaluate results of the proposed knowledge acquisition method, acquired diagnostic rules were compared with the corresponding knowledge base of this expert system. Table 1 describes the comparison with the existing knowledge base. The results of comparison can be classified by the following kinds of contribution to the existing knowledge base : a)Acquisition of new knowledge acquisition (44 rules), b)Refinement of the existing knowledge (19 rules), c)Justification of the existing knowledge (16 rules), and d)Failure of transformation to knowledge. Case "a" means that causal information included in the conclusional part of a rule is not included in the existing knowledge base. Case"b" means that causal information included in the conclusional part is included in the existing knowledge base, but the diagnostic information about some causal paths linking faults to causes is not included. Case "c" means that the rules are similar to those in the existing knowledge base.

The considerations in Section 6.1 and 6.2 confirm the ability to perform efficient generalization of adjustment cases by the presented method using the causal model and operationality criteria to make effective operational rules. Moreover, it shows that this method has possible use for adding new knowledge to the knowledge base.

Table 1: Comparison with the existing knowledge base

Classification	*Num. of cases*	*Num. of rules*
a) Acquisition of new knowledge	31 cases	44 rules
b) Refinement of the existing knowledge	9 cases	19 rules
c) Justification of the existing knowledge	7 cases	16 rules
d) Failure of transformation to knowledge	3 cases	——
Total	50 cases	79 rules

6.3 Incompleteness of the causal model

Case"d" shown in Table 1 failed in generalization. The reasons for the failure to transfer to knowledge are generally : a)the problem of case justification , b)the problem of incomplete knowledge about the causal model and operationality criteria.

By virtue of the fact that cases used in this study actually experienced and their representation is precise, they exclude problem"a". Problem"b" is related to the incomplete problem[4][9] ,so that a positive example is failed to be proven. The lacking of causal paths to explain cases is a reason for the failure of justification of some cases of the causal model used in this study. Further work is required in the future to make the causal model more elaborated.

7 Conclusion

This paper presented a knowledge acquisition method from machine adjustment cases using a causal model concerning mechanical properties and operationality criteria of observability and strategic importance. Under the framework of EBL, the method, which reflects the domain experts' utilization of the causal model, adjustment cases, and measuring/testing knowledge and diagnostic strategy, has been proven to be quite effective.

To realize the knowledge acquisition system, the following functions have been developed : a)generation of explanation structure, b)reduction of explanation structure and c)transformation to rules. Using these functions, a real system automating knowledge acquisition has been implemented and has been applied for 50 actual adjustment cases. It has been confirmed that the proposed method can perform efficient generalization of adjustment cases and can provide information for refinement of knowledge base in the existing expert systems.

References

[1] Davis,R. *Diagnostic Reasoning Based on Structure and Behavior*, Artifi. Intell., Vol.24, pp.347-410(1984).

[2] Proc.of a Workshop on Case-Based Reasoning, DARPA(1989)

[3] Michalski, R.S.,et al(eds.) Machine Learning, Vol.2, Tioga Pub.Co. (1986).

[4] Mitchell,T.M.,Keller,R.M.and Kedar-Cabelli,S.T. *Explanation-Based Generation; An Unifying View*, Machine Learning, Vol.1,pp.47-80(1986).

[5] Keller,R.M. *Defining Operationality for Explanation- based Learning*, Proc. of AAAI-87, pp 482-487(1987) (revised version in Artifi. Intell., Vol.35, pp.227-241(1988).

[6] Barletta,R.,et al. *Explanation-based Indexing of Cases*, AAAI-88, pp.541 - 546(1988).

[7] Koseki,Y. *Experience Learning in Model-Based Diagnostic Systems*, IJCAI-89, pp.1356-1362(1989).

[8] Mitchell,T.M.,et al. *LEAP: A Learning Apprentice for VLSI Design*, IJCAI-85,pp.573-580(1985).

[9] Rajamoney S.,et al. *The Classification, Detection and Handling of Imperfect Theory Problem*, Proc. of IJCAI-87, pp242-248 (1987).

[10] Nakamura K., Kobayashi S. *A Fault Diagnosis System Based on Parallel Interaction*, Proc. of International Workshop on Artificial Intelligence for Industrial Applications IEEE, pp100-105(1988).

[11] Nakamura K., Kobayashi S. *An Interactive Case-Based Reasoning Model and Its Application to Supporting Machine Adjustment*, Journal of Japanese Society for Artificial Intelligence Vol.4, No.6, pp704-713(1989)(in Japanese).

ISAK: Interview System for Acquiring Design Knowledge

-- A new architecture of interview systems using examples --

Riichiro MIZOGUCHI I.S.I.R., Osaka University, 8-1, Mihogaoka, Ibaraki, Osaka, 567 Japan.
Katsushi MATSUDA Faculty of Engineering, Kansai University.
Yasuhiro NOMURA Faculty of Engineering, Kansai University.

Abstract

Knowledge acquisition is known as a bottleneck of expert system building. A lot of works has been done to overcome this difficulty and many systems that help acquire expertise are developed to date. But most of them are based on interview which asks many questions to domain experts. This paper discusses an architecture unifying interview techniques and learning mechanism which contributes to decreasing questions. EBL is employed in our system. ISAK, which is a knowledge acquisition system for design knowledge, is designed based on the new architecture. ISAK is composed of two modules UBL and UBI. UBL is a module that compiles design knowledge by understanding examples of design drawings and UBI is a module activated, when UBL fails to understand an example, to acquire design know-how or missing domain theory which contributes to improvement of the future performance of UBL. A detailed description of ISAK is presented in the paper together with some examples.

1. INTRODUCTION

Knowledge acquisition is one of the important tasks in building knowledge-based systems. Nevertheless, domain experts are very weak at articulating their knowledge, since they are solving problems using the knowledge which has already been compiled into skills. Currently, knowledge engineers are performing the knowledge acquisition task by interviewing with domain experts.

Research on knowledge acquisition has been made extensively in order to overcome the knowledge acquisition barriers[1]-[3]. However, most of them ask domain experts of all they need, so many questions are asked, which bother domain experts. ISAK is a system which is designed to integrate interview and learning mechanisms aiming at reducing questions and ask crucial questions which contributes to making knowledge acquisition more efficient. This paper presents the philosophy of ISAK along with a detailed description of its mechanism.

2. OVERVIEW OF ISAK

2.1 Design philosophy

The authors have been involved in the development of several interview systems[4][5]. One of the problems of interview systems encountered in the experience is that it asks many questions to the domain experts, since it has no knowledge about the domain under consideration. Another issue in building an interview system is how to know what knowledge is needed. In order to cope with these two difficulties, the authors designed another interview system called ISAK which utilizes EBL[6] as a central engine of the interview process. Given an example, ISAK tries to

understand it according to its domain theory given in advance. When it succeeds, the result is stored in the knowledge base. Otherwise, it activates the interview subsystem to acquire knowledge necessary for fixing the failure of its explanation process. The integration of EBL and interview technologies is called UBI(Understanding-Based Interview) mechanism. ISAK exploits UBI mechanism through its knowledge acquisition process.

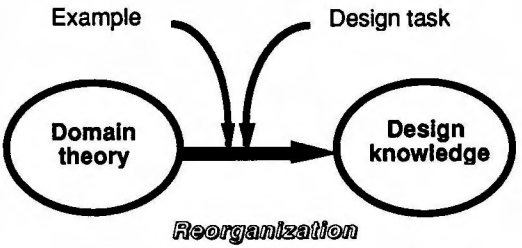


Fig.1 EBL as specialization of domain theory.

2.2 EBL as an operationalization technique of domain theory

EBL is usually viewed as a domain-theory-guided generalization of an example. Although it is true, this view gives us nothing valuable from the knowledge base point of view. Another view of EBL is that it is an operationalization of domain theory(see Fig. 1). This alternative view of EBL is very useful for building knowledge bases.

Domain theory is usually task-independent and is stored as basic facts and relations between objects in the domain. So, they cannot be directly used for solving problems. It is necessary for using it for problem solving to know what, when and how to use. In other words, domain theory is not operational in its nature. It should be made operational when it is used for solving problems.

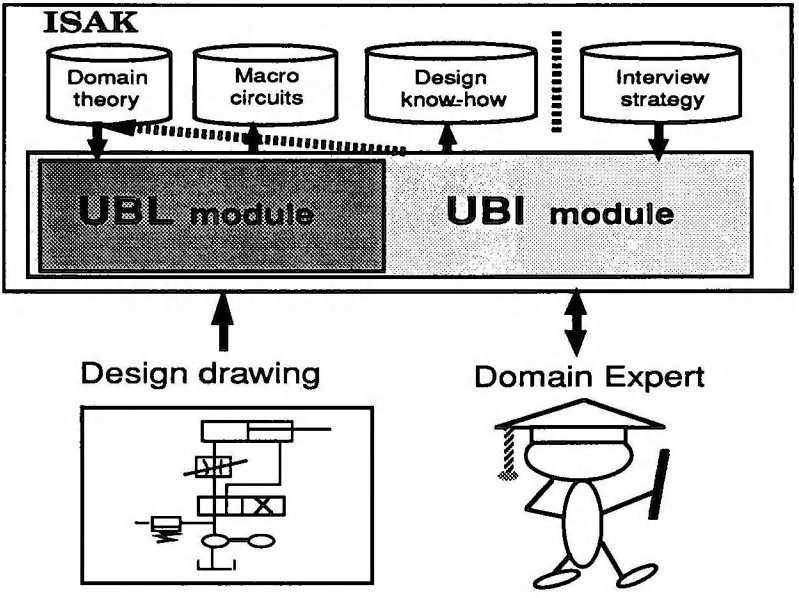
Consider a design problem. Even if one knows a lot of domain theories of the domain under consideration, he/she cannot design anything because he does not have any heuristics concerning how to use them in what order. But he/she can understand examples and explain why some parts are used like that way. After he/she understands an example, he/she can compile the understanding process into a macro knowledge useful for later design.

In ISAK, design of oil hydraulic circuits is employed as its task. The domain theories are some principles of fluid and fundamental functions of elementary components. The goal concepts are requirement specifications of the design drawings which are given as training examples. When ISAK succeeds in understanding the example, it stores the resulting description of the form of "If {specification} then {use the component in this structure}" which is operational in the design task of oil hydraulic circuits. When it fails to understand the example, interview is activated to obtain know-how or domain theories.

3. ISAK

3. 1 System description

FIgure 2 shows a block diagram of ISAK composed of two major modules and four knowledge bases. UBL(Understanding-Based Learning) is a learning module and acts as a driver of the interview module UBI(Understanding-Based Interview). Knowledge bases for domain theories and interview strategies are given in advance. The other two knowledge bases are built during the interview process. Macro circuits obtained through understanding of examples are stored in the knowledge base. The other type of knowledge which is obtained through interview can be considered as know-how of the domain experts, since EBL fails to explain it in terms of the domain theories.



ISAK: an Interview System for Acquiring design Knowledge
UBL: Understanding-Based Learning
UBI: Understanding-Based Interview

Fig. 2 The block diagram of ISAK.

3.2 UBL

UBL is given the following four kinds of information like EBL:

Goal concepts:	Requirement specifications of the circuits.
Training example:	A design drawing of oil hydraulic circuit.
Domain theory:	Basic theories and functions of elementary components.
Operationality:	To describe the knowledge in the form of "If {specification} then {macro circuit}.

From this description, one may think UBL is the same as EBL. Although UBL employs the framework of EBL, the explanation and concept description processes are much more complicated than those in EBL. UBL is composed of 4 modules as shown in Fig. 3. Understanding module is further divided into two submodules such as proof module and explanation module which consists of extraction and explanation generation modules.

3.2.1 Translator module

There are two levels for conceptual description in our task. One is the level for human experts and the other is for the system. Requirement specifications are described in terms of speed, force and other concepts used in user side world. On the other hand, descriptions for the system has to use concepts in physical parameters which the modules in the system can manipulate. The translator is responsible for translation between these two kinds of concepts.

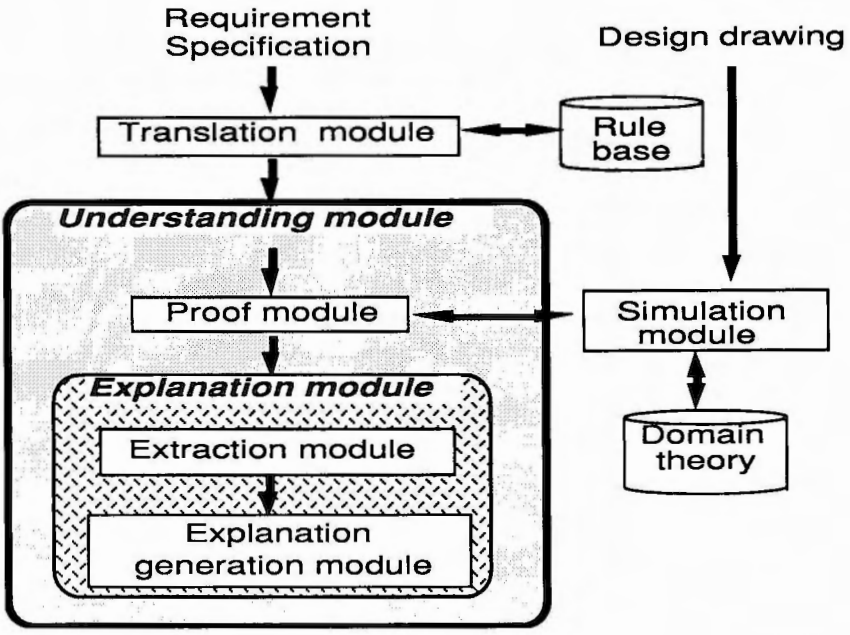


Fig. 3 The block diagram of EBL.

STATE	Pressure is less than a limit
POSITION	In-port of the cylinder
CONDITION	None
STATE	Flow rate is controlable
POSITION	In-port of the cylinder
CONDITION	None
STATE	Flow rate is 0
POSITION	Out-port of the cylinder
CONDITION	Load is negative
STATE	Flow rate is constant
POSITION	Out-port of the cylinder
CONDITION	Load is decreasing

Fig. 4 Some examples of goal concepts.

The major tasks of this module is to generate the goal concepts from the requirement specifications appropriate for understanding by the following modules. A goal concept generated is composed of three parts as shown below.

Goal concept:
STATE
POSITION
CONDITION

STATE shows the parameter and its value which is a goal concept in ordinary sense, POSITION the location in which the value of the parameter should be evaluated and CONDITION the mode of the circuit when the simulation should be made. A few examples are shown in Fig. 4. The translator module is designed as a rule-based system.

3.2.2 Simulator module

In order to understand the oil hydraulic circuit, the system has to know its behavior. Simulation module obtains several descriptions of the behavior of the circuit according to three simulators such as effective path simulator, pressure simulator and flow rate simulator.

There are many possible paths in the circuit some of which are redundant, so the system has to eliminate such paths and find effective ones suitable for simulation concerning each parameter. The last two simulators perform simulation based on the paths identified. Both ones obtain formulas with constraints for calculating the parameters appearing in the goal concept by tracing the components along the paths. The results are described in the format as follows:

$$X = f(P1, P2, ..., Pm) : (C1, C2, ..., Cn),$$

where X is a parameter of goal concept, f is a function of m arguments Pi related to components on the path and Ci are constraints associated with X. These three simulators are also designed as rule-based system. Simulation is performed by consulting with domain theory and circuit topology. All the kinds of knowledge is described in declarative form. Fig. 5 shows some examples of domain theories.

single-rod-cylinder
- **terminals**
  - t1:head
  - t2:rod
  - ex:external
- **parameters**
  - pressure, flow, force, speed
- **constants**
  - a1, a2
- **relations**
  - pressure[t2]=pressure[t1]-force[ex]/a1
  - pressure[t1]=pressure[t2]-force[ex]/a2
  - flow[t1]=(a1/a2)*flow[t2]
  - flow[t2]=(a2/a1)*flow[t1]
  - speed[ex]=a1*flow[t1]
  - speed[ex]=a2*flow[t2]

2-input junction
- **terminals**
  - t1:in
  - t2:in
  - t3:out
- **parameters**
  - flow
- **relations**
  - flow[t3]=flow[t1]+flow[t2]

relief valve
- **terminals**
  - t1:in
  - t2:out
- **parameters**
  - pressure, flow
- **constants**
  - sp
- **relations**
  - pressure[t2]=1
  - pressure[t1]=sp:pressure[t1]>=sp
  - pressure[t1]=pressure[t1]:pressure[t1]<sp
  - flow[t2]=0

Fig. 5 Some examples of domain knowledge.

3.2.3 Proof module

This module proves that the circuit shown in the design drawing satisfies the goal concepts derived from the requirement specifications. As was mentioned above, the simulation results are

described in terms of two kinds of expressions such as function of several parameters and constraint attachment. According to these two forms, the proof module consists of two processes shown below.

(1) Evaluation of functions and

(2) Pattern matching for constraint identification.

Evaluator is composed of several submodules corresponding to specific types of goal concepts. For example, suppose that a goal concept such that parameter P is greater than X and the simulation result is as follows:

$$P = X + Y.$$

Then, the proof is made by showing Y is positive. On the other hand, we have a goal concepts such as P is controllable and the simulation result as follows:

$$P = f(X, Y) : (controllable, P <= Pmax).$$

In such a case, this module tries to find out the constraint "controllable" in the resulting expression according to the pattern matching method.

3.2.4 Explanation module

Explanation module consists of two submodules for extraction of key components and explanation generation. In order to explain how the circuit satisfies the specifications, we have to identify the key components which contribute to realization of the performance. For this purpose, extraction module analyzes the proof process by tracing back to every component which might have the key function corresponding to goal concept.

When the extraction succeeds, an additional information concerning the location of the component is also obtained to make the information useful. Then, an explanation is constructed according to the following templates:

When CONDITION holds,
the Component at the Location realizes the STATE at POSITION,

where CONDITION, STATE and POSITION are defined in the goal concept.

3.2.5 Design knowledge generation

The explanation obtained is not adequate for use in the design phase, since all the action parts are written in terms of physical parameters. The translator mentioned in 3.2.1 is again used here to re-describe the explanation in terms of concept at function specification level. Furthermore, the condition and action parts are reversed so as to be used by the forward-chaining production systems.

3.3 Example

Figure 6 shows a design drawing of a simplified circuit. This is called a "Difference-driven circuit" which has a junction connecting in-port and out-port of cylinder to the pump. It has two modes depending on the state of the "direction control valve". In the mode shown in this figure, the piston moves to the right when the oil comes to flow, since the force at the left side of the piston is greater than that at the right side. The difference of the force is caused by that of areas of the left side and right side of the piston. The velocity of the piston is determined by the flow rate at the piston. The oil flowing out of the piston joins the in-port stream from the pump at the junction, so the flow rate at the in-port is greater than that coming out of the pump, which causes the piston to move to the right very fast. This is one of the major characteristics of the "Difference-driven circuit".

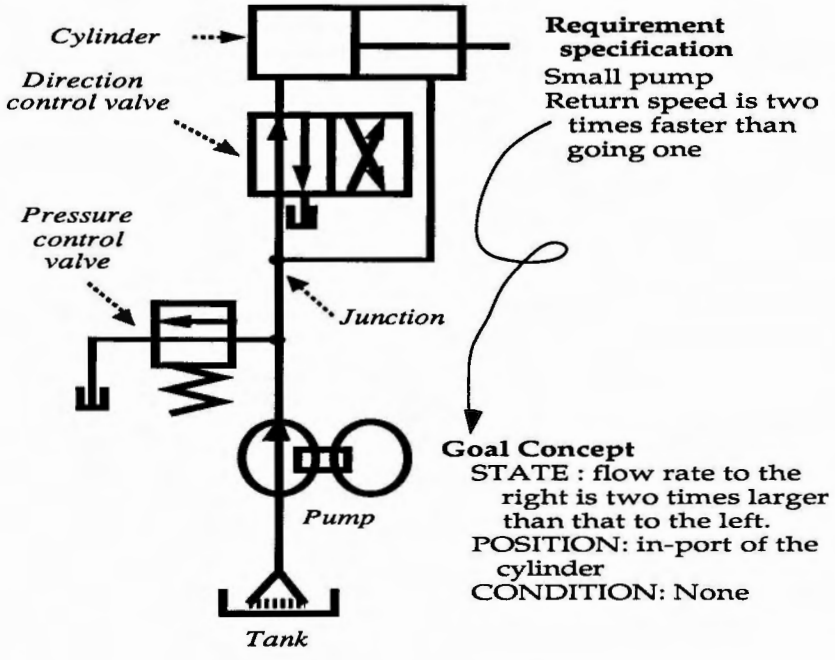


Fig. 6 An example drawing and its specification.

Now, suppose that the design drawing shown in Fig. 6 is given with a specification stating

"The return speed of the piston is two times faster than going one".

Then, UBL is given a goal concept composed of three components such as

STATE	Flow rate to the right/Flow rate to the left = 2
POSITION	Piston
CONDITION	None

Next, it begins the simulation to obtain the relations shown in the Fig. 7. After that, the proof module proves the STATE as follows:

$$\begin{aligned} Q_{in,right}/Q_{in,left} &= (A1*Q_{pump}/(A1-A2)) \,/\, (A1*Q_{pump}/A2) \\ &= A2 \,/\, (A1 - A2) \\ &= 2 \,/\, (3 - 2) \\ &= 2 \end{aligned}$$

When the proof is done, the extraction module tries to detect the key component. The process is shown in Fig. 8. It traces the proof process from the final result of the proof until it encounters

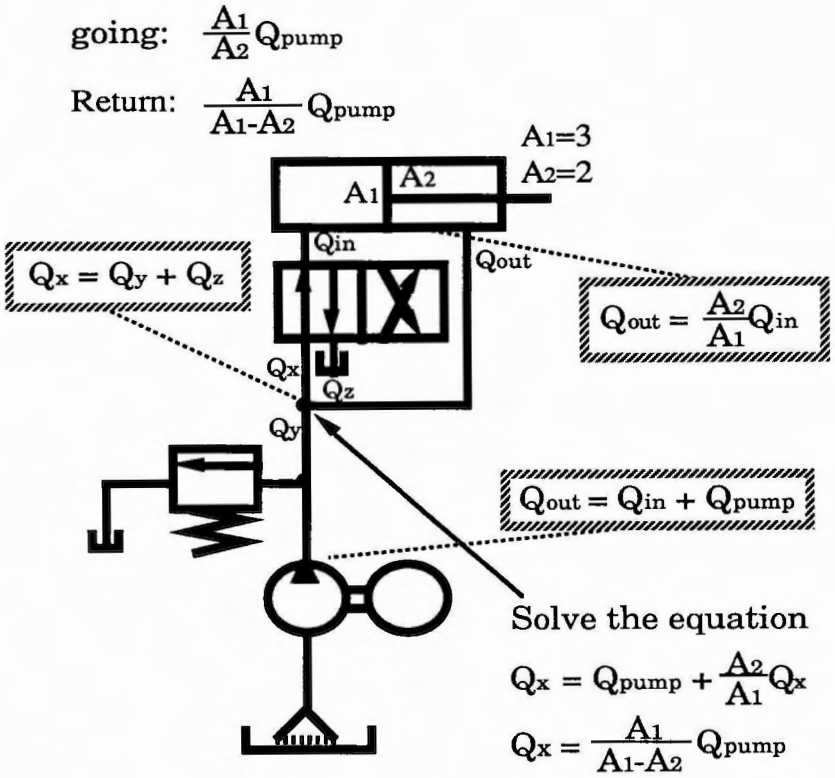


Fig. 7 Simulation results.

circuit components. In this example, it tries to know how the enumerator and denominator were obtained and finds out that both were flow rates at the in-port and out-port of the piston. The former was obtained by solving the equation

$$Qin = Qpump + Qin * A2 / A1$$

which is an instance of the equation of the flow rate at the junction. In this equation, *Qpump* is an output flow rate of the pump and the term *Qin * A1 / A2*, which is the flow rate at out-port of the cylinder, was obtained according to the areas of the piston. Thus, three components such as junction of the pipe, pump and the piston with the areas A1 and A2 are identified as candidates of the key components. However, *Qpump* makes no contribution to the proof, since it appears in enumerator and denominator of the final fraction. So, pump is eliminated from the candidates. Finally, the structure information of the junction is extracted as shown below.

"The junction is located among pump, in-port and out-port of the cylinder."

After extracting necessary information, an explanation is generated. The result is

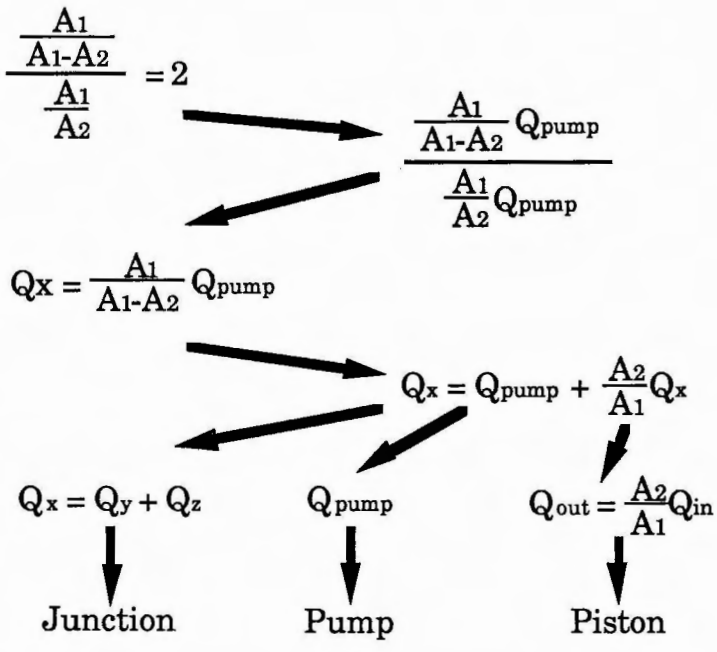


Fig. 8 Extraction process.

If there is a junction connecting pump, in-port and out-port of the cylinder and a piston with areas A1(=3) and A2(=2),
***then** the flow rate at the in-port of the cylinder is two times larger than that of out-port of it.*

In the design knowledge generation module, the above result is converted to an expression in appropriate form.

If the return speed of the piston is two times faster than that of going one,
***then** use a piston with the areas of A1(=3) and A2(=2) and a junction connecting in-port, out-port of the cylinder and pump.*

Thus, a simple design knowledge has been successfully obtained.

3.4 UBI

UBI is composed of two modules other than UBL. One is failure analysis module and the other is interview module.

3.4.1 Failure analysis module

There are three cases where a failure occurs:
(1) The knowledge obtained is inadequate or incorrect.
(2) Knowledge corresponding to a certain goal concept is not obtained.
(3) Knowledge corresponding to a certain component is not obtained.

This module is activated to recognize the current situation according to the following procedures:

(1) When each piece of knowledge is generated, it asks the domain expert wether it is correct or not.

(2) When UBL is finished, it examines if knowledge is obtained for all the goal concepts and components.

After the failure analysis, UBI is activated to obtain knowledge necessary to fix the errors.

3.4.2 Interview module

According to the types of errors, UBI tries to obtain three kinds of knowledge: hidden requirement specifications, additional domain theories and design know-how difficult to explain in terms of the domain theory by invoking one of the following three interview modules corresponding to the failure:

(1) Functional specification acquisition module
   a) Confirmation of relating components
   b) Identification of their functions
   c) Acquisition of several conditions
   d) Translation of the results into functional specification
(2) Domain theories acquisition module
   a) Identification of a module responsible for the failure
   b) Identification of a cause of the failure
   c) Acquisition of domain theory necessary for the incomplete explanation according to the strategies similar to those in TEIRESIAS
(3) Design know-how acquisition module
   a) Acquisition of condition parts of the know-how
   b) Acquisition of action parts of the know-how
   c) Generation of the know-how

Knowledge acquired through interview conducted by UBI is used for future understanding of examples in UBL, thus the scope of its understanding extends as the interview proceeds.

4. CONCLUDING REMARKS

ISAK is currently implemented on MacIvory in Common Lisp and Flavors. ISAK is a step towards unification of learning and interview system for knowledge acquisition. It is crucial in ISAK project to refine interview strategies for identifying the cause of the failure of understanding. Although it is under development, ISAK will help acquire design knowledge for oil hydraulic circuits.

REFERENCES

[1] J. H. Boose: A knowledge acquisition program for expert systems based on personal construct theory, J. Man-Machine Studies, Vol. 20, pp. 21-43, 1985.

[2] L. Eshelman et a.: MOLE: A knowledge acquisition tool that uses its head, Proc. of AAAI '86, pp. 950-955, 1986.

[3] B. Gains: An overview of knowledge acquisition and transfer, J. Man-Machine Studies, Vol.26, No.4, pp. 453-472, 1987.

[4] A. Kawaguchi, N. Taoka, R. Mizoguchi, T. Yamaguchi and O. Kakusho: An intelligent interview system for conceptual design of database, Proc. of ECAI, Brighton, Vol. 2, pp. 39-45, 1986.

[5] A. Kawaguchi, R. Mizoguchi and O. Kakusho: SIS: A shell for interview system, Proc. of IJCAI87, Milan, Vol. 1, pp. 359-361, 1987.

Assisting the Articulation of the Mental World

Koichi HORI*
Setsuo OHSUGA*

(*)Research Center for Advanced Science and Technology
The University of Tokyo
4-6-1 Komaba, Meguro-ku, Tokyo 153, JAPAN
Email: hori@ohsuga.u-tokyo.ac.jp

Abstract:
This paper gives a method to assist human experts to articulate the mental world. Articulation means to build the initial set of symbols. For assisting the articulation, we give a topological space which indirectly reflects the human mental world. We also give a topological space which shows the interpretation of what can be represented by a knowledge representation language. The user can reconsider the articulation of his world through the process of building knowledge representation as a compromised structure between his world and the internal world of knowledge representation.

1. Introduction

There have been an enormous number of works on knowledge acquisition. The authors of this paper have also had many experiences of building knowledge based systems[7][1][9]. Looking at those previous works, we find that one of the hardest difficulties of knowledge acquisition processes lies in that the experts of each domain do not necessarily have their knowledge in clear forms. The experts often say that they know something but that they cannot explain it any more clearly.

The starting point of this paper is the assumption that there is no one correct set of knowledge in any domain but there are only tacit and nebulous materials for building knowledge in the experts' mind, which cannot be always verbalized. The aim of this paper is to give a tool to assist experts to articulate the mental world by the experts themselves. The articulation means to determine boundaries of objects and to make structures composed of the objects[3]. The main difference between the articulation process and the previously studied knowledge acquisition process is that building the initial set of symbols is the main task in the articulation while constructing structures from pre-defined primitives is the main task in the knowledge acquisition. For example, an expert of an airplane design says "the design process of an wing is a process to seek for a beautiful wing." Moreover, they may say that it is not easy to formally formulate what is "beautiful". The knowledge of the expert has the potential to reveal its form in different forms. Their

knowledge is something like fluid which can change the form depending on the container of context. For example, some experts can be almost satisfied with the calculation of aerodynamic and structural characteristics of wings as the representation of "beautiful wing" and other experts think of different representation depending on their intention.

In many of previous studies on concept representation and knowledge representation, researchers sought for an appropriate set of primitives and a composing system for representing concepts or knowledge possessed by humans. However, there always remain semantic gaps between human world and knowledge representation language however well we design the language.

In this paper, we abandon to represent human concepts directly at some points, and instead, we give a method to reflect the mental world indirectly and we give a process to help the articulation of the worlds in order to build a knowledge representation as a compromise between the nebulous human world and the stubborn computer. For that purpose, we propose a method to reflect the nebulous mental world indirectly by topologically arranged words. Interpretations of internal knowledge representation are also presented topologically. Users can consider articulation of their world so as to build compromised world between their world and the internal world of knowledge representation by looking at and rearranging the two topological spaces.

2. Topological Space for Indirectly Representing the Mental World

We give a topological space for indirectly reflecting the nebulous mental world. For making a compromise between the human world and the computer. we also give another space which represents the interpretations of what can be done by the computer. Through these spaces, users reconsider the articulation of the worlds.

To use topology as a mathematical tool for analyzing concepts or knowledge is not new itself. We can find an early example in the work of the philosopher Husserl[5]. There was a work on data model based on topology of concepts for a database[10]. In the studies on pattern recognition, algorithms for the analysis of the similarity of figures were invented related to the mathematics of topology [6]. Some researchers including the author of this paper have considered topological structure of meanings of words in natural languages[8][2]. This paper gives another model of worlds based on topology but it differs from the previous works on the settings of the neighbors in the topological structures. The biggest reason we use topology as a mathematical tool is that it is flexible and powerful enough for reflecting any structure of concepts. The better feature of topological representation compared with other representations is that we can often find new concepts in empty places in the topological space, especially in metric space, and that it is easy to look at the world from different views in the topological representation.

First, we give a space which indirectly reflects the human mental world. In this space, elements are words which are used in human expert language, for example "wing", "aerodynamic characteristic" and so on. Topology is given for the set of the words by determining which words can be in nearest neighbor, where neighbor can correspond to any relation or association among the things meant by the words in the mental world. Which word can be a nearest neighbor

of which word is determined by the human user. Figure 1 gives an example of the topological representation of the world of feedback control system theory. (Definitions are explained later.)

It should be noted here that this space is pretty different from the previous models of semantics. In most of previous studies, we gave primitives to represent entities and relations in the real or mental world a priori and then described relations among language and the real or mental world. For example, we gave a primitive john0 for a dog named John in the real world and described that "John MEANS john0" in the semantic theory. In our formalism, we do not give such primitives as john0 but use the word in natural language "John" itself, because the starting point in our study is to give up the direct description of the real world. For some person, "John" might mean a dog named John which has a long ear. In the previous models, we should describe like "John MEANS john0 (has-a (ear1 (is-a ear)(has-property long)))". In our model, we also give up to give such primitives as "has-a", "is-a", because it is practically impossible and useless to enumerate such primitives for open applications. Instead, we use again the words "long", "ear" themselves, and we only indicate that things meant by those words - "long" and "ear" - have some relations with the thing meant by the word "John" and thus they are in neighbor of "John".

Compared with previous studies on conceptual modeling, this space lacks the functions on manipulating the semantics. It is the user's task to recall the content in his mental world looking at this topological space. The same topological structure might mean different mental world for different users; the same neighbor can mean any relation such as is-a, has-a, has-property and so on. However, this is not a deficit. It should be recalled here that we already have a knowledge representation language on one side. What we need is a clue to recall and reflect a user's mental world which might differ from the semantic world which can be represented by the knowledge representation language. It should be flexible

U={control_theory, control_system, process, control, input, output,
stability, controllability, robotics, chemics, cement, system,
element, transfer_function, step_response, impulse_response,
Laplas_transformation, stabilization, examination_of_stability,
root_loci, modern_control_theory, classic_control_theory,
objective_function, optimization, continuous_system, discrete_system,
Z_transformation, block_diagram, parameter, gain, delay, margin,
first_order_delay, 2nd_order_delay, pole, feedback, loop,
compensator, part, size, computer, funout, funin, overshoot,
undershoot, time_constant, differential_equation,
frequency_response,.....}

D={{stability, controllability}, {transfer_function, root_loci},
{transfer_function, differential_equation},
{differential_equation, Laplace_transformation},
{root_loci, pole}, {transfer_function, frequency_response},
{control_theory, control_system}, {control_theory, stability},
{stability, overshoot}. {stability, undershoot},.....}

Fig.1 An Example of a Topological Representation

enough to present any relations. If we want to explicitly show the types of relations, we can also put the words to show the relations in the topological space. We have proved through an experimental system that the idea to represent any relation or association as a topological neighbor is very powerful and useful. We can easily get different views to the same world and we can find new words in empty places.

Mathematically speaking, this space can be formulated as follows. The informations the users give little by little as a partial reflection of their worlds are

1) U : A set of words w1, w2, w3...wn.

2) D : A set of sets s1, s2, s3...sk, where each si is a set which consists of two words which have some relation.

For example, U={John, Jane, Frank, Mary}, D={{John, Jane},{John,Frank},{Jane, Frank}, {John, Mary},{Jane, Mary}}.

In this example, let's consider the case that Frank and Mary are children of John and Jane. The user knows the world of the family well but he does not know how to represent the world for knowledge representation. So, he just puts the words that come to his mind, which are John, Jane etc., and he indicates that John has close relation with Jane that John has close relation with Frank etc. For the set in D, the user gives a pair which can be recognized as the pair of words which has the closest relation for each word. This is equal to considering metric of distance among words in the form of ordering and to focus just on the nearest for each word. We may consider many other sorts of settings for capturing the world topologically but this setting is simple enough and strong enough.

The set D has the characteristic of symmetry but does not have transitivity. That is, if $\{x,y\}\in D$, it also means $\{y,x\}\in D$. But when $\{x,y\}\in D$ and $\{y,z\}\in D$, $\{x,z\}$ are not always the element of D.

This space gives a metric space. The distance between words can be defined as follows.

$d(x,x)=0$

$d(x,y)=a$ ($\{x,y\}\in D$)

$d(x,y)=b$ ($x\neq y$ and $\{x,y\}\notin D$)

where $0<a<b<2a$

(a and b are certain constants. The condition $0<a<b<2a$ is due to triangle axiom of metric space.)

This space is equivalent to discrete metric space as a topological space but has more information on distance (i.e. difference between a and b). (Discrete metric space is a space where any subset of U is a neighbor and any word is in the same distance to any word.)

We approximately transform this space into a two-dimensional Euclid space by the algorithm proposed by Kakusho and Mizoguchi[6]. This algorithm gives an Euclid space which conserves the original local order of distance as much as possible. In our system, the order simply consists of "nearest" or "not nearest". That is, in the resulted Euclid space, the nearest words are located in near distance for each word and the words which are not nearest are located farther.

For example, the family given above can be arranged as follows.

Mary
John Jane
Frank

Looking at this space, users can think of new words such as "is married to", "marry", "bear", "born in", "brother", "sister", "spouse", "parent of", "father of", "mother of", "son of", "daughter of" and so on. This example is very simple, but in the real problems, experts cannot give those words at the beginning but can easily give them looking at such space as shown above. Adding those words, we can rearrange the space farther. Rearranging the space, we add and change the words and change the structure composed of the words. This is equal to reconsidering the articulation of the world.

The articulation of the world should be biased by the internal world for knowledge acquisition. For the above example, which of "parent of" and "father of" is better should be determined by what the user wants to do and what can be done by the reasoning system. We show what can be done by the computer also by a topological space. This space shows interpretations on knowledge representation.

Strictly speaking, the representation of interpretation of knowledge representation language should be the representation of the relations between the language and the things which can be represented by the language. However, we just represent the interpretation by showing the interpreted things using the words in natural language. The metric space for the interpreted words is made according to the semantic structure of the reasoning system.

For example, some knowledge representation languages require users to distinguish "functional structure" from "physical structure" and to give relation between the "physical structure" and the "functional structure". The meanings of "physical structure" and "functional structure" are clear only to the people who have invented the knowledge representation language in many cases; Only those who know what and how the knowledge based system works can correctly know the meanings of those technical terms of knowledge representation language. Moreover, some other knowledge representation languages use different terms, for example, "object structure" and "object functionality", or ,say, "entity" and "relation". The relation between "object structure" and "object functionality" is slightly different from the relation between "physical structure" and "functional structure" and it is difficult for users to understand the difference.

In our system, the system shows the words in each domain previously stored in the knowledge base in separate topological spaces. For example, the system presents such words as "transfer function", "frequency response" and "root locus" used in the CAD system for feedback control system[1] in one space of previous experiences of knowledge representation. Most of the users, who are the experts in some domain, understand the word "transfer function" used in feedback control system design better than the word "object structure" used in the knowledge representation language, because the words used in the established domains usually have well-articulated meaning; Even the developer of the knowledge representation language cannot always determine whether something is "physical structure" or "functional structure" at once clearly. For example, are the expressions of transfer functions "physical

structure" or "functional structure"? Should they be represented by "terms" or by "predicates"? It should be determined by the relations among the "transfer functions" and other structures. Our system presents the words which have relations with "transfer functions" around the word "transfer function" in the metric space. These words have links to the internal knowledge representation. The user can pick up the words and know how those words were translated into the knowledge representation language in the previous experiences of knowledge representation.

The ideas mentioned so far are summarized in Figure 2.

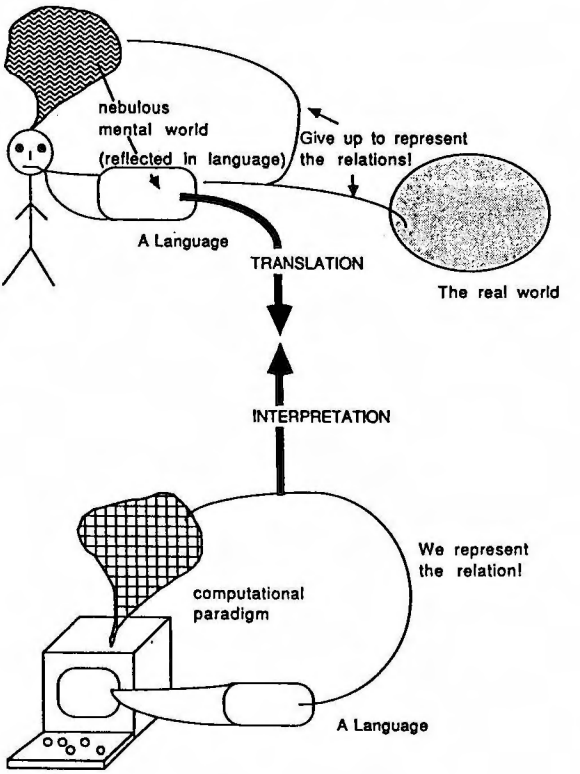


Fig.2 Interaction among the Worlds

3. Interaction between a Human Expert and a Computer

We have built a knowledge acquisition system based on the formalism described so far. The main aim of this knowledge acquisition system is to build the initial representation. In other words, the purpose of the system is the verbalization of tacit knowledge of human experts. We can say that most of the previous studies on knowledge acquisition systems have started from the point where the initial set of symbols for representing the domains are already prepared by the system designers. Our interest is in what computers can do for the evolution of symbols through the interaction between the internal world of computers and the external mental or real world.

In the current implementation, what the users can do mainly are what are mentioned so far, that is, they can put their words in their space, put neighbor relations among words, get configuration of the words in a metric space, browse the words in the spaces of previous examples, get the internal representation of the previous examples.

This system is connected neither to a knowledge editor nor to a knowledge debugger yet. The system will be connected to them in near future. Especially, we have a plan to connect the system to the meta reasoning system of KAUS[7]. The purpose of the connection is to give a user the environment of trial of reasoning on the fractions of knowledge representation. By this connection, the initial representation the user builds can be assured to work correctly. Although these connections have not been implemented yet, this system is useful enough at present as an advisor for building the first knowledge representation for a new problem. Practically, this system is also useful for building and structuring new ideas even if they might not be able to be converted to knowledge representation. The topological space of human language is powerful enough to stimulate the users for recalling or making up new ideas[4].

Figure 3 gives an example of the usage of the system.

The outline of the usage of the system is as follows.

- o The user gives some words which come to his mind first.
- o The words newly given by the user are located in arbitrary place in the space for this user.
- o The user can make another space for another context if she wants.
- o The user can give neighbor relation between words by picking up the words by mouse.
- o The system presents the configuration of words in Euclid space according to the principle mentioned in the previous section. The user can change the focus for the initial configuration to get different configuration of the metric space.
- o Looking at the spaces, the user adds words, changes words, adds neighbor relations and change neighbor relations.
- o When the user gives a new word, the system checks whether the same word is already known in previous experiences. If the system finds such ones, the system shows the configuration of those words in the previous examples with related words in separate spaces.
- o The user picks the word in the previous examples and knows how those words were translated into knowledge representation.
- o Looking at the previous examples, the user changes his own world so

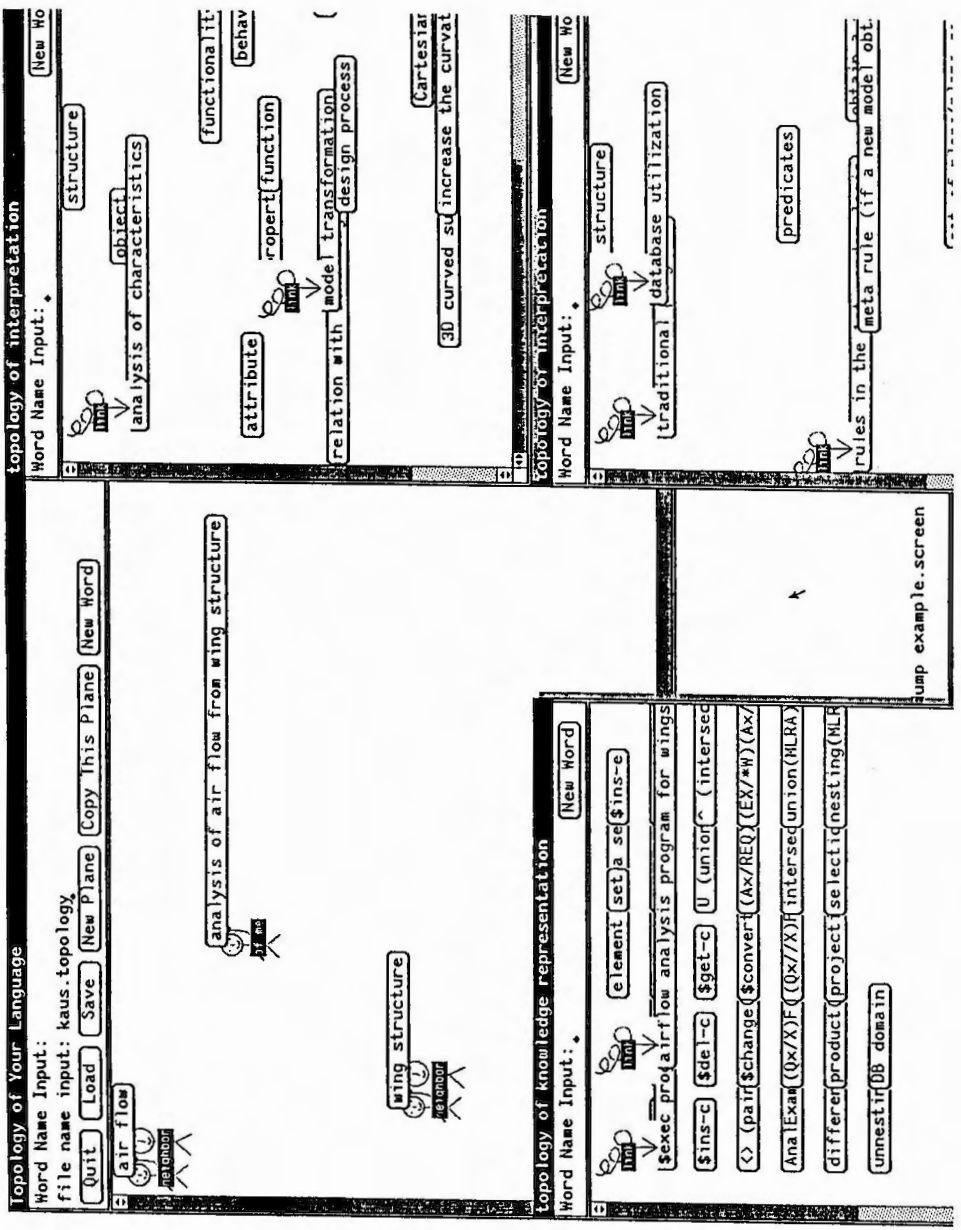


Fig.3 An Example of a Display of the Knowledge Acquisition System

as to make his world fit the world of knowledge representation.

As may be clear from this outline, it can be said that the system assists the users to make analogical reasoning for making knowledge representation. The actual work of analogical reasoning is done by the user but the system gives enough clues for this work. We think this framework is more practical than seeking for a machine which makes analogical reasoning automatically by itself.

4. Example

This section gives an example of articulation of the world through the compromise among human expert worlds and a knowledge representation.

We have already shown an example of a topological structure which reflects the mental world of an expert on feedback control theory in Figure 1. Looking at this structure, we can know that "root loci" has some relation with "transfer function" and so on. What we should do is to translate these relations to computational relations. For example, there are possibilities that an expert thinks of the relation between "root loci" and the "transfer function" as "root loci are calculated from transfer function" or "we must build a transfer function for satisfying the conditions on the root loci" or "bi-directional calculation between transfer function and root loci".

The translation depends on the user's intention and what can be represented by a knowledge representation language. For example, if the user intends to make a system to check the stability of a feedback control system, he maybe thinks of the display of root loci calculated from the transfer function of the system. Or some user may want a graphic display of root loci in which loci moved by a user inversely determines a transfer function. If the latter relation, i.e. determination of transfer function from root loci, can be achieved on computers, it will be very useful for the design of feedback control system.

The user must check here what can be done by the computers. If we already have examples of root loci and transfer function, the knowledge acquisition system shows those previous examples, and the user knows what kind of functionality can be achieved by the reasoning processes.

Let's consider the case that we do not have previous examples of the exact words of "transfer function" and "root loci" but the examples for wing design system.

Looking at the topological representation of the previous examples in our KAUS system, the user knows that there are many previous examples of the structure of computation of characteristics of objects from the structures of objects. In fact, this structure is one of the most recommended skeletons in KAUS [7]. There are two choices for the user here. One is to use this skeleton for his system, and the other is to invent a new style of usage of a knowledge representation language. Let's consider here the former case, because it is much easier for the user.

There are two choices again. One is to copy a previous example, and the other is to use a general skeleton. For example, if the user thinks that the previous example of "wing structure" realized by some data structure can be used for "transfer function", he can copy it. Or he can copy the general word "structure" which can be realized by several kinds of data structures for "transfer function". In the former case, we can copy many specific functions together. In the latter,

the room for several kinds of representation remains. Let's consider here the latter case. Now we have link "transfer function" to "structure", i.e. we translate "transfer function" to "structure". We can also link "root loci" to "characteristics".

Assuming that we have carried out similar operations to other words, we now consider the operations for writing the codes of knowledge representation. Since we have chosen general words of "structure" and "characteristics" in this example, we have many possibilities to select the real representation of "structure" and "characteristics". For example, "structure" can be realized by database structure or list structure or Cartesian product of sets or typed data structure, etc. The system shows the user the routes which connect the words. For example, the system shows how "characteristics" realized by values of variables can be computed from the "structure" realized by database structure. The user must select one style of representation for each interpretation. For example, let's assume that the user has determined that he represents the structure of transfer function by a graph structure in KAUS. This structure works well for calculating the characteristics of root loci. However, it is not good for retrieving the former design data, because the graph structure is not connected to the operation of store and retrieve in KAUS. There are three possible operations for connecting the structure with the operation of store and retrieve. The first is to find some operations connected to graph structure which can be interpreted as store and retrieve. The second is to find or make new connection between graph structure and database structure which is convenient for store and retrieve. The last is to change the structure of human world. For this example, the first and the second are not easy but the third works well. The user who has known that database structure is good for store and retrieve and that graph structure is good for the calculation of root loci can reconsider his intention. If he is clever enough, he can decide that he does not need to store the graph structure which shows the connection structure of feed back control system but just he needs to store and retrieve the parameters of gain and delay of each element in the control system. This means that he divides the structure of feed back control system to the structure of connection and the data structure of parameters. By this recognition of his problem, all the problems of computation can be solved.

All the processes mentioned above were the real processes for building an intelligent CAD system for a feedback control system [1]. These processes were achieved once only with the manuals of KAUS system when the system presented in this paper was not ready yet, and we have confirmed that those same processes can be achieved more efficiently when aided by the system presented in this paper.

5. Evaluation

We have realized, through some experiments, that our topological formalism and the knowledge acquisition system based on the formalism have two main big effects. One is flexible and effective structurization of human ideas. The other is an effective presentation of internal world.

As for the former point, we can know the effectiveness of our system presented in this paper from many of our experiences of building knowledge based systems[7]. Every time we built a knowledge based system, we had spent

much time for discussion with the human experts in each domain. We have known that the concepts in the expert's mind are often nebulous and they can be restructured flexibly. The space of semantics of human language effectively reflects the nebulous world of human expert and it greatly helps restructuring ideas. As for one experiment, when one user used our system for considering a new object oriented language, he said he could have found a new idea for the concept of message passing by rearranging the space of the words in his world. That is to say that our system works well as an aid system for inventing new ideas or revealing hidden concepts. The authors themselves also have used the topological space for structuring necessary ideas to build the experimental system for the knowledge acquisition itself.

As for the latter point, one of the basic ideas which lies in our formalism is the idea that we cannot represent the whole real world a priori but we can represent the whole world in the computer in the form which is easy to understand. Even if we limit the application domain to rather small domain, it is not easy to describe the domain in advance. For example, the same Newton equation in the mental world can have different effects for the expert of electronic engineering and the expert of civil engineering. Instead, we have described the internal world of computers from many different viewpoints in the space of the interpretation of knowledge representation language. This space effectively helps the users to understand what can be done by computers and the users can easily change his own structures of concepts.

6. Conclusions

We have given a method to reflect indirectly the mental and real world in topological space. We also represented the interpretations on knowledge representation topologically. We gave a system to reform the spaces through the interaction between the users' worlds and the worlds in the computer.

The human world was indirectly reflected as a topological structure where any relation between words is represented as a neighbor. The world of knowledge representation language was modeled as a topological structure which shows the semantic relations based on the reasoning system.

We have implemented a knowledge acquisition system which acquires knowledge by making compromise among the spaces. We gave examples of the compromise.

Practically, the success of our knowledge acquisition system depends on how rich examples we can give as the interpretations on knowledge representation. Since the authors already have many experiences of the usage of our knowledge representation system KAUS[7] especially for intelligent CAD systems, we are building the topological representations of interpretations for those applications. It is an interesting open topic to make an experiment of using these structures for other applications such as CAI, diagnosis and so on.

Acknowledgment

This work was done mainly during the stay of the first author in University of Compiegne, France. The authors greatly appreciate the discussion with the professors in University of Compiegne, especially Professor Pierre Morizet. The authors also greatly appreciate the discussion with Professor Aart Bijl of Edinburgh University.

References

[1] Guan,J., Ohsuga,S.: "An Intelligent Man-Machine System based on KAUS for Designing Feedback Control Systems", in Gero,J.S. ed., *Artificial Intelligence in Engineering: Design*, Elsevier Computational Mechanics Publications(1988).

[2] Hori,K., Toda,S., Yasunaga,H.: "Learning the Space of Word Meanings for Information Retrieval Systems", *Proc. 11th. International Conference on Computational Linguistics (COLING86)*, Bonn, West Germany, pp.374-379 (1986).

[3] Hori,K., Ohsuga,S.:"Articulation Problem - A Basic Problem for Information Modelling", *Joint Scandinavian-Japanese Seminar on Information Modelling and Knowledge Bases*, Ellivuori, Finland, June(1989).

[4] Hori,K., Ohsuga,S.:"Toward Computer Aided Creation", *Proc. 1st. Pacific Rim International Conference on Artificial Intelligence(PRICAI90)* (to appear 1990).

[5] Husserl,E.: *Ideen: Zu Einer Reinen Phaenomenologie und Phaenomenologischen Philosophie*, 3-1-72, Martinus Nijhoff, Haag(1950).

[6] Kakusho,O., Mizoguchi,R.: "A New Algorithm for Non-Linear Mapping with Application to Dimension and Cluster Analyses", *Pattern Recognition*, Vol.16, No.1, pp.109-117(1983).

[7] Ohsuga,S.: "Toward Intelligent CAD Systems", *Computer-Aided Design*, Vol. 21, No.5, pp.315-337(1989).

[8] Rieger,B.B.: "Semantic Relevance and Aspect Dependency in a Given Subject Domain: Contents-Driven Algorithmic Processing of Fuzzy Wordmeanings to Form Dynamic Stereotype Representations", *Proc. COLING84* (1984).

[9] Takasu,A., Ito,Y., Futatsugi,S., Ohsuga,S.: "Intelligent Wing Design Support System", *Proc. Scandinavia Artificial Intelligence Conference* (1989).

[10] Uchimani,S., Tezuka,Y.: "On the Formal Properties of the Topological Information Space Generative Grammar and Its Space", *Trans. IECE*, Vol. J63-D, No.1, pp.79-86(1980).

AUTOMATED VERSUS MANUAL KNOWLEDGE ACQUISITION: A COMPARISON IN A REAL DOMAIN

Francesco Bergadano, Research Associate
Attilio Giordana, Associate Professor
Lorenza Saitta, Full Professor

Dipartimento di Informatica
Universita` di Torino
Torino 10149 Italy

This paper describes the application of the learning system ENIGMA to the diagnosis of electromechanical devices at ENICHEM (Ravenna, Italy). The system ENIGMA is capable of learning structured knowledge from examples and a domain theory, using an integrated inductive/deductive paradigm. The results are compared with the ones obtained by a hand-coded expert system, designed for the same task, in which the knowledge base was acquired using the traditional method of expert interview. The comparison indicates that performances obtained by the learning system are systematically better than the ones obtained by the manually developed expert system. The conclusion is that, even if still leaving room for improvements, automated learning is a viable approach to the construction of expert systems, from the point of view of both obtainable performance and of limiting the development time and cost.

1. INTRODUCTION

It is widely recognized that the feasibility of expert systems exhibiting human like performances strongly depends upon the possibility of developing mechanisms for automating the processes of knowledge acquisition and maintenance. In the last decade, a number of research projects have been devoted to machine learning and knowledge acquisition. Although many steps ahead have been made, especially for the problem of learning concept descriptions from examples, we have still to recognize that the problem is, in general, extremely hard. This is confirmed by the fact that a number of learning systems have been described in the literature, but very few real applications have been addressed, in which machine learning proved to be capable of generating a knowledge base with the same (or better) performance as the one constructed by a human expert. We mention, in this sense, the results obtained by Michalski in developing automatic classification systems for agricultural [7] and medical [8] applications. In domains where the learning events can be represented as

vectors of <attribute, value> pairs, interesting results have been obtained using decision trees [11,12]. Many of the above mentioned results in classification systems have been obtained using mainly inductive methods; however, an important requirement, which characterizes many applications, is that the learned rules be understandable in the light of a pre-existing knowledge of the domain; this is particularly true for diagnostic systems.

This paper describes the work done, and the results obtained, in a pilot project aimed at checking the real possibilities offered by the state of the art in Machine Learning in order to automate the process of knowledge base construction for an expert system oriented to electromechanical troubleshooting. In order to achieve these goals, the prototype system ENIGMA, based on an integrated inductive/deductive paradigm [4], has been developed and an extensive experimentation has been performed, the most important aspects of which will be described in the following. The performance of the knowledge base acquired by ENIGMA has been compared with that of MEPS, a rule based Expert System, whose knowledge has been manually acquired by interviewing the domain expert [5]. However, performances are not the only useful parameter for the comparison: also knowledge understandability and meaningfulness and development time have been taken into consideration. The results have been considered encouraging enough to justify a large funding, by ENI, for a new project aimed at developing an industrial version of this learning system.

2. THE LEARNING PROBLEM

The case study has been supplied by the Enichem-Anic chemical plant at Ravenna, Italy. In this plant a technique of predictive maintenance, often called "on-condition" maintenance, has been used for a long time now. According to this technique, all the electromechanical apparata undergo periodical predictive controls, and a revision of a machine is performed only if it appears to be necessary. Then, the technicians are faced not only with the identification of faulty components, but also with a prognostic decision about the opportunity of halting the machine, balancing costs and possible dangers. About 1500 apparata, corresponding to more than 3800 machine components, are kept under this control. Should the technician detect, during the inspective control, something anomalous in the machine behavior, a more accurate analysis, called "Mechanalysis", is performed. Mechanalysis requires specially trained experts, with long experience.

The electromechanical apparata, diagnosed by means of mechanalysis, range from small motor-pumps (see, for instance, Fig. 3) to very large turbo-alternators supplying electric power to the whole factory. All these apparata share the common feature of possessing a rotating shaft to which various rotors are connected. When a machine possesses rotating elements, several unavoidable vibratory motions are induced in its parts; these vibrations occur also during the correct machine operation and are not dangerous as

long as their amplitude remains limited. When some fault occurs in the machine, new, anomalous vibrations appear, beside other manifestations. The aim of the mechanalysis is to locate faults and to diagnose their severity, through an analysis of these vibrations. Mechanalysis basically performs a Fourier analysis of the vibratory motions taken in prespecified and labeled points, precisely on the supports of the machine components. By means of a special *analyzer*, the technician obtains, for each support, the amplitude and velocity of the global vibration along the vertical, horizontal and axial direction; furthermore, the same data can be taken for each of the harmonic components of the vibrations. Also qualitative evaluation of the vibration phase can be done.
Mechanalysis has strong mathematical foundations in vibration theory, and, hence, the relationships between anomalous frequencies and faults could be, in principle, predicted. In practice, things are not so simple, as, usually, many more vibrations than those predicted occur; this is due to several reasons, such as mechanical imperfections in the parts, mutual influence among motions, resonance phenomena and fault co-occurrence. Moreover, a vibration does not begin abruptly, but its intensity increases over time, until a level, deemed to be dangerous, is eventually reached.

After an off-line analysis of the data gathered during the mechanalysis, the expert takes one of the following decisions:

- The machine operation is considered normal.
- A repetition of the mechanalysis, after some time, is suggested in the case the machine shows some anomalous symptoms, but not yet sufficient to predict the evolution of its behavior.
- A revision of the machine is recommended and the decision of immediately halting it or not is left to the department foreman.
- The machine must undergo a revision immediately.

In the last two cases the mechanalyst is often (but not always) able to precisely indicate the faulty component and the type of fault. It has to be noted that the mechanalysis does not always allow to precisely state what is going on in the machine; this is due to the fact that no one-to-one relationship between the vibration harmonics and the fault exists. In many cases, what really happened can be discovered only during the repair of the indicated component.

Then, the proposed learning task was that of learning automatically from examples of mechanalysis and with the help of background knowledge a knowledge base suitable to derive diagnoses of the type produced by human experts. This task can be seen as a classical case of learning concept descriptions [6], but shows many difficulties which are not present in other learning tasks described in the literature, residing both in the kind of available data and in the conceptualization of the problem.

First of all, the examples are complex, each one consisting, for the motor-pumps, of about 20 to 60 measures taken in different points and conditions of the machine. Each measure has 2 or 3 attributes (value, direction and frequency if appropriate). But, more than that, the examples are very noisy; in fact all the measures are affected by large uncertainty margins depending

both on the intrinsic limits of the measurement apparata and on the human subjectivity in recording the observed values. Another source of confusion comes from the simultaneous presence of vibrations associated to different faults, both because a multiple faults situation is quite frequent and because a rotating machinery has always many vibrating parts at various frequencies.

From the conceptual point of view, the principal difficulty resides in the fact that the expert's conclusion mainly arises from a global evaluation of the mechanalysis measures: a particular frequency pattern (or value) may acquire great relevance in a given context and may not be significant in another. Moreover, only few of the many measures are important for that particular diagnosis and the human expertise consists exactly in knowing how to identify them. This "globality" characteristic makes it difficult to define an adequate description language. It turns out, in fact, that the single mechanalysis measures are too low level as features and cannot be used directly to build up a description space. On the contrary, features of a higher level, defined in terms of groups of items, are to be introduced in order to describe hypotheses. In the current implementation, this form of "constructive" learning [10] has been strongly guided by the domain theory.

3. THE LEARNING SYSTEM

The system ENIGMA is basically an evolution of an earlier version, the system ML-SMART [2,3], enhanced in order to include deductive capabilities as described in [4]. In fact, several attempts to apply to the described case study the former version of ML-SMART, which was a purely inductive system, generated knowledge that was very difficult to understand in the light of the existing domain theory. We will not describe here the details of the system ENIGMA, being a detailed description already available in [1,3,5], but we will mention some points necessary to understand how the system has been applied.

ENIGMA receives in input a set of learning events and a body of background knowledge, represented as a Horn theory, and produces in output a structured knowledge base of classification rules. The peculiarity that makes this system suitable to deal with structured domains is that the learning events are described as vectors of items. Each item is in turn a vector of <attribute,value> pairs and corresponds to a part (subpattern) of a concept instance. In the present case the learning events correspond to the mechanalysis data, obtained through Fourier analysis, which are collected in a table of the type described in Fig. 1.

Sup-port	Direc-tion	Total Vibration		Fourier Analysis				
		Amplitude [μm]	Speed [mm/s]	ω [CPM]	v [mm/s]	.....	ω [CPM]	v [mm/s]
A	Hor	7/11	2.4/2.6	3000 i	0.7/0.9	.....	18K	0.7
	Vert	4/8	1.2/1.4	3000	0.2/0.7	.....	18K	0.4
	Ax	20	12	3000	3/3.2	.....	18K	0.8/1

Fig. 1 - Organization of the data collected during a mechanalysis.

Inside the table the data are arranged into groups of three rows; each group corresponds to a given support and each row to one spatial direction (Horizontal, Vertical and Axial). A first group of two columns (denoted by "Total Vibration") contains the measures of amplitude and velocity of the total vibration, whereas a second group (denoted by "Fourier Analysis") contains the measures of frequency, velocity and (possibly) phase of the harmonic components of the vibration. Notice that, for the harmonics, the measure of the velocity v (for which more reliable analyzers exist) allows the amplitude to be known as well, being this last proportional to v through the (known) ω. The measure consists, in some cases, of a single value (the index of the analyzer is stable), whereas, in others, of a range (the analog index of the analyzer oscillates between two extremes). This distinction is an important factor for the differential diagnosis. The qualitative behavior (stable, unstable, oscillating, rotating, fixed) of the vibration phase, when observed, is denoted by a letter attached to the corresponding frequency value. For instance, the "i", occurring in the first row of Fig. 1, denotes an instable phase.

A mechanalysis table is described to ENIGMA by supplying an item for each non empty entry. Each item is described by a vector of attributes characterizing the support, the direction, the amplitude, the type (total vibration or Fourier Analysis), the value of the measure, the "normal" value, and the rotation speed of the shaft. These attributes correspond to what in Explanation Based Learning are called "operational" predicates (elementary features). Other predicates (higher level features) can be defined using a Horn theory. In particular, it is possible to let ENIGMA work by applying pure EBG [9], if a complete theory, defining a non operational description of the concept, is given. However, this is practically impossible to achieve and ENIGMA was provided with a theory that only defines high level features capturing contextual information.

The rules learned by ENIGMA take the general form:

$$r: \quad H_j ; \ \varphi \overset{w}{\dashrightarrow} H_k \sqrt{H_l} \qquad (1)$$

Rule (1) can be interpreted as follows. Suppose that H_0 is the set of all classes and $h \in H_0$ is the concept to be identified in a given event f.

Suppose moreover, that, owing to some reasoning made by using other rules (of this type), we arrived at supposing that h can only belong to the subset $H_j \subset H_0$; then, if the assertion φ is

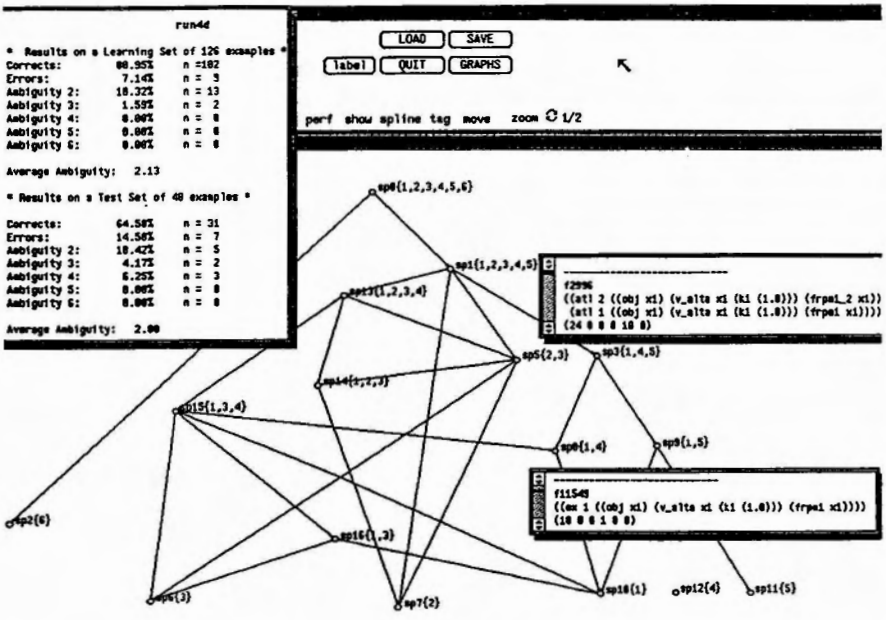


Fig. 2 - Graph of production rules generated by ENIGMA

supposing that h can only belong to the subset $H_j \subset H_0$; then, if the assertion φ is

verified on f, the rule (1) concludes that h belongs to H_k with probability w or to H_l with probability (1-w). The relations $H_k \cap H_l = \emptyset$ and $H_j \supset H_k, H_l$ always hold. H_j is called the *context* of the rule, H_k the *primary implication* and H_l the *secondary implication*. If the probability is w=1, the secondary implication is not present. As a special case, the set H_k may consist of a single concept h.

The assertion φ is a first order logic formula, expressed with operational predicates only. Numerical quantifiers such as Atleast n, Atmost n, Exactly n and negation are also allowed. Usually, the primary implication of a rule coincides with the context of another formula. Formulas having the same primary implication are implicitly considered as OR-ed. As a consequence, a knowledge base learned by ENIGMA can be described in a graph, as shown in Fig. 2. ENIGMA produces such a graph of rules by searching in a

space of rules through a specialization process, which starts from the most general formula true -> H_0 and generates more and more specific formulas until classification rules are discovered. This process is guided by statistical heuristics which trades off consistency and completeness criteria and by the background knowledge supplied at the beginning. The strategies and the heuristics are described in [2,3,4,5].

4. THE LEARNING SET

All the experiments have been performed using a set F_0 of N=209 mechanalysis tables (examples), filled by an experienced domain expert and referring to diagnoses of motor-pumps. In Fig. 3 the general scheme of a horizontal centrifugal pump is reported.

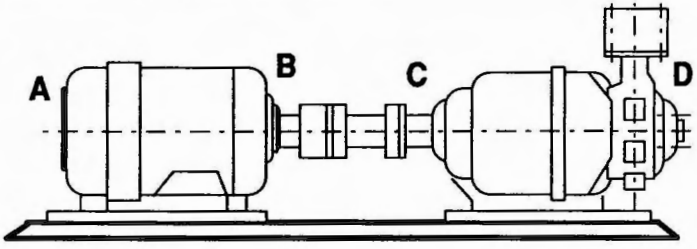


Fig. 3 - Scheme of a horizontal centrifugal pump

The vibrations in the four supports A, B (motor) and C, D (pump) are marked.
The rolling bearings are located in the supports. An elastic joint connects the pump's and the motor's shafts. Furthermore, the basement of the machine is anchored to the ground.

The considered faults can be grouped into six classes:

C_1 = Problems in the joint
C_2 = Faulty bearings
C_3 = Mechanical loosening
C_4 = Basement distortion
C_5 = Unbalance
C_6 = Normal operation conditions

However, as mentioned in Section 2, these faults rarely occur in isolation and, even then, it is not always possible to individuate them precisely. Thus, not all the N examples have been univoquely classified by the human expert, but, on the contrary, the diagnoses generated by him followed the taxonomy reported in Fig. 4.

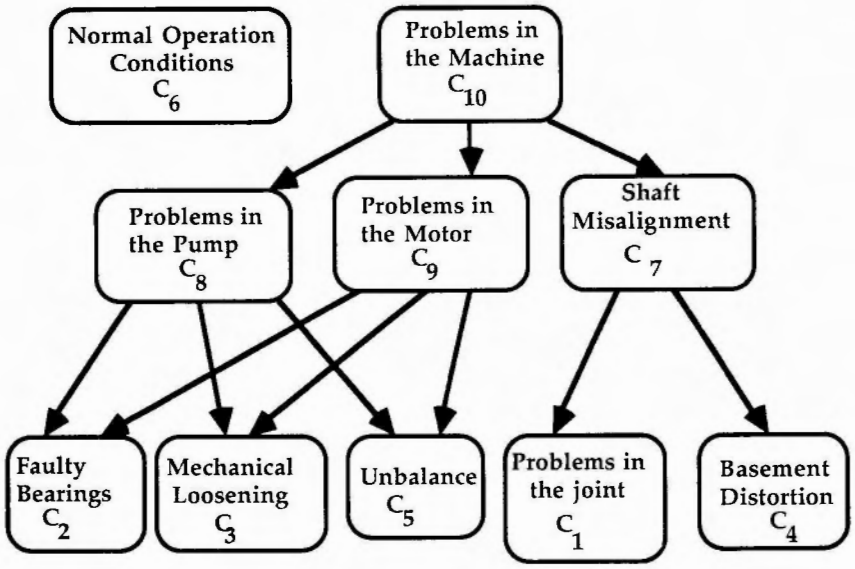


Fig. 4 - Diagnostic taxonomy of the motor-pump faults

According to this diagnostic taxonomy, the following intermediate classes have been used by the expert:

C_7 = Shaft misalignment	$(C_7 = C_1 \cup C_4)$
C_8 = Problems in the pump	$(C_8 = C_2 \cup C_3 \cup C_5)$
C_9 = Problems in the electric motor	$(C_9 = C_2 \cup C_3 \cup C_5)$
C_{10} = Problems in the machine	$(C_{10} = C_8 \cup C_9)$

The *ambiguity* $\alpha(f)$ of a classified example f is the minimum number of classes f is hypothesized to belong to; for instance, an example f classified in class C_8 has an ambiguity $\alpha(f) = 3$. Moreover, a diagnosis H_i will be said *more specific* that a diagnosis H_j iff H_j is an ancestor of H_i in the diagnostic taxonomy. Obviously, we desire that the automated system be at least as specific as the expert was.

In Table I the expert's classification of the N examples is reported; the average ambiguity of the examples is $\alpha = 2.08$. Notice that, whereas an internal node of the diagnostic taxonomy denotes uncertainty about the right choice among the descendant nodes, the intersection $C_i \cap C_j$ denotes co-occurrence of both C_i and C_j; for this reason, the ambiguity of the diagnosis $C_i \cap C_j$ is evaluated as the minimum between those assigned to C_i and C_j. Regarding the expert's classification, it has to be pointed out that, in many cases, the expert was actually able to generate a less ambiguous hypothesis than the one reported in

Table I. However, he judged this more specific diagnosis as out of reach for a system not acquainted with a deeper understanding of the domain; then, he indicated what, in his opinion, was an acceptable answer for an automated system, working only on the basis of the mechanalysis data. Moreover, the generation of a more precise diagnosis, by the part of the expert, entrained an error rate of about 5% (according to the expert's subjective estimate), whereas the diagnoses reported in Table I are the most specific ones, which are still error free. In this context, we say that an *error* occurred when the true class is not included in the set of the proposed ones.

5. RESULTS OBTAINED USING THE EXPERT SYSTEM MEPS

MEPS is a prototype expert system [6] developed manually by means of interviews with the same domain expert who supplied the classified examples. MEPS' knowledge is represented by means of rules and frames and contains both diagnostic and structural information. The chosen implementation environment is the GOLDWORKS shell on an IBM Personal Computer. The system contains about 290 diagnostic rules and 70 structural frames. Its representation language is a first order logic based language with an associated continuous-valued semantics. The process of designing and implementing the MEPS prototype took about 18 months, 12 of which devoted to the acquisition, encoding and maintainance of the knowledge base.
For what concerns the interviews with the expert (who has been very cooperative and enthusiastic about the project), it has to be noted that the considered application required, by the part of the knowledge engineers, very specialized mechanical engineering knowledge. Without this last, even to start speaking with the expert would have been impossible; fortunately, two of the authors had part of this knowledge in their cultural background, but, nevertheless, a lot of time was spent in recalling and completing it in order to be able to understand what the expert was explaining.

In Table II the results obtained from the expert system MEPS are reported. MEPS performs, on a given case, an evidential reasoning and generates, as a result, a list of possible faults, ordered according to decreasing value of evidence. The recognition rate has been evaluated in two ways, a pessimistic and an optimistic one. In the pessimistic case, only the best scored hypothesis h_{best} has been considered; if h_{best} is a descendant, in the diagnostic taxonomy, of the node corresponding to the diagnosis given by the expert, then h_{best} is considered correct (the system gives an answer of equal or higher specificity then that of the expert). In the optimistic case, MEPS' diagnosis is considered correct if the set of generated hypotheses contains at least one hypothesis more or equally specific than the one of the expert. The resulting pessimistic recognition rate is $\eta = 0.89$, whereas the optimistic one is $\eta^* = 0.95$. Finally, the average ambiguity of the classification was $\alpha = 1.46$.

Table I
Diagnoses generated by the expert for the examples of the set F_0.
The classes correspond to the taxonomy of Fig. 4.

Class	No. of Examples	Ambiguity	Class	No. of Examples	Ambiguity
C_{10}	29	5	$C_7 \cap C_5$	2	1
C_7	42	2	$C_7 \cap C_2$	2	1
C_8	26	3	$C_1 \cap C_3$	2	1
C_9	8	3	$C_1 \cap C_8$	1	1
C_1	13	1	$C_7 \cap C_3$	1	1
C_2	23	1	$C_1 \cap C_2$	4	1
C_3	6	1	$C_1 \cap C_9$	1	1
C_4	5	1	$C_4 \cap C_9$	1	1
C_5	13	1	$C_4 \cap C_8$	1	1
C_6	27	1	$C_3 \cap C_4$	1	1
			$C_1 \cap C_5$	1	1

Table II
Results of MEPS on the examples of the set F_0

Number of Generated Hypotheses (Ambiguity)	Number of Cases	The best hypothesis was the correct one	The right hypothesis was included in the proposed set
1	131	122	122
2	60	49	59
3	18	15	18

6. RESULTS OBTAINED USING ENIGMA

The learning experiments with ENIGMA were performed exploiting the incremental abilities of the system and consisted of a sequence of six runs (phases). At the beginning, the global set F_0 of cases has been divided into two

subsets, one of which, containing 42 randomly chosen examples, served as fixed test set (FTS) for each run. The remaining 167 examples have been randomly partitioned, in the first phase, into a learning set LS_1 of 60 cases and a variable test set VTS_1 of 107 cases. ENIGMA was run on LS_1 and was tested on LS_1, VTS_1 and FTS. In the second phase, a subset of VTS_1 has been added to LS_1, obtaining a new learning set LS_2, on which ENIGMA was run again. This process has been repeated five times and the results are reported in Table III.

Table III
Results of the automated learning system ENIGMA

	Phase 1	Phase 2	Phase 3	Phase 4	Phase 5	Phase 6
Cardinality of the Learning Set (LS)	60	80	104	125	145	167
Cardinality of the Variable Test Set (VTS)	107	87	63	42	22	0
Cardinality of the Fixed Test Set (FTS)	42	42	42	42	42	42
Recognition Rate Learning Set (LS)	0.97	0.99	0.96	0.98	0.97	0.96
Recognition Rate Variable Test Set (VTS)	0.93	0.94	0.94	0.90	0.91	//
Recognition Rate Fixed Test Set (FTS)	0.90	0.90	0.86	0.90	0.86	0.86
Recognition Rate Complete Set	0.94	0.95	0.93	0.95	0.94	0.94
Average Ambiguity	1.15	1.24	1.18	1.18	1.23	1.19
Number of subproblems	13	13	13	17	18	18
Number of Generated Formulas	2788	3817	4067	1801	1643	1635
Number of Rules in the Knowledge Base	39	70	128	131	142	147
Number of Deleted Rules	//	20	28	40	46	47
Number of Added Rules	//	51	86	43	57	52
CPU time (sec)	47826	43647	44252	36464	19992	19876

7. DISCUSSION

An interesting comment about the results of the automatic learning is that the performances are quite stable by varying the number of training examples; actually, there is a slight peak at the forth run, however not statistically significant. This counter-intuitive phenomenon is due to the fact that, because of the ambiguity inherent in the examples, adding new ones to the training set does not contribute any new information, but, on the contrary, mixes up information from different faults. In fact, by denoting by α_j(LS) the average ambiguity of the learning set used in the j-th phase, we notice a steady increment in the confusion of this set, which is responsible for the above mentioned phenomenon. More precisely, the main confusion arises among the five fault classes, whereas the examples in class C_6 (non faulty machine) are always good example; in fact, both the expert system MEPS and the knowledge generated by ENIGMA never confuses among faulty and non faulty machines. Table IV summarizes the values of the ambiguity for the involved sets; from this table it is apparent that the training examples added in Phase 5 are the worst ones.

Table IV
Ambiguity of the learning set in each learning phase. The last column refers to examples of classes C_1-C_5.

	Ambiguity in the added Learning Set	Ambiguity in the global Learning Set	Ambiguity in the fault examples of the Learning Set
Phase 1	/ /	1.98	2.18
Phase 2	2.23	2.04	2.24
Phase 3	2.36	2.11	2.31
Phase 4	2.23	2.13	2.33
Phase 5	2.85	2.23	2.43
Phase 6	2.36	2.25	2.44

As it is not possible to select more "clean" examples than those used in these experimentations, the only way to further increase the performances seems to be that of supplying both MEPS and ENIGMA with a deep model of the domain, allowing a more complex, but more subtle, reasoning to be performed. The evolution of the approach toward this direction is under development and preliminary encouraging results have already been obtained [5].

References

[1] Aho and Ullman. Universality of data retrieval languages. In *Proc. ACM symp. on principles of programming languages*, 1979.

[2] F. Bergadano, F. Brancadori, A. Giordana, and L. Saitta. A system that learns diagnostic knowledge in a database framework. In *Proc. of the IJCAI89 workshop on knowledge discovery in databases*, Detroit, Michigan, 89.

[3] F. Bergadano, R. Gemello, A. Giordana, and L. Saitta. Smart: a problem solver for learning from examples. *Fundamenta Informaticae*, 12:29–50, 1989. Elsevier, Amsterdam.

[4] F. Bergadano and A. Giordana. Pattern classification: an approximate reasoning framework. *International Journal of Intelligent Systems*, 4(1):23–44, 1989. John Wiley and sons, New York.

[5] F. Bergadano, A. Giordana, and L. Saitta. Automated concept acquisition in noisy environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(4):555–577, 1988. New York.

[6] F. Brancadori and S. Radicchi. Acquisizione automatica della conoscenza. *Informatica Oggi*, 9(53):81–93, 1989.

[7] B. Buchanan and T. Mitchell. Model-driven learning of production rules. In D. Waterman and F. Hayes-Roth, editors, *Pattern-directed inference systems*, pages 297–312, Academic press, New York, 1978.

[8] G. F. DeJong and R. J. Mooney. Explanation based learning: an alternative view. *Machine Learning*, 1, 1986.

[9] G. Gardarin and E. Simon. Les systemes de gestion de bases de donnees deductives. *Synthese*, 6(5), 1987.

[10] F. Hayes-Roth and J. McDermott. An interference matching technique fo inducing abstractions. *Communications of the ACM*, 21:401–411, 1978.

[11] L. J. Henschen and S. A. Naqvi. On compiling queries in recursive, first order databases. *JACM*, 31(1), 1984.

[12] W. Iba, J. Wogulis, and P. Langley. Trading off simplicity and coverage in incremental concept learning. In *Proc. of the Fifth Int. Conf. on Machine Learning*, pages 73–79, Ann Arbor, MI, 1988.

[13] Y. Kodratoff. Improving the generalization step in learning. In R. S. Michalski, J. G. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 215–244, Morgan Kaufmann, Palo Alto, CA, 1986.

[14] E. L. Lozinskii. Evaluating queries in deductive databases by generating. In *Proc. of the IJCAI*, Los Angeles, CA, 1985.

[15] R. S. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence*, 20:111–161, 1983.

[16] R. S. Michalski and R. L. Chilauski. Learning by being told and learning from examples: an experimental comparison of the two methods of knowledge acquisition. *Int. Journal of Policy Analysis and Information Systems*, 4(2), 1980.

[17] R. S. Michalski, J. Hong, N. Lavrac, and I. Mozetic. The aq15 inductive learning system: an overview and experiments. In *Proc. of the First Int. Meeting on Advances in Learning*, Les Arcs, France, 1986.

[18] S. Minton, J. G. Carbonell, C. A. Knoblock, D. R. Kuokka, O. Etzioni, and Y. Gil. Explanation based learning: a problem solving perspective. *Artificial Intelligence*, 40:63–118, 1989.

[19] T. Mitchell, R. M. Keller, and S. Kedar-Cabelli. Explanation based generalization: a unifying view. *Machine Learning*, 1:47–80, 1986.

[20] S. Muggleton. Machine invention of first order predicates by inverting resolution. In *Proc. of the Fifth Int. Conf. on Machine Learning*, pages 339–352, Ann Arbor, MI, 1988.

[21] R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[22] L. Rendell. Learning hard concepts. In *Proc. of the 3rd european working sessions on learning*, pages 177–200, Glasgow, UK, 1988.

[23] J. D. Ullman. *Principles of database systems.* Computer Science Press, second edition, 1982.

[24] P. Utgoff. *Machine Learning of Inductive Bias.* Kluwer Academic Publ., 1986.

[25] S. A. Vere. Induction of concepts in the predicate calculus. In *Proc. of the IJCAI*, pages 351–356, Tbilisi, USSR, 1975.

[26] L. Vieille. Recursive axioms in deductive databases: the query sub-query approach. In *Proc. of the First Int. Conf. on expert database systems*, Charlston, SC, 1986.

[27] P. H. Winston. Learning structural descriptions from examples. In P. H. Winston, editor, *The Psychology of Computer Vision*, McGraw Hill, 1975.

[28] F. Bergadano and A. Giordana. A knowledge intensive approach to concept induction. In *Proc. of the Fifth Int. Conf. on Machine Learning*, pages 305–317, Morgan Kaufmann, Ann Arbor, MI, 1988.

A Proposal Guided Knowledge Acquisition Support System

Hirokazu Taki*and* Satoshi Terasaki
ICOT Research Center,
Institute for New Generation Computer Technology,
4-28, Mita 1-Chome, Minato-ku, Tokyo 108 Japan
e-mail: htaki@icot.or.jp, terasaki@icot.or.jp

Abstract

This paper describes a proposal guided knowledge acquisition support system which generates and proposes candidates of knowledge by induction. This system is an enhanced version of the interactive knowledge acquisition support system EPSILON/One. This system elicits basic information from an expert, makes hypothesis, and modifies target knowledge with the expert. Some of the major problems of using induction is that it requires a large amount of computing power and creates numerous hypotheses. In order to make hypotheses by induction more efficient, the forms of hypotheses must be limited. EPSILON/One has a knowledge representation, Expert Model, which consists of operations. The operations are limited to seven types which are derived from analyzing real diagnostic knowledge bases. The system realizes efficient inductive hypothesis generation by using this limited numbers of o peration representations. This papers discusses inductive operation presumption algorithms and interactions between the expert and the system in this architecture.

1 Introduction

A knowledge acquisition support system builds a knowledge base by interviewing an expert[1]. The knowledge base must contain general knowledge to be applied to various cases[2]. The knowledge acquisition support system elicits target domain general knowledge directly from the expert by interview[3], while a learning system generates general knowledge from examples. This paper describes a knowledge acquisition system that has not only an interactive knowledge acquisition function that elicits general knowledge from the human expert, but also an induction function which presumes general knowledge from examples. Both the knowledge acquisition support system and the induction system have advantages and disadvantages. The induction system generates general knowledge from examples, but there are two disadvantages in generating this knowledge. It creates numerous candidate hypotheses of knowledge and requires a large amount of computing power. To reduce hypotheses and calculative complexity, the target form/language generated by induction must be limited. If the expert is not aware of general knowledge, there is a problem occurs where the interactive knowledge acquisition cannot extract enough general knowledge. We have evaluated our

*Current position: Information Systems and Electoronics Development Laboratory, Mitsubishi Electoric Corporation. e-mail:taki@isl.melco.co.jp

interactive knowledge acquisition system, EPSILON/One[4], and found that it must have presumed candidates of knowledge from example in some knowledge acquisition cases. The Expert Model[5], the knowledge representation of EPSILON/One, consists of seven types of operations which represent problem solving knowledge. These operation types limit the form of knowledge to be acquired. We also tried to use the special feature of the Expert Model to limit the form of consequence of induction. The knowledge acquisition support system with the induction function has a function for making proposals as general knowledge. We call this system the proposal guided knowledge acquisition support system. This paper introduces operation (knowledge) presumption and the proposal guided knowledge acquisition support system architecture.

2 Knowledge Acquisition and Induction

2.1 Knowledge Acquisition by Interview and Problems

The basic functions of the knowledge acquisition support system are knowledge elicitation, knowledge arrangement, knowledge refinement, and knowledge representation translation. A conventional approach to knowledge elicitation is through a psychological method to elicit general knowledge from the expert. The normal knowledge elicitation method stimulates the expert to remember new knowledge by redisplaying raw information or reformed knowledge which has been extracted from the expert. The personal construct psychology[6][7] and the pre-post method[8] which are supported by EPSILON/One are two examples of psychological methods. It is difficult for the expert to represent knowledge when he doesn't arrange and recognize knowledge related to problem solving[9]. However, when a kind of knowledge is proposed, he can judge easily whether it is used by him or not. Therefore, in order to enhance the knowledge elicitation function of the knowledge acquisition support system, it should propose candidates of knowledge for the expert. An example of a function which proposes candidates of knowledge is a function which can generate hypotheses that are more general than raw information extracted from the expert. Abduction, analogy, and induction are well known inference methods for generating hypotheses.

2.2 Integration of Knowledge Acquisition and Learning

Both the knowledge acquisition support system and the learning system are used for knowledge base building[10][11]. However, they each have different approaches to building knowledge bases. The former acquires general knowledge from the expert directly. The latter generates hypotheses as knowledge from given information by advanced inference. A method of knowledge generation is a concept formation by inductive generalization. It creates general knowledge from examples. There are some problems in using induction for generating knowledge. It generates numerous hypothesis candidates and its calculation cost is high. For efficient induction, a language (form) of hypothesis representation must be limited to reduce candidates of knowledge, and domain specific knowledge must be used in the induction process to eliminate hypotheses[12]. We call the latter learning heuristics. A knowledge representation of problem solving is limited for efficient knowledge elicitation and refinement knowledge in the interactive knowledge acquisition. This limitation can also be used for language limitation of hypothesis generation by induction. Learning heuristics are dependent on the target domain. Therefore, the expert must cooperate with the induction system

to reduce hypotheses by using heuristics. The interactive knowledge acquisition method is suitable for cooperative jobs with the expert. We explained that the knowledge acquisition support system and the learning system each compliment one another. A knowledge acquisition support system which is integrated into the learning system, can build a knowledge base by interviewing general knowledge and examples.

2.3 Knowledge Elicitation and Proposal

Basic functions of the knowledge acquisition support system with inductive functions are example elicitation, proposal generation by induction, and interactive editing proposal knowledge. The expert can require the next hypothesis, check modified knowledge using an inference engine, and add new example. These interactions between the proposal guided knowledge acquisition support system and the expert are shown in Fig.1.

3 Knowledge Acquisition Using Expert Model and Induction

3.1 Expert Model and Knowledge Acquisition

The knowledge representation of EPSILON/One is the Expert Model[5]. It consists of seven types of operations. They are *classification, selection, input, output, ordering, combination,* and *translation.* These types are derived from analyzed rules of knowledge bases in production rule form for diagnosis. The structure of the expert model consists of operations, a script which contains operation relations and inference control knowledge, evaluators which are the contents of operations, and elements which are evaluated in operations, as shown in Fig. 2. EPSILON/One supports a knowledge elicitation method (pre-post method) using the expert model. It consists of three stages; (1) operations and script elicitation, (2) evaluator elicitation using operation types and (3) element elicitation. In the second stage, knowledge is extracted effectively using operation type limitation. However, the knowledge to be extracted must be general knowledge. Therefore, if the expert is not aware of general evaluation criterion of an operation, the knowledge becomes very difficult to extract. However, it is not difficult for the expert to show examples which are input and output elements of the operation. The knowledge acquisition support system must contain a function which presumes general evaluation criterion from input and output element examples of the operation.

3.2 Inductive Knowledge Presumption Algorithm Using Operation Types

Operation presumption algorithms which uses operation types for consequence limitation of induction are introduced in this section. The seven operation types are *classification, selection, input, output, ordering, combination,* and *translation.* The translation operation consists of three sub operations; *attribute value replacement operation, mathematical calculation operation* and *element decomposition operation.* To use induction, we must prepare *language which represents a target concept, positive and negative examples related to the target concept, knowledge which defines generalization of the concept,* and *induction algorithms.*

- Language which represents a target concept: Evaluation criterion for each operation type A basic criterion is related to one attribute.

- Positive and negative example related to the target concept: Input and output elements of an operation
- Knowledge which defines generalization of the concept: Generalization method for each evaluation criterion
- Induction algorithms: Operation presumption algorithms for each evaluation criterion

EPSILON/One supports an efficient operation presumption function using operation types to limit language of inference consequence, treatment of examples, and methods of criterion generalization[13]. Here, we introduce three operation presumption algorithms; *selection and classification operation*, *attribute value replacement operation* and *ordering operation*. Currently, the algorithm of the combination operation is fixed and doesn't require any criterion. Solving the evaluator of the mathematical calculation operation is equivalent to solving simultaneous equations. Therefore, we don't prepare induction algorithms for combination operations and calculation operations. It is not necessary to presume evaluation criterion of *input operation*, *output operation*, and *element decomposition operation* because criteria of these operations are easily extracted from the expert. We explain the following items for three operation presumption algorithms:

- Evaluation algorithms of operations
- Evaluation criterion for each evaluation algorithm
- Treatment of examples
- Evaluation criterion presumption algorithms

3.3 Selection and Classification Operation Presumption

The selection operation selects elements which are suitable for a selection criterion from a given set. Classification operation classifies elements of a given set into sub sets according to classification criteria. This operation can be realized to combine several selection operations.

(a) Selection Algorithm

If an attribute and its value of an element satisfy a selection criterion, then the element is selected. The selection criterion consists of an attribute and its possible value sets or a conjunction of attributes and their possible value sets. If an element satisfies the selection criterion, the value of the attribute of the element is included in a value set of the criterion.

(b) Selection Criterion

A selection criterion related to one attribute is defined as follows:
(attribute-name, a set of possible values)

(c) Treatment of Examples

Examples are given as multiple elements of an input element group and an output element group. Elements selected by a selection operation are positive examples which represent a

concept of selection criterion. Elements which aren't selected by the selection operation are negative examples. These examples (elements) are defined by the expert. Elements for examples have some attributes which have one value for each attribute.

Example-1: Given an input and an output element group as follows.
Input element group = {e1, e2, ..., ei, ei+1, ..., en}
Output element group = {e1, e2, ..., ei} , ej (j = 1 ...n) means an element.
then
Positive examples = {e1, e2, ..., ei}
Negative examples = {ei+1, ..., en}

(d) Selection Criteria Presumption Algorithm
Step1: Making a set of attributes which are included in all examples. This set is called "a total attribute list".
Step2: Making sets of values of each attribute in the total attribute list from positive examples. If an element doesn't have an attribute, its value is "undefined".
Step3: Making sets of values of each attribute in the attribute set from negative examples.
A matrix of the total attribute list (aj, j = 1 ...m) and elements (ek, k = 1 ...n) with values vjk (k = 1 ...n, j = 1 ...m), is made from the results of step2 and step3.

Positive	$a1$	$a2$	$a3$	...	am
$e1:$	$v11$	$v21$	$v31$	...	$vm1$
$e2:$	$v12$	$v22$	$v32$	...	$vm2$
		$\vdots$			
$ei:$	$v1i$	$v2i$	$v3i$	...	vmi
Negative					
$ei+1:$	$v1i+1$	$v2i+1$	$v3i+1$	...	$vmi+1$
$ei+2:$	$v1i+2$	$v2i+2$	$v3i+2$	...	$vmi+2$
		$\vdots$			
$en:$	$v1n$	$v2n$	$v3n$	...	vmn

Sets of values for an attribute *aa* are defined as P(aa) when made from positive examples and N(aa) when made from negative examples.

$$
\begin{aligned}
P(a1) &= \{v11, v12, \ldots, v1i\} \\
P(a2) &= \{v21, v22, \ldots, v2i\} \\
&\vdots \\
P(am) &= \{vm1, vm2, \ldots, vmi\} \\
N(a1) &= \{v1i+1, v1i+2, \ldots, v1n\} \\
N(a2) &= \{v2i+1, v2i+2, \ldots, v2n\} \\
&\vdots \\
N(am) &= \{vmi+1, vmi+2, \ldots, vmn\}
\end{aligned}
$$

Step4: Checking intersection of P(aa) and N(aa).

if P(aj) $\cap$ N(aj) = $\emptyset$ (empty set)
  then select attribute aj
  else reject attribute aj , (j = 1 ... m)

If a selected attribute set is {aa, ab, ac}, P(aa), P(ab) or P(ac) is a selection criterion.

Caution: As P(aj) $\cap$ N(aj) = $\emptyset$ (empty set), $\neg$ N(aa) $\cap \neg$ N(ab) $\cap \neg$ N(ac) is also a selection criterion. $\neg$ N(aa) means that the value of the attribute *aa* isn't included in N(aa).

Step5: Showing selection criterion.
If there are multiple criteria, the system shows them in order. In this example, the system shows P(aa), P(ab), P(ac), and $\neg$ N(aa) $\cap \neg$ N(ab) $\cap \neg$ N(ac) in this order.

(e) Guidance of generalization of selection criterion
The system shows how to make wider selection criterion by a set of attribute value generalization.

Example-2:	{ 1, 2, 3, 4, 5 }	$\rightarrow$	{>= 1, <= 5}
		$\rightarrow$	Integer
	{ small-apple, big-apple }	$\rightarrow$	{ apple }
		$\rightarrow$	{ fruit }

The system shows existence of elements which don't belong to either positive or negative examples.

Example-3: P(aa) = { 1, 2, 3 }, N(aa) = { 5, 6, 7 }

An element, where the value of an attribute *aa* is 4, doesn't belong to either P(aa) or N(aa). If a selection criterion is P(aa), this element is recognized as a negative example. If a selection criterion is $\neg$ N(aa), this element is recognized as a positive example. Therefore, $\neg$ N(aa) is a more general selection criterion than P(aa).

If there is no attribute aj which satisfies at P(aj) $\cap$ N(aj) = $\emptyset$, then there is some dependency among attributes. In this case, the system must make selection criteria containing multiple attributes. (cf section 3.6).

3.4 Ordering Operation Presumption

(a) Ordering Algorithm
The ordering operation sorts elements according to ordering criterion related to one attribute. This ordering criterion is a total order. An order of elements means an ordering of processing elements in an operation.

(b) Ordering Criterion
An ordering criterion of one attribute is defined as follows:

(attribute-name, order list of values), (attribute-name, ascending),
or (attribute-name, descending)

(c) Treatment of examples

Examples are elements of an input element group and an output element group. An order of elements in the output element group is an ordering criterion of positive examples. Negative examples are relations of two items which differ from the input element group and the output element group. The input element group is not always necessary. Because there is no intersection between positive and negative examples, and negative examples aren't useful in limiting a criterion made from positive examples.

Example-4: Given an input and an output element group as follows.

Input element group = {e1, e2, e3, e4}

Output element group = {e1, e3, e4, e2}

A relation that the expert prefers "a" to "b" is shown as "a >> b".

In this example, positive ordering relations are "e1 >> e3 >> e4 >> e2", negative ordering relations are "e2 >> e3" and "e2 >> e4".

(d) Ordering Criterion Presumption Algorithm

This algorithm deals with only the output element group as positive examples.

Step1: Making a set of attributes called a total attribute list which are included in positive examples.
Step2: Selecting one attribute from the total attribute list
Step3: Making a value list of the attribute according to the order of elements.
Step4: Checking inconsistent order pairs of two items. An inconsistent order pair means that both "a >> b" and "b >> a" exist.
Step5: If there is no consistent relation, return to step2.
Step6: If elements of the value list are all numbers, then check whether their relation is ascending or descending. Even if they are numbers and the value list has other relations (ex. preference of odd number), the system treats them as literal symbols.

The system shows one ordering criterion . If there are some candidates of criteria related to different attributes, the system shows them in order.

3.5 Attribute Value Replacement Operation Presumption

(a) Attribute Value Replacement Algorithm

Values of a certain attribute of elements are replaced with different values according to a replacement criterion. This criterion is a replacement rule table which contains pairs consisting of a source value and a destination value.

(b) Replacement Criterion of Attribute Values

A replacement criterion of one attribute is defined as follows: (attribuite-name, a set of (source value, destination value))

(c) Treatment of Examples

Examples are elements of an input and an output element group. Concerning an attribute, positive examples have different input and output values, negative examples have the same input and output values.

(d) Replacement Criterion of Attribute Values Presumption Algorithm
Step1: Searching for an attribute which has different values in elements of input and output. This attribute is called a remarkable attribute.
Step2: Making pairs of source values and destination values of the remarkable attribute in positive examples
Step3: Selecting combinations where the source value is the same but destination values are different. These are uncertain replacement rules.
Step4: Picking up values of the remarkable attribute in negative examples.
Step5: Checking whether the pairs created in Step2 have source values included in values made in Step4. If a source value of a pair is included in values made in Step4, the pair is an inconsistent rule.
Step6: Displaying pairs which are not uncertain and inconsistent.
Step7: Displaying pairs which are uncertain and inconsistent for reference.

3.6 Evaluation Criteria Related to Multiple Attributes

We introduced basic presumption algorithms which treat criteria related to one attribute. These criteria are positioned in most general parts of a version spaces[14] shown in Fig.3. This section describes selection criteria related to multiple attributes. A method which we introduce in this section presumes complex criteria while it spreads target parts of the version space to include special concepts step by step. This method also uses the algorithm descried in section 3.2. In order to control the spreading the target part of the version space, this system uses learning heuristics from the expert through interaction. Selection criteria related to multiple attributes is represented in conjunction with some selection criteria of one attribute as follows:

A selection criterion consists of sets of values of an attribute combination;
{ {a value of attribute-1, a value of attribute-2, ...}, ...}

The version space (version graph) of the example in section 3.2 is shown in Fig.3. In this figure, the algorithm of section 3.2 processes (a)-part (P(aa), P(ab), P(ac), $\neg$ N(aa) $\cap \neg$ N(ab) $\cap \neg$ N(ac)) of this figure. CP(aa, ab) is a set which includes an element of a value of aa and a value of ab, for each positive example. CN(aa, ab) is made from negative examples. To treat selection criteria related to multiple attributes, the system must deal with (b)-part in this figure. In order to stimulate the expert to remember necessary attributes, the system calculates similarities between positive and negative examples. This information enables the expert to remember which attributes are important for separating positive and negative examples.

Example-5:

Positive	at1	at2	at3	at4
e1:	1	1	1	0
e2:	1	1	0	1
e3:	1	1	0	0
Negative				
e4:	0	1	0	1
e5:	1	0	0	0

In this example, positive and negative examples cannot be separated by a selection criterion related to one attribute. Therefore, the system calculates distances of pairs between positive and negative examples for all attributes.

distance(e1, e4) = $|v11 - v41| + |v12 - v42| + |v13 - v43| + |v14 - v44|$

If a is equal to b, then $|a - b|$ means 1 else it means 0. Nearest pairs of a positive and a negative example are (e2, e5) and (e3, e5) which distances are 1. The system shows these pairs to the expert to order attributes according to his preference. The expert can eliminates unnecessary attributes. If the expert selects at2, at1 and at4 in this order, the ordering of hypothesis generation and testing is a sequence of $at2 \wedge at1$, $at2 \wedge at4$, $at1 \wedge at4$, $at2 \wedge at1 \wedge at4$. According to conjunctions of attributes, lists of attribute values are made.

The algorithm in section 3.2 processes one attribute as a conjunction of attributes and one value as their value list. For example, in the case related to two attributes, combination information of element $e1$ is shown as follows:

pairs of attributes	(at1, at2)	(at1, at4)	(at2, at4)
pairs of values	(1, 1)	(1, 0)	(1, 0)

A candidate of selection criteria is (at1, at2) = { (1, 1) } (i.e. CP(at1, at2) = { (1, 1) }) in this example.

4 Proposal Guided Knowledge Acquisition Support System

4.1 Knowledge Acquisition by Proposal Generation

The basic frame work of proposal guided knowledge acquisition is to generate hypotheses from given information and to feedback these to the expert to remember new knowledge. Information of this interaction (cf. Fig.1) is shown as follows:

Information from the expert to the system:

(1) Examples (elements of input and output for an operation)
The two types of supply examples which are incremental supply and total supply.

(2) Knowledge modified by the expert
(The expert can modify knowledge as hypothesis proposed by the system.)

Information generated by the knowledge acquisition support system:

(3) Speciality in examples (ex. similarity of examples)

(4) General knowledge generated from examples by induction

(5) Results of evaluation of knowledge modified by the expert
The system evaluates modified knowledge with given input elements and shows output elements of the operation.

Requirements of the expert for the system:

(6) Requirement of next hypothesis

(7) Requirement of evaluation of knowledge modified by the expert

Current implementation of the proposal guided knowledge acquisition support system is EPSILON/One and a sub-module which has an operation presumption function (shown in Fig.4). The target problem of knowledge acquisition is the diagnosis problem because the system handles seven operation types for diagnosis. This sub-module is implemented on PIMOS which is a parallel operating system on the PSI-2 (personal sequential machine) in a parallel logic programming language KL-1[15].

4.2 An Example

We introduce a simple example of the operation presumption about bearing selection, bearing ordering and their value replacement in a machine tool design problem[16]. In this example, the expert selects some bearing types from a total of twenty bearings[17], orders the bearings and replaces values of an attribute (cf. Fig.5).

(1)Presumption of Bearing Selection Knowledge

The expert sets twenty bearing types in an input element group and five bearing types in an output element group. These five bearings are positive examples and the rest of the bearings are negative examples. A bearing type consists of the following:

$[deep-slot-ball-bearing, [force-of-radial, good],$
$[force-of-axial, good], [direction-of-axial, bi-direction],$
$[mixed-force, good], [high-speed, best],$
$[high-accuracy, best], [low-noise, best],$
$[possible-sloop, better], [fixed-case, possible],$
$[free-case, conditional]]$

Examples are given as follows:

Input element group = [twenty types of bearings]
Each bearing has an attribute *high-accuracy* which values are *best, better, and undefined.*

Output element group = $[deep-slot-ball-bearing, angular-ball-bearing, cylinder-roll-bearing, combinatorial-angular-ball-bearing, multiple-cylinder-roll-bearing]$
A presumption result is high-accuracy = $[[best]]$ (shown in Fig.6 in Japanese).

Fig.6 also shows some windows of criteria related to two attributes and three attributes.

(2) Presumption of Bearing Ordering Knowledge

The expert sets an ordering of the five bearings in an output element group. The ordering is as follows:

Output element group = $[[deep-slot-ball-bearing, [high-speed, best], \ldots],$
$[angular-ball-bearing, [high-speed, best], \ldots],$
$[cylinder-roll-bearing, [[high-speed, best], \ldots],$
$[combinatorial-angular-ball-bearing, [[high-speed, better], \ldots],$
$[multiple-cylinder-roll-bearing, [[high-speed, better], \ldots]]$

This ordering is related to the attribute *high-speed*, so the system shows *high-speed* = [best, better] as a presumption result.

(3) Presumption of Bearing Attribute Value Replacement Knowledge

The expert sets an output element group. The input element group of this operation is equal to the output element group of the ordering operation. A value of an attribute of each element in the

output group is different from one in the input element group.

Examples are shown as follows:
Output element group = $[[deep-slot-ball-bearing, [high-speed, range-4], \ldots],$
$[angular-ball-bearing, [high-speed, range-4], \ldots],$
$[cylinder-roll-bearing, [[high-speed, range-4], \ldots],$
$[combinatorial-angular-ball-bearing, [[high-speed, range-2], \ldots],$
$[multiple-cylinder-roll-bearing, [[high-speed, range-2], \ldots]]$
This replacement is related to the attribute *high-speed*, so the system shows
high-speed = [[best → range-4], [better → range-2]] as a presumption result.

5 Related Works

Case based reasoning[18] is a method which searches for a similar example from a case base through analogy and modifies it to suit a target problem. The knowledge acquisition support system prepares a knowledge base which contains general knowledge before reasoning, while the case base reasoning uses the case base directly during reasoning by generalization and analogy.

ID3[19] is an induction algorithm for generating a classification tree. Its classification criterion is an efficient search of the classification tree. Our operation presumption algorithm solves a classification criterion from given leaves of the classification tree, while ID3 generates the classification tree and leaves of the tree from a given classification criterion and items to be classified.

The MIS (Model Inference System)[20] solves a horn clause by induction. It specifies a horn clause with the oracle by asking yes/no questions. Candidates of the horn clause are defined in the limited space of refinement operators. This search space is a kind of version spaces. The MIS prunes unnecessary candidates efficiently through skillful questions. Our operation presumption algorithm deals with more special forms of knowledge and more complex interaction than the MIS.

AQUINAS[21] is a knowledge acquisition support system for classification problems. It extracts classification knowledge in the form of a rating grid. It also makes an implication graph which includes many hypotheses of implication relations among traits. The expert can refine these implications. This refinement style is a kind of proposal guided knowledge acquisition. The induction method of AQUINAS is simpler than our method.

6 Current Problems

In the case that the expert cannot supply learning heuristics, the complexity of current operation presumption algorithm for selection criteria requires a large amount of computing power and is equal to the complexity of a total solution search of a version space. Therefore, it is important to use an elicitation method of learning heuristics. For this problem, the speciality of examples must be illustrated with a cluster tree to represent similarities between examples.

7 Summary

We introduced a knowledge acquisition support system which makes hypotheses by induction and proposes candidates of knowledge. This research suggests a suitable integration of the engineering knowledge acquisition method and logical learning method. An important point is usage of operation types to limit forms of consequence of induction. This system[22] has been implemented in parallel logic programming language GHC (KL1) on a PSI-2 inference machine developed by ICOT. We are planning to apply this system to not only diagnosis problems[23] but also to design problems which are a kind of constraint problem solving problems[24][25].

Acknowledgement

We wish to express our thanks to members and the chief of ICOT 5th Laboratory, Mr. Kenji Ikoma, and members of the PGF and KSA working groups. We would also like to thank Mr. Marc Linster at GMD for his good suggestions. We must express our thanks to the Director of ICOT research Center, Dr. Kazuhiro Fuchi.

References

[1] Mizoguchi, R., Taki, H. and et al., Knowledge Acquisition Support Systems, ICOT TM No. 654 (in Japanese), 1988.
[2] Kobayashi, S., Sawamoto, J., and et al., Knowledge Acquisition Support for Analytical Knowledge Systems, ICOT TM No.581 (in Japanese), 1988.
[3] Kawaguchi, A., Mizoguchi, R., Yamaguchi, T. and O. Kakusho, O., SIS: A Shell for Interview Systems, Proceeding of IJICAI-87, Vol.1, pp359-361, 1987.
[4] Taki, H., A Knowledge Acquisition Support System EPSILON/One, Proceedings of 7th Symposium of Fifth Generation Computer, pp17-18 (in Japanese), 1989.
[5] Taki, H. and Tsubaki, K., Expert Model: A Knowledge Representation for Knowledge Acquisition, Journal of Japanese Society for Artificial Intelligence, Vol.5, No.2 (in Japanese), 1990.
[6] Kelly, G.A., The Psychology of Personal Constructs, New York, Norton, 1955.
[7] Boose, J.H., A Knowledge Acquisition Program for Expert Systems Based on Personal Construct Psychology, J. of Man-Machine Studies, Vol. 23, pp 495-525, 1986.
[8] Taki, H., Tsubaki, K. and Iwashita, Y., Expert Model for Knowledge Acquisition, Proceedings of Proceedings of IEEE Expert Systems in Government Conference, 1987.
[9] Tsubaki, K., Taki, H. and Osaki, H., A Knowledge Acquisition Support System EPSILON/One (a evaluation report), Proceedings of 3rd JSAI Conference, Vol.2, 12-5, pp 719-722 (in Japanese), 1989.
[10] Boose, J. and Gaines, B. (Eds.), A SUMMARY OF THE THIRD EUROPEAN WORKSHOP ON KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS, Knowledge Acquisition for Knowledge-based Systems Newsletter, Vol.1, No.2, Aug., 1989.
[11] Boose, J. and Gaines, B. (Eds.), A SUMMARY OF THE FOURTH KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP, Knowledge Acquisition for Knowledge-based Systems Newsletter, Vol.1, No.3, Nov., 1989.
[12] Genesereth, M. and Nilsson, N., Logical Foundations of Artificial Intelligence, Los Altos, CA: Morgan-Kaufman, 1986.
[13] Taki, H. and Fujii, Y., Operation Presumption: Knowledge Acquisition by Induction, Proceedings of Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems, pp 34-48, 1989.
[14] Mitchell, T.M., Version Spaces: An Approach to Concept Learning, Stanford Technical Report STAN-CS-78-711, HPP-79-2, 1978.
[15] Chikayama, K., and et al., Overview of Parallel Inference Machine Operating System(PIMOS), ICOT TR No. 483, 1989.
[16] Inoue, K., Nagai, Y., Fujii, Y., Imamura, S. and Kojima, T., Analysis of Design Method of Machine Tools,ICOT Technical Memorandum, No.494 (in Japanese), 1988.
[17] Nippon Seiko Corp., Roll Bearing (Catalogue) (in Japanese), 1986.
[18] Kolodner, J.L., Simpson, R.L. and Sycara, K., A process model of case-based reasoning in problem-solving, Proceedings of IJICAI-85, 1985.
[19] Quinlan, J.R., Learning Efficient Classification Procedures and Their Application to Chess End Games, in Machine Learning: An Artificial Intelligence Approach, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds.), Tioga, Palo Alto, Calif., 1983.
[20] Shapiro, E.Y., Inductive Inference of Theories from Facts, Technical Report 192, Yale University Computer Science Dept., 1981.

[21] Boose, J.H. and Bradshaw, J.M., Expertise Transfer and Complex Problem Using AQUINAS as A Knowledge-Acquisition Workbench for Knowlegde-Based Systems, J. of Man-Machine Studies, Vol.26, No.1, pp3-28, 1987.
[22] Taki, H and Terasaki, S., Induction Programs for Operation Presumption in KL1 Language , ICOT Technical Memorandum , 1990.
[23] Nagai, Y., Taki, H., Terasaki, S., Yokoyama, T. and Inoue, K., A Tool Architecture for Design Expert Systems, Journal of Japanese Society for Artificial Intelligence, Vol.4, No.3, pp61-67 (in Japanese), 1989.
[24] Terasaki, S., Nagai, Y., Yokoyama, T., Inoue, K,, Horiuchi, E. and Taki, H., A Machine Design Support Systems Building Tool -MECHANICOT-, Journal of Japanese Society for Artificial Intelligence, SIG-KBS-8803 (in Japanese), 1988.
[25] Mizoguchi, R., Taki, H. and et al., An Image of Knowledge Acquisition Support Systems for Design Problems, ICOT TM No.747 (in Japanese), 1989.
[26] Suwa, M., Sawamoto, J., and et al., Knowledge Acquisition Support for Synthesis Knowledge Systems, ICOT TM No. 588 (in Japanese), 1988
[27] Taki, H., Knowledge Acquisition by Abductive and Inductive Explanation, Proceedings of 4th Knowledge Acquisition for Knowledge-Based Systems Workshop, pp 34-1 - 34-19, 1989.
[28] Linster, M., Report on my Visit to ICOT in August, 1989.

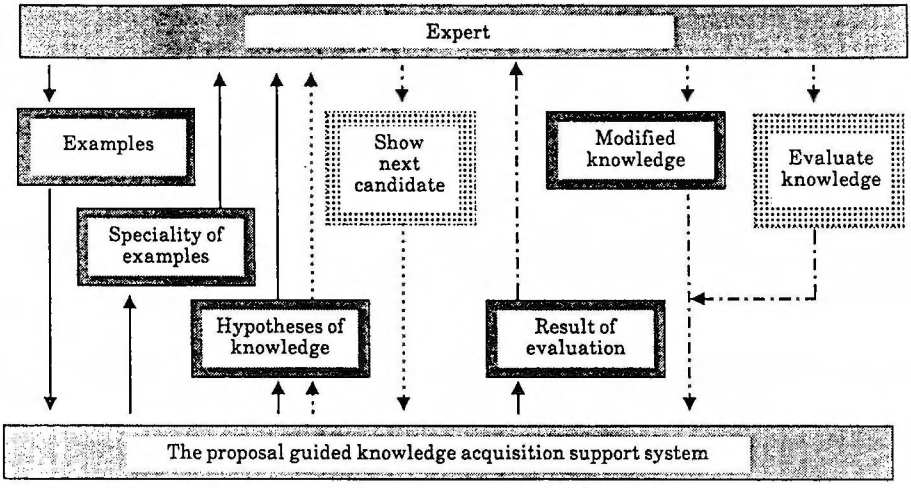


Fig.1 Interaction between the proposal guided knowledge acquisition support system and the expert

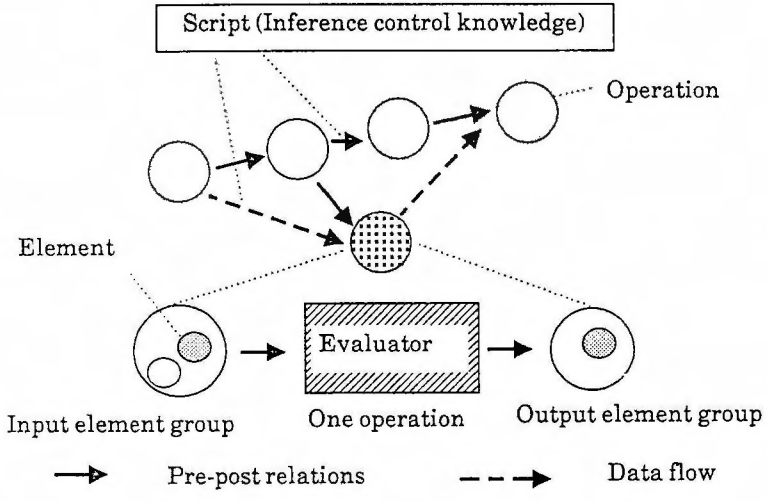


Fig.2 the Expert Model basic structure

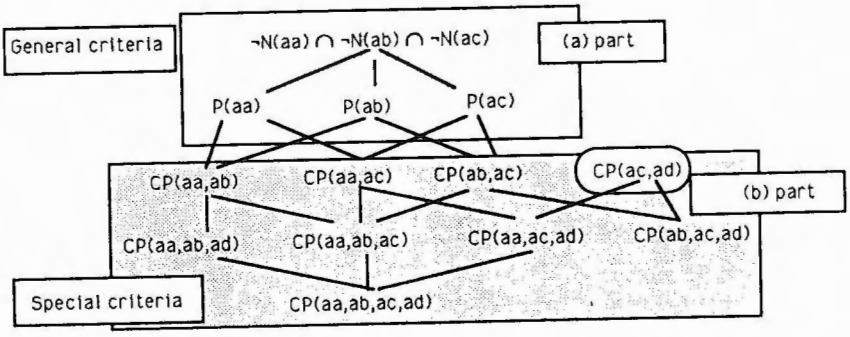


Fig.3 A version space

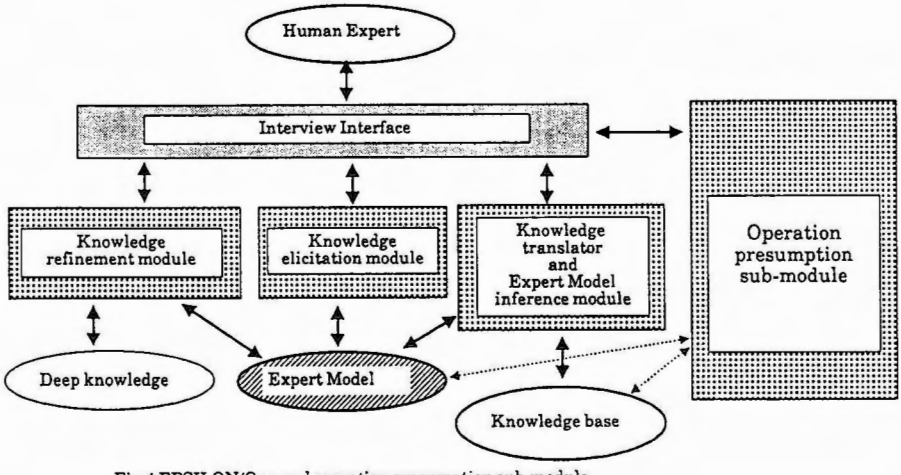


Fig.4 EPSILON/One and operation presumption sub-module

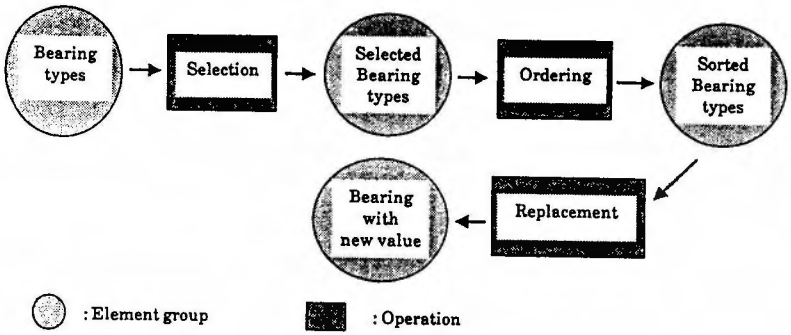


Fig.5 Example

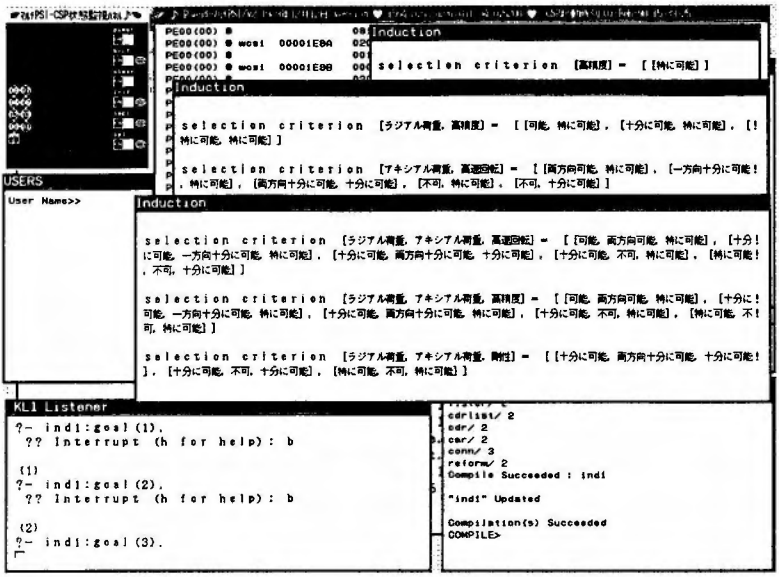


Fig.6 An example of Operation presumption on PSI-2

A Task Analysis Interview System that Uses a Problem-Solving Model

Yuri Adrian Tijerino
Tadahiro Kitahashi
Riichiro Mizoguchi

The Institute of Scientific and Industrial Research,
Osaka University
8-1 Mihogaoka Ibaraki-shi, Osaka-fu 567 Japan

ABSTRACT

This paper presents an interview system which is not restricted to elicit domain knowledge for a single task. This interview system which we call MULTIS, an acronym for MULti-Task-Interview System, configures prefabricated search-control task primitives, called building blocks, to construct the inference engine of an intended knowledge-based system. MULTIS builds a model of expert problem solving (PS) with a network of highly-generalized cognitive primitives, designated as generic tasks, and maps that network to a combination of building blocks. The generic tasks are defined in terms of generic vocabularies, i.e. task-dependent lexica independent of the domain, and their definitions used throughout the interview not only for coherency, but also for identifying generic tasks being used by the domain expert.

Since generic tasks might be combined in different manners to model PS for experts that perform a similar task, the possible combination of the generic tasks is exponentially proportional to the number of generic tasks. Therefore, the problem of identifying the right mix of building blocks becomes NP hard. Consequently, MULTIS is being implemented as a case-based reasoner to avoid inefficiently searching for all possible generic task combinations during every interview session by modifying previously successful ones.

Key words: Interview system, generic vocabulary, generic task, building block, task analysis, case-based reasoning.

1. Introduction

Knowledge engineers have been responsible for some time now of the difficult process of transferring expertise from domain experts to knowledge-based systems (KBS). Therefore, it is to be expected that knowledge engineers have become experts of expertise transfer. Based on this notion, interview systems have been constructed to simulate knowledge engineer behavior. However, current interview systems are said to be task-dependent in as much as an interview system can only help acquire knowledge for a predetermined task for which the search-control structure is already in existence, i.e. the inference engine of the KBS is known in advance.

Knowledge engineers engage in PS by analyzing a domain expert's PS behavior and designing a system (i.e., knowledge-based system or expert system), that simulates that behavior. Consequently, a knowledge engineer performs both analysis and synthesis tasks when engaging in PS. Analysis is performed when trying to describe what type of PS (task) the domain expert executes and why it is executed in that particular manner. Then, synthesis comes into play by the combination of useful component pieces that come to shape a system that can engage in the same task in a similar fashion. In this paper, the authors propose both a model in which primitives are employed to analyze tasks and a framework in which task components can be synthesized to devise a search-control structure that simulates that of a particular task. Both, model and framework, are implemented in MULTIS which is a task-independent interview system that attempts to construct the search-control structure of the intended knowledge-based system before endeavoring to acquire domain knowledge.

MULTIS's approach differs from previous interview systems in that it employs both bottom-up and top-down interview to avoid task-dependence. MULTIS attempts to construct the search-control structure of the intended knowledge-based system before endeavoring to acquire domain knowledge. Other interview systems simply perform acquisition of domain knowledge needed by existing inference engines, thus the kind of tasks they can be applied to are highly dependent on those inference engines.

2. Generic Vocabulary and Generic Tasks in MULTIS

MULTIS assumes that there exist task-dependent terminology for each PS task that can be represented in the form of generic vocabulary, i.e. task-dependent lexica independent of the domain. Generic vocabulary can serve two purposes, that is, to analyze the tasks for hidden cognitive primitives, which we call generic tasks, and to help the interviewer be fluent and coherent during interview.

In the real world, generic vocabulary can be found in most fields of human cognitive processes. Let's look at word-processing as an example. In most, if not all, word-processing programs one can use the words mark, delete, cut, copy, paste, paragraph and others. Nevertheless, the name of this actions (verbs) may vary from implementation to implementation, not to mention from language to language. No matter how different one may choose to call these words, the action intended is still very similar, if not the same. Due to this fact, when a person becomes well acquainted with one word processor it becomes very simple for him/her to use any other word processor once the necessary mapping or translation of the previous word-processor's vocabulary has been made to the new word-processor's vocabulary. The same is true for generic tasks, which are primitives defined in terms of generic vocabulary. In our word-processing example, a generic task could be paragaph-deletion for which a paragraph has to be marked before being deleted. The generic task to generic vocabulary relation can be written as: paragraph-deletion = delete marked paragraph. The most important relation between generic vocabulary and generic tasks is that generic tasks usually combine two or more generic words to describe an action.

We have identified some generic tasks for planning and scheduling, but intend to expand to other PS tasks such as monitoring and diagnosis. Our generic tasks differ from Chandrasekaran's [1] in that our generic tasks have to be combined in order to obtain knowledge-based system building blocks. In other words, our generic task are used to model human PS at the cognitive level. Chandrasekaran's generic tasks can be used as building blocks as they are. Hence, they can be readily used to construct the inference engine of a KBS.

The concept of building block is a little harder to grasp than generic vocabulary or generic task because building blocks are transparent to the interviewee. Though MULTIS builds a model of PS interactively with the interviewee using generic vocabulary and generic tasks, a mapping of such model is made transparently to the building blocks needed to construct the inference engine of the knowledge-based system intended. This approach is consistent with the knowledge engineer's approach because they code the inference engine without showing it to the interviewee after building their PS model interactively. Figure, 2.1 illustrates MULTIS's PS model to inference engine mapping.

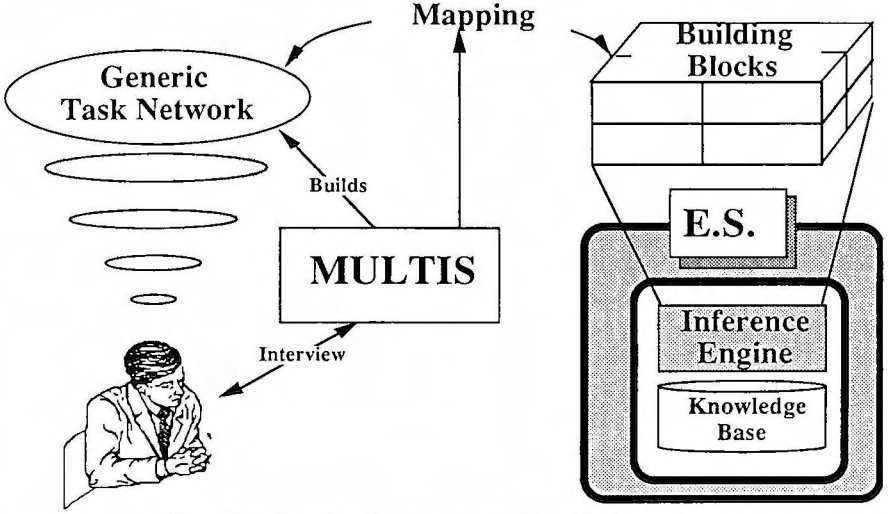


Figure 2.1. Generic task network to building block mapping.

Table 2.1 exhibits some generic tasks defined in terms of a lexicon for scheduling. Although, perfect generic vocabulary and generic task libraries cannot be constructed, we believe that they provide enough guidance for analysis and synthesis of the tasks under consideration. What matters the most, is that the building blocks derived from the generic tasks are able to successfully accomplish their final goal, that is, that by identifying the correct mix of building blocks and acquiring their required knowledge from the expert, they will help construct (synthesize) the desired KBS.

We have divided generic tasks into three types: generative, testing and modifying tasks which can interact with each other during PS. In the case of scheduling, an example of a generative generic task is goal-modularization which generates subgoals for a given goal. Similarly, constraint-conflict-checking is a testing task and solution-refinement is a modifying task. A description of the generic tasks given in table 2.1 is presented in Appendix A.

Generic tasks present structures that are better employed to explain human PS at the cognitive level than at the implementation level. Therefore, the need arises to design other structures, i.e. building blocks, that can be used to fabricate the inference engine of the intended knowledge-based system. In MULTIS, a mapping between generic tasks and building blocks is made in such a way that one or more generic tasks may correspond to a single building block. Building blocks present very simple control structures and, as the generic tasks, can be classified into three kinds: generator, tester and modifier. Hamanaka and Mizoguchi [2] argue that structures of these building blocks depend very much on the kind of solution expected and its properties. For instance, a generator for an analysis task might consist of a control structure that chooses a solution from a set of predetermined choices, while a generator for synthesis tasks generates a solution from component pieces of subsolutions that might have been generated by either kind of generator.

Table 2.1. Generic tasks defined in terms of generic vocabulary for scheduling.

Generic Tasks	Definitions in Generic Vocabulary
alternative-solution-generation	generate alternative solution that satisfies constraints
constraint-conflict-checking	identify constraints not satisfied because of conflict
constraint-identification-and-categorization	identify and assign constraints to categories
constraint-prioritation	assign priorities to constraints
constraint-relaxation	relax tolerance bounds of flexible constraints
constraint-tolerance-setting	assign tolerance to flexible constraints
constraint-updating	identify new or delete constraints
goal-modularization	generate a subsolution for each job
job-batching	generate batches for jobs with similar processes
job-identification-and-prioritation	identify jobs to be scheduled and assign them priorities
job-updating	identify new or delete jobs
preference-allocation	assign preference to job, batch, resource or constraint
process-mapping	identify processes needed by job or batch
process-sequencing	generate a sequence of processes that satisfies job or batch constraints
resource-capacity-analysis-and-assignment	analyze resource capacity to satisfy a job or batch need
resource-updating	identify new or delete resources
solution-modification	modify a solution to satisfy new constraints
solution-refinement	apply heuristics to modify schedule and satisfy all constraints
subsolution-merging	generate a solution by merging all subsolution satisfying all constraints

3. MULTIS's Interview Approach

MULTIS's interview strategy consists of the following four phases:

1. Task identification. Identify the PS task the expert performs.
2. Task analysis and synthesis. Analyze the task and identify needed search control knowledge.
3. Knowledge acquisition. Elicit domain knowledge and task implementation knowledge.
4. System evaluation. Evaluate the system by imputing an already solved problem.

These steps are further described in the following sections.

3.1. Task Identification

MULTIS first identifies the task the domain expert, from whom the knowledge is to be elicited, performs during PS. During this phase of the interview, the tasks for which MULTIS can elicit knowledge are currently planning and scheduling, but can be expanded later to cover other tasks such as diagnosis, monitoring and so on. Since a taxonomy of tasks already exists for the type of tasks being performed by humans [3], this phase can be implemented as a classification problem. Therefore, the interview is based on heuristics about the taxonomy of tasks. Roget [4] is an earlier interview system that attempts to identify the type of task the interviewee performs before engaging in knowledge acquisition. However, it limits the type of task it can elicit knowledge to diagnosis by using sentence-parsing techniques that could prove somewhat inefficient if other tasks were to be added. MULTIS performs interactive interview with a user friendly interface to avoid falling into a similar problem (an example is given in Appendix B).

Since an expert sometimes performs more than a single task in his/her field of expertise, it is necessary to identify what is the main task being performed. Once that task is identified, the rest of the tasks being performed can be ignored, because later during the interview their corresponding generic tasks will be incrementally added to the model of the main task. This is a problem that hasn't been treated properly in the past because most interview systems have assumed that human experts perform only a single task. However, human PS is usually more complex, that is, it consists of a mixture of analysis and synthesis at different levels. With this in mind, MULTIS identifies the main task the expert performs and then incrementally identifies what other tasks are bieng performed.

MULTIS then extracts the generic vocabulary corresponding to the task identified and translates it to the interviewee's vocabulary. As the term generic implies, a generic vocabulary is generalization of a vocabulary for a given task. Therefore, a vocabulary must be translated to reflect the interviewee's vocabulary. In MULTIS the vocabulary is translated interactively by the user himself. An interactive translation example is depicted in figure 3.1. In this figure there are three examples chosen randomly from previous interview sessions and posted in the three right columns. The untranslated vocabulary is put in the left-most column. The second column is left blank so that the interviewee can put his translated vocabulary. The interviewee can explore, with use of a pointing device, any of the previous translations to see how they were translated or the untranslated vocabulary for their definitions. The interviewee then writes the translation to his/her vocabulary on the second column or he/she can paste it directly from any of the examples or untranslated vocabulary when they correspond to the same vocabulary. The interviewee can leave out the translation of any vocabulary when he/she thinks they are irrelevant. Figure 3.2. depicts the flow of task identification in MULTIS.

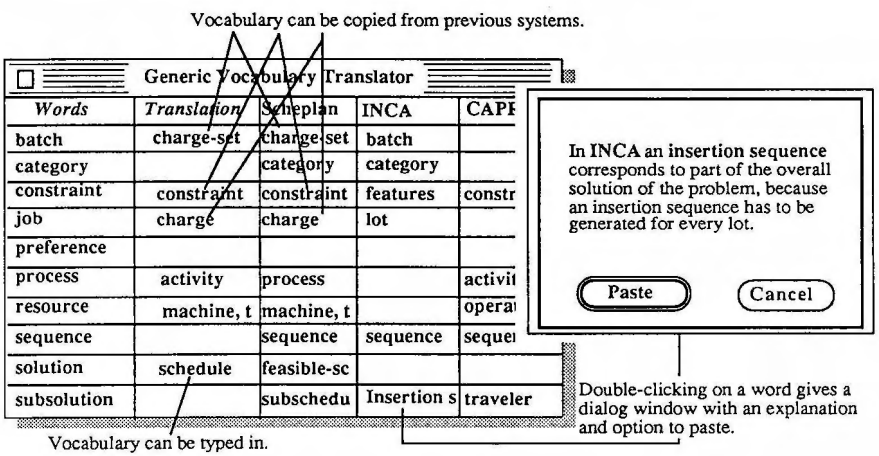


Figure 3.1. The generic vocabulary translator.

- Lack of standardization in the knowledge representation resulting from the techniques and tools making it difficult to integrate them and to interface them to existing knowledge based systems;
- Lack of standardization in the forms of data required by the techniques and tools making it difficult to apply them in the same situation and compare them;
- Lack of standardization in the user interfaces to the interactive tools making it difficult to integrate them in an effective environment for human-computer interaction;
- Lack of portability in the run-time environments required by the tools making it difficult to integrate them with other systems.

These are expected problems at this stage of development of knowledge based systems. We are in the midst not only of the evolution of a rapidly developing technology but also in a revolution of our understanding of the nature of knowledge and knowledge processes in society and technology. Premature standardization resulting in over-rigid systems would be inappropriate and a major impediment to progress. However, we have to balance the need for integration, dissemination and application of knowledge acquisition techniques and tools against the dangers of rigidity. Certainly we may expect the major thrust of research on knowledge acquisition for knowledge-based systems during the 1990's to be concerned with overcoming these impediments.

The workshops are playing a major role in this, and the involvement of technological communities world-wide is to be welcomed. The First Japanese Workshop on Knowledge Acquisition for Knowledge-Based Systems provides state of the art surveys of knowledge acquisition methodologies, techniques and tools, and research papers illustrating current developments. This paper provides a background to this material by surveying the foundations of knowledge engineering.

2 KNOWLEDGE-BASED SYSTEMS

One problem in understanding the processes of knowledge engineering is that the distinction between knowledge acquisition and performance systems is an invidious one. It can be taken to imply that expertise is a static collection of skills and knowledge, capable of being elicited and transferred to an expert system that will then be able to reproduce the expertise. This static model is inappropriate to systems design, leading to systems that date rapidly, interfere with the growth of expertise, have poor reliability and high maintenance costs. The following explains how these conditions arise, and leads to approaches for avoiding them.

Hawkins [26] has abstracted from industrial experience in developing mineral exploration expert systems and proposed a model of human expertise relevant to expert systems. Figure 1 presents a summary of the essential features of this model:

- The expert first elicits data about the problem from the client;
- He or she develops a minimal model that accounts for the data provided;
- He or she generates advice based on the model and feeds this back to the client;
- The client may accept the advice, or query it and, possibly, the model;
- The queries lead to further data elicitation, and repeat of the elicitation / modeling / advice / query cycle.

Thus, in Hawkins' model, the client plays an active role in further developing the model by providing more data until he or she is satisfied with the model and consequent advice. Expert advice giving and taking is part of a cycle of negotiation around a process of model formation as shown in Figure 1.

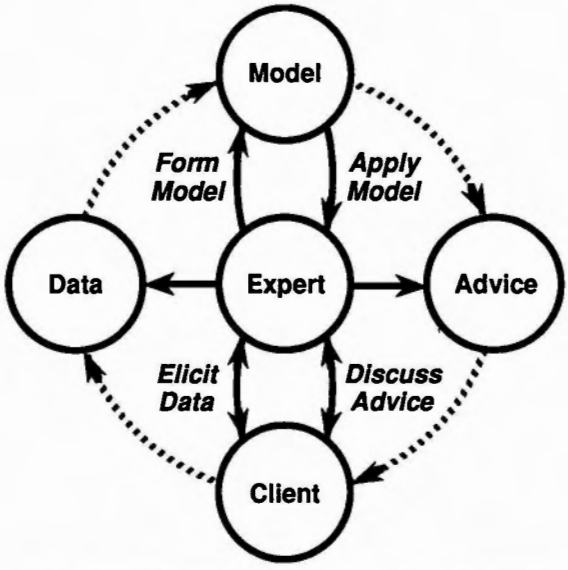


Fig.1 The negotiation cycle in expert-client interaction

The particular importance of this model of expert performance is the emphasis it places on the embedded knowledge acquisition process. The expert is managing a data collection, modeling and decision cycle within a consultation, and is able to enhance his or her collection of case histories, models and advisory strategies through the consultation. Skilled performance is not the static utilization of expert knowledge and skills in Hawkins' model—it is instead a microcosm, and an essential part of, the wider processes of knowledge acquisition which are intrinsic to the role of an expert.

Access to the learning experiences of attempting to solve the problems of a client community is one major source of knowledge in the development of the expertise of an individual expert. The client community provides access to practical experience, a variety of problems, particularly novel ones that go beyond existing expertise, and it also manages the growth of expertise through systems of reward, criticism and access to resources. However, learning from experience is a slow and error-prone process, and socially significant areas of expertise become associated with professional communities that attempt to expedite learning and reduce errors through the sharing of experience. Figure 2 extends the model of Figure 1 to include these other processes, such as apprenticeship, instruction, training, education, workshops, conferences, journals and books. Professional communities usually also play a major role in refining and directing the client community's reward, criticism and resource allocation systems.

The role of the professional community can be seen as an indirect enhancement of the basic processes of knowledge acquisition in the negotiation cycle of expert performance. The individual can acquire the data of others through access to reported case histories. He or she can enhance his or her repertoire of models through the published theories developed by others to account for their data and the data of many others. He or she can enhance his or her repertoire of advice through the published advisory strategies developed by others and evaluated by them, or yet others, against past case histories, theoretical foundations, and experience in use.

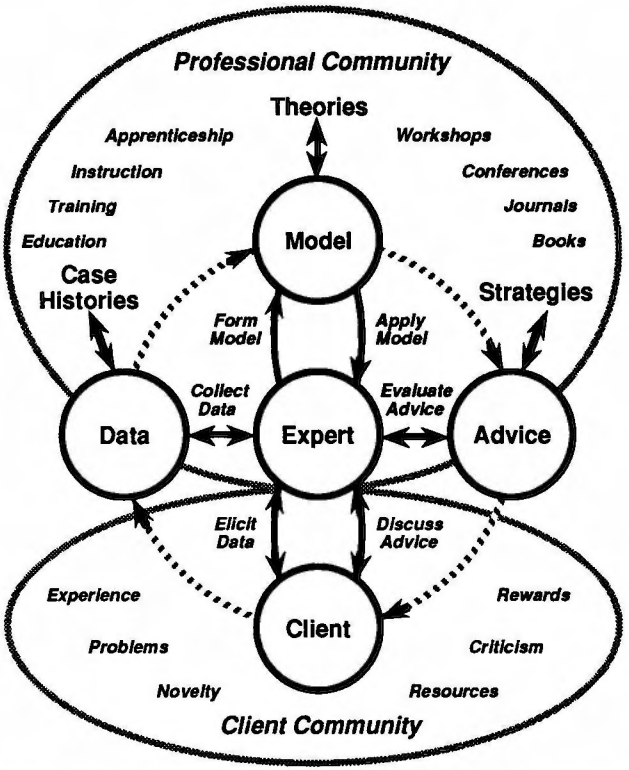


Fig.2 Extended knowledge acquisition process in expert-community interaction

This model of individual expertise as intrinsically a knowledge acquisition process at multiple levels, embedded in and supported by essential social processes, is consistent with Stich and Nesbitt's [42] view of the social role of experts as managers of the *reflective equilibrium* of the inductive process of knowledge generation in society. It is also consistent with the emphasis in professional development books on the importance of balancing experts' authoritarian roles as knowledge repositories and appliers with their entrepreneurial roles as knowledge acquirers [35].

Sociologists have noted the very strong positive feedback processes in the dynamics of knowledge acquisition in the scientific community [25]. Merton [32] coined the term the "Matthew effect" for those features of the reward system in research that were biased towards allocating greater credit for the same discovery to those with an already established reputation. The qualitative effect of such positive feedback processes is to amplify random differences within a population, creating strong distinctions which have no other basis than the feedback process itself. That is, even if all scientists were created equal in their capabilities, some would become experts relative to others because random differences in early performance generated credit that affected resource allocation and gave them greater opportunities to acquire knowledge. The strength of this effect can be demonstrated by studies of the behavior of 'experts' simulated by learning automata in a competitive learning environment with access to resources related to demonstrated 'expertise' [18].

In many areas of expertise where expert systems are expected to play a major role these positive feedback processes are exceptionally strong. In medicine, for example, the key learning resource is access to medical problems, and the 'owner' of such a problem has a strong personal interest in only allowing someone of very good reputation to handle it. The system, including considerations of legal liability, strongly funnels problems to those who are regarded as experts. It is, however, precisely these problems from which new knowledge is generated. Hence direct experience of case history data is made available primarily to those with already established reputations for having capabilities to deal with it. Similar considerations apply to the award of scholarships, invitations to scientific congresses, and so on [2]. They also apply not only to individuals but also to social units, such as a company subject to government procurement procedures that are heavily biassed to contractors with prior experience and with whom the government agency has prior experience.

The introduction of knowledge-based systems in the processes shown in Figure 2 may be seen as providing technological support to the 'expert' role in the center of the figure. The system developer operates in the same environment and Figure 2 may be restructured to show the sources available to the developer of a knowledge-based system. Figure 3 illustrates the knowledge acquisition process with the system developer drawing upon three major sources: knowledge encoded in *relevant media*, such as books, journals and videotapes; knowledge available by observing and modeling the *relevant domain*; and knowledge available from discourse with, and observation of, the *relevant community*. It is this last source that is most associated with expert systems as such, although all three sources play significant roles in the development of any practical system.

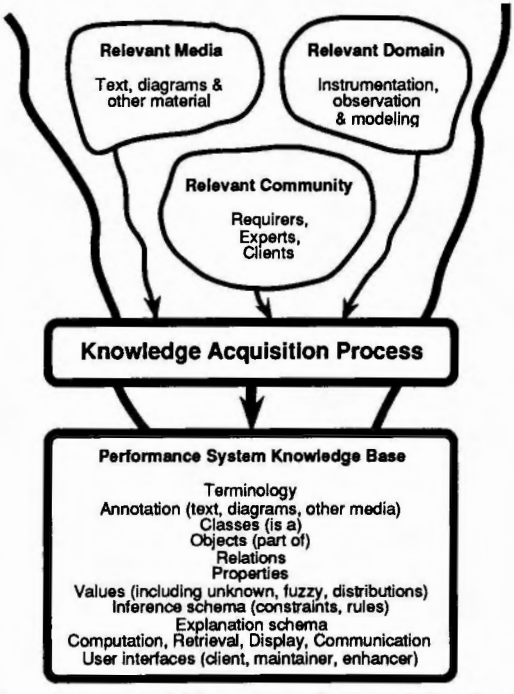


Fig.3 The knowledge acquisition process for knowledge-based systems

Note that the *relevant community* has been split into requirers, experts and clients. Again, domain 'experts' are the expected source of knowledge for an expert system, but detailed requirements specifications and involvement of end-users play as significant a role as they do in other computing system design. It is useful to think of those with these different roles as being all 'experts' in some different aspect of the system functioning, since the knowledge acquisition techniques used with domain experts are often also well-suited to establishing requirements specifications and end-user perspectives.

The annotation of the *knowledge base* in Figure 3 illustrates the range of technical features which an adequate knowledge-based system shell may be expected to support:

- *Terminology* is often neglected as computationally significant. However, it is particularly important in knowledge-based systems where it is critically important that the terms used in questions, recommendations and explanations are understood by clients in the same way as they are by experts. Obtaining the terminology direct from experts without distortion is an important design consideration for knowledge acquisition systems.
- *Annotation* is used here as a term for non-computational knowledge such as text, diagrams, photographs, videos, and other media, that may be used to impart knowledge to people. Some of it may be suitable for computer storage but not for inferential processing. Providing appropriate access to such knowledge is important in many knowledge-based systems, and collecting and organizing such knowledge is part of the knowledge acquisition process.
- *Classes, objects, relations, properties* and *values* form the abstract infrastructure of computational knowledge representation. Object-oriented knowledge-bases provide a formal conceptual and computational schema which subsumes and supports a wide variety of knowledge representation schema. These primitives also correspond to natural cognitive operations of classifying, instantiating, relating, attributing and measuring.

Where knowledge representation goes beyond conventional data representation is largely in the variety of types of value that must be supported. In a data base, a numeric field may be expected to have a clearly defined value. In a knowledge base, because knowledge is 'less than truth', it may be necessary to allow for the value being unknown, constrained by a fuzzy hedge, a range, a distribution, or some combination of these. There may also be mutual constraints between such ill-defined values.

- *Inference schema* are systems of constraints that express the relations between values in a knowledge base. These constraints may be thought of as rules expressing the relations and many of them may necessarily be expressed by explicit rules in the knowledge-base. However, the effective organization of the knowledge base in terms of generic classes and relationships should allow the majority of the constraints to be expressed implicitly in a natural way. This is the *framing problem* in knowledge representation for a domain, and an important aspect of knowledge acquisition is to determine a representation schema that minimizes the number of low level rules required.
- *Explanation schema* are forms of argument that generate satisfactory explanations of the inferences made from a knowledge base. Early expert systems generated explanations by displaying the chain of reasoning that was used from premises to conclusions. This approach is useful but does not necessarily 'explain' the conclusions in a way that an expert would, or in a way satisfying a client's need for knowledge. The acquisition of knowledge necessary for adequate explanation goes beyond that adequate for performance.
- *Computation, retrieval, display, communication* and other data processing activities play major roles in most knowledge-based systems. Modern expert system shells make provision for integration with data-processing, simulation, database, graphics and communications programs. The systems analysis for these needs to be integrated with the knowledge acquisition process.

- *User interfaces* are very important in knowledge-based systems as they are generally highly interactive and involve clients with little or no computer experience. The design of the interface in consultation with experts and clients is an important part of knowledge acquisition. Since most knowledge is fairly volatile, knowledge acquisition and updating the knowledge base are important aspects of system operation, and the interface to system maintainers and enhancers also needs careful attention.

The emphasis on the complexity of the social processes underlying knowledge creation, dissemination and application in society, and the need to treat knowledge-based systems as an integral part of information technology, makes the development of such systems appear somewhat daunting. And so it is—it is easy to underestimate the effort required for the development of effective 'expert systems.' However, knowledge acquisition for knowledge-based systems has also tracked other trends in information technology, notably the use of the computer itself to support the system designer through computer-aided software engineering (CASE) [19]. The following section describes these trends and their effects on the role of the 'knowledge engineer.'

3 KNOWLEDGE ENGINEERING

Knowledge acquisition for practical system development has come to be termed *knowledge engineering*, following Feigenbaum's [14] use of the term to describe the reduction of a large body of knowledge to a precise set of facts and rules. The term *knowledge engineer* has come to be used for the person responsible for such system development, and concise job descriptions for knowledge engineers have been given:

> "Knowledge acquisition is a bottleneck in the construction of expert systems. The knowledge engineer's job is to act as a go-between to help an expert build a system. Since the knowledge engineer has far less knowledge of the domain than the expert, however, communication problems impede the process of transferring expertise into a program. The vocabulary initially used by the expert to talk about the domain with a novice is often inadequate for problem-solving; thus the knowledge engineer and expert must work together to extend and refine it. One of the most difficult aspects of the knowledge engineer's task is helping the expert to structure the domain knowledge, to identify and formalize the domain concepts." [27]

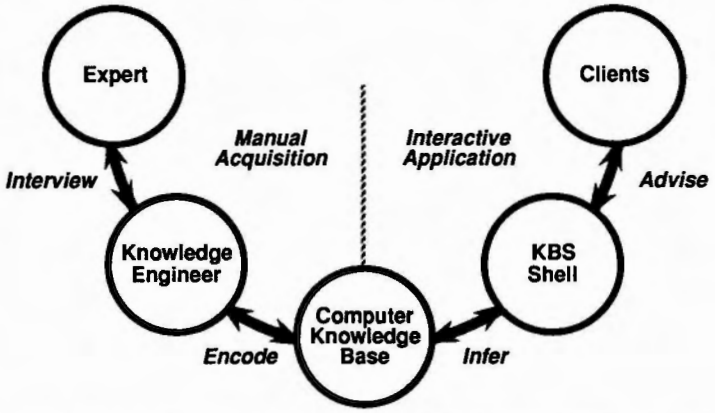


Fig.4 Manual knowledge acquisition by a knowledge engineer interviewing an expert

Thus, the basic model for knowledge engineering has been that the knowledge engineer mediates between the expert and knowledge base, eliciting knowledge from the expert, encoding it for the knowledge base, and refining it in collaboration with the expert to achieve acceptable performance. Figure 4 shows this basic model with manual acquisition of knowledge from an expert followed by interactive application of the knowledge with multiple clients through an expert system shell:

- The knowledge engineer interviews the expert to elicit his or her knowledge;
- The knowledge engineer encodes the elicited knowledge for the knowledge base;
- The shell uses the knowledge base to make inferences about a particular case;
- Clients use the shell to obtain advice about particular cases.

It seemed possible that knowledge engineering might develop as a profession on a par with systems analysis and programing, and that a major shortage of skilled knowledge engineers would cause problems initially that would be overcome eventually as the profession developed. However, this scenario now appears less and less likely. There is certainly a shortage of knowledge engineers, and major problems in developing applications, but doubts have been cast on the notion that human labor is the appropriate solution to the knowledge engineering problem:

- The decline in costs of both hardware and software support for expert systems has brought the technology into a mass-market situation far more rapidly than originally envisioned;
- This has lead to a growth in demand for expert systems that is proceeding far more rapidly than the growth in supply of trained and experienced knowledge engineers;
- The declining costs of expert system technology are also making the expense of human labor in tailoring the technology for particular applications appear to be the dominating constraint and an excessive cost;
- A move towards a labor-intensive activity such as knowledge engineering is contrary to all trends in industry;
- In particular, it is contrary to the trend towards automatic programing techniques in the computing industry;
- The role of the knowledge engineer as an intermediary between the expert and the technology is being questioned not only on cost grounds but also in relation to its effectiveness—knowledge may be lost through the intermediary and the expert's lack of knowledge of the technology may be less of a detriment than the knowledge engineer's lack of domain knowledge.

These considerations aroused interest in the possibility of providing computer-based knowledge elicitation systems to automate knowledge engineering as a process of direct interaction between domain experts and the computer [7, 15]. The tools and techniques used in such systems are described in later sections. Figure 5 shows how the role of the knowledge engineer has changed as such systems have come into use.

Interactive knowledge acquisition and encoding tools can greatly reduce the need for the knowledge engineer to act as an intermediary, but, in most applications, they leave a substantial role for the knowledge engineer. As shown in Figure 5, knowledge engineers have responsibility for:

- Advising the experts on the process of interactive knowledge elicitation;
- Managing the interactive knowledge acquisition tools, setting them up appropriately;
- Editing the unencoded knowledge base in collaboration with the experts;
- Managing the knowledge encoding tools, setting them up appropriately;
- Editing the encoded knowledge base in collaboration with the experts;

- Validating the application of the knowledge base in collaboration with the experts;
- Setting up the user interface in collaboration with the experts and clients;
- Training the clients in the effective use of the knowledge base in collaboration with the expert by developing operational and training procedures.

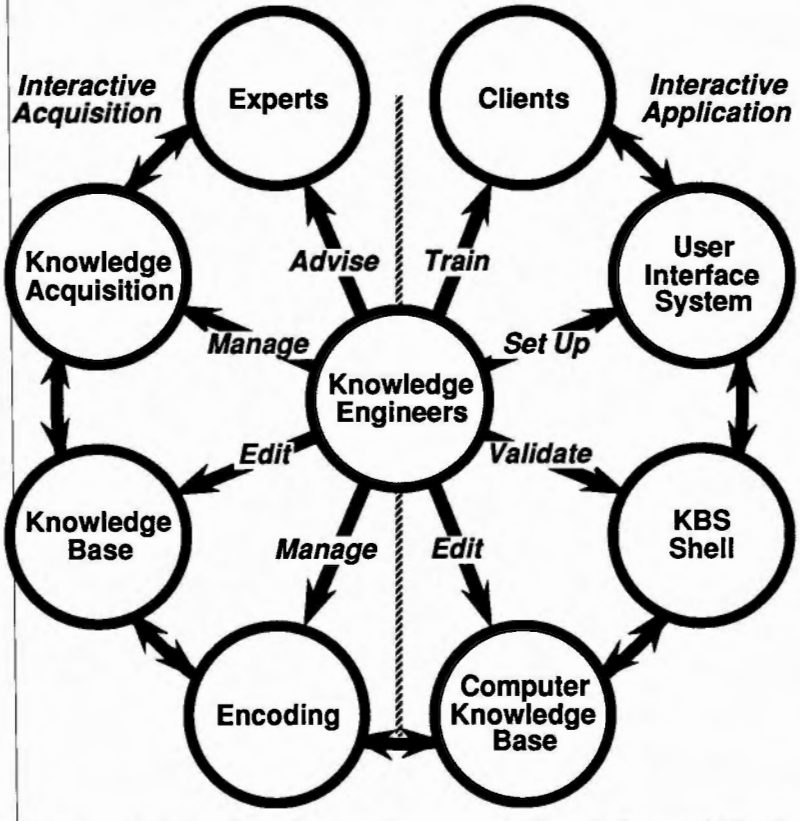


Fig.5 The knowledge engineers' roles using automated knowledge acquisition tools

This use of interactive elicitation can be combined with manual elicitation and with the use of the interactive tools by knowledge engineers rather than, or in addition to, experts. Knowledge engineers can:

- Directly elicit knowledge from the expert;
- Use the interactive elicitation tools to enter knowledge into the knowledge base.

One of the most important developments in knowledge engineering in recent years has been the development of general domain and task ontologies that provide widely applicable generic models for knowledge engineering [10, 31]. These are the basis of general knowledge-based systems analysis techniques such as those of KADS [9]. The applicability of such generic models reflects the perspective presented by Clancey [11] that *all* knowledge-based systems are essentially models at some level.

4 THE KNOWLEDGE-BASED SYSTEM VIRTUAL MACHINE HIERARCHY

The quality of the knowledge-based system support environment is very important in the development of expert systems, particularly in supporting the interaction between knowledge engineers and experts. The relation of this environment to knowledge acquisition is best seen by putting both in the context of the complete virtual machine hierarchy for knowledge-based systems. Figure 6 shows significant levels of this hierarchy:

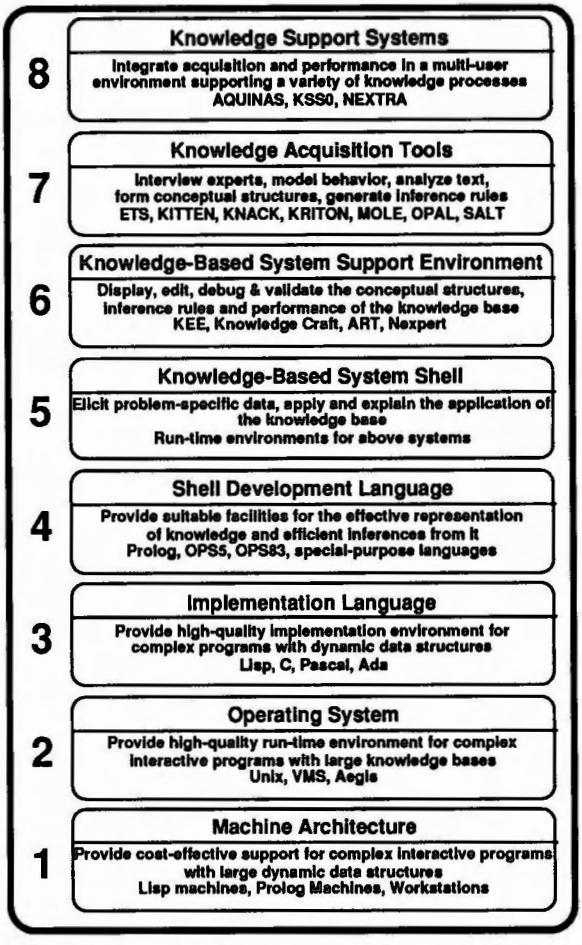


Fig.6 The virtual machine hierarchy for knowledge-based systems

- *Machine Architecture*: at the lowest level of the hierarchy is the machine on which the knowledge-based system runs. This is really irrelevant, but use of Lisp in the early days of expert systems has focused attention on Lisp machines. Nowadays any good workstation is adequate.

- *Operating System*: at the second level of the hierarchy is the operating system within which the implementation runs. This needs to provide good interfaces to other programs, large databases and communications.
- *Implementation Language*: at the third level of the hierarchy is the implementation language which actually interfaces to the computer. This tended to be Lisp in the early days of expert systems but as speed and space efficiency have become significant, and knowledge representation has become better understood, other languages that support dynamic data structures such as C and Pascal have become widely used.
- *Knowledge-Based System Shell Development Language*: at the fourth level of the hierarchy is the language in which the knowledge-based system shell is written, generally a special-purpose environment for coping with knowledge representation and inference.
- *Knowledge-Based System Shell*: at the fifth level of the hierarchy is the expert system shell as a run-time environment that elicits problem-specific information from the user, provides advice based on its knowledge base, and explains that advice in as much detail as required.
- *Knowledge-Based System Support Environment*: at the sixth level of the hierarchy is the equivalent of the Application Programing Support Environment (APSE) in conventional systems. The Knowledge-Based System Support Environment provides editing, browsing and debugging tools for the knowledge base. This level encompasses the major differences between the high-quality knowledge-based system shells that support major system development through extensive support tools, and the simpler shells that do not.
- *Knowledge Acquisition Tools*: at the seventh level of the hierarchy are the tools for automating the systems analysis for expert systems, through automatic interview procedures, modeling actual expert behavior, and analysis of knowledge in textual form. Automation of systems analysis is more significant for knowledge-based systems than for conventional data processing because they tend to be targeted on narrow areas of in-depth specialist knowledge, rather than broad general problems. In conventional terms, knowledge-based systems are satisfying the requirement for highly customized data processing based on idiosyncratic operations rather than general algorithms.
- *Knowledge Support Systems*: at the top of the hierarchy represent the objectives of recent research in knowledge acquisition that focuses on the integration of acquisition and performance systems to provide an environment supporting a wide variety of knowledge processes.

The hierarchy of Figure 6 highlights some of the problem that system managers often face in establishing development capabilities for knowledge-based—should they use Lisp machines?—should they train their staff in Lisp or Prolog?—how does an expert system shell compare with existing programing languages?—what is knowledge engineering and how does it differ from systems analysis? The problem is that knowledge-based systems are at the top of the virtual machine hierarchy which has developed in computing, and there are many layers of technology between a knowledge-based system and the computer on which it is running. In theory, system developers should not need to know about the lower levels of the hierarchy—machine architectures, operating systems and implementation languages are remote from knowledge processing. However, in practice, these lower levels are the foundations on which systems are built and any defects in them can undermine the functionality of the upper levels.

The virtual machine hierarchy for computing is very complex with many branches. Figure 6 focuses on a path through the hierarchy that links the basic hardware and software to knowledge-based systems and knowledge acquisition issues. It omits other parts of the hierarchy that illustrate the integration issues discussed previously. Figure 7 shows some of these issues within a hierarchical framework [21].

The split between users and technology at the top of Figure 7 emphasizes the importance of effective person-computer interaction in knowledge-based systems. As with much modern information technology, usability considerations dominate effectiveness—if the interface is poor no amount of advanced technology in the rest of the system can compensate.

The tree below this to the top left depicts the distinctions between user roles and, in particular, the tasks of the knowledge engineer already discussed.

The tree to the top right first splits the technology between knowledge-based and conventional components, depicting some of the integration issues already discussed, and then further splits the knowledge-based components into support environment, shell, and acquisition as already discussed.

The remainder of this paper focuses first on the sub-tree to the bottom left, analyzing the knowledge-based system support environment and validation, and then on that to the bottom right, analyzing and exemplifying the methodologies, tools and techniques used in knowledge acquisition.

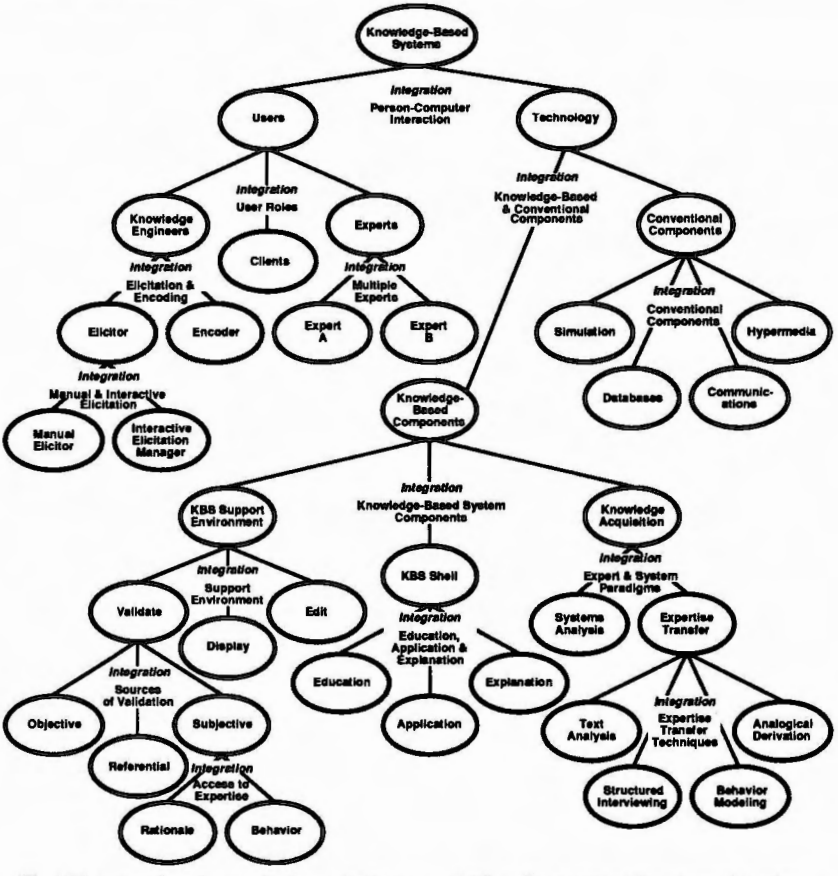


Fig.7 Integration issues in knowledge acquisition for knowledge-based systems

5 EDITING AND VALIDATION

The sub-tree on the lower left of Figure 7 shows some of the major issues in the knowledge-based system support environment at level 6 in Figure 6 in terms of the distinctions involved. The basic tools required are those for editing, displaying and validating the knowledge base. It is important that these tools present a uniform and consistent view of the knowledge base and that they aid consistency maintenance in the knowledge base during editing.

Editing is often regarded as a mundane requirement. However, the variety of related knowledge representations that will usually be present in a knowledge-based system, particularly in the acquisition phase, makes the editing of knowledge bases very difficult to support. If representation A, for example text, is processed to give representation B, for example entities, attributes and values, then the editing of the material in representation A should result in changes to that in representation B, and vice versa. However, the relation between representations is usually not functional but merely constrained, that is knowledge in representation A does not determine that in representation B, and vice versa, but only constrains it in some way as shown in Figure 8. Thus, the result of editing knowledge in one representation does not necessarily result in well-defined changes in other representations, only the detection of constraint violations. The editing system should minimally highlight the violations, but usually it can do more, for example it can adumbrate "likely" changes in other representations that will remove the violation.



Fig.8 Constraint propagation in editing different representations

Validation is also a complex topic in its own right [1, 40, 41], and it is important that a variety of tools are provided to support different forms of validation. The objective validation of the knowledge base in terms of its performance, its subjective validation in comparison with experts, and its referential validation in comparison with other knowledge sources should be supported. Objective validation may require significant case histories that can be collected only from an operational system in actual applications. Referential validation may require deep analysis of derivations of the knowledge base from other knowledge bases. There is a natural separation in subjective validation between experts' direct agreement that their rationale corresponds to that of the knowledge base, and observed correspondence between their behavior and that of the system based on the knowledge base. Both types of validation should be supported.

The validation of knowledge-based systems has all the dimensions of normal scientific validation together with those concerned specifically with the emulation of expertise [17, 20]. As shown in Figure 9, the system can be validated through four primary pathways:

- Through its correspondence to actual events checked against empirical data;
- Through its efficacy in applications checked against actual applications;
- Through the adequacy of its foundations checked against other disciplines corresponding to the derivation of presuppositions from other disciplines—this is sometimes termed 'deep knowledge';
- Through the comprehensibility of explanations given checked against the relevant reference community—corresponding to explanation and communicative action.



Fig.9 Dimensions of validation of knowledge-based systems

The expert systems paradigm introduces the two additional pathways shown on the right of Figure 9 by taking into account the possibility that there may be "experts" who are competent in applications and explanations:

- Through its correspondence to expert performance checked against the behavior of experts in applications;
- Through its correspondence to expert knowledge checked against the rationale of experts in communicating with their reference community.

This diagram may be iterated a number of times to show the source of the knowledge structures of experts from empirical data and other disciplines, introducing considerations of the validity of experts and reference communities. Similarly, it may be iterated to show the way in which other disciplines in turn are validated against their own empirical data, deeper foundations, applications and reference communities. There is no intrinsic conflict between the additional pathways of validation introduced and the positivist viewpoint of science emphasizing the primacy of empirical data. However, there is a major increase in the realism of this model in modeling the actual processes of knowledge acquisition rather than the normative criteria adopted in 'science' as ultimate arbitrators.

6 THE MAJOR SOURCES FOR KNOWLEDGE ACQUISITION

The significant features of the knowledge acquisition component on the lower right of Figure 10 are best analyzed by considering the natures of the classical and knowledge-based system design paradigms. It is clear that knowledge-based systems use some new computational techniques, but perhaps less apparent that the system design approach involved is itself fundamentally changed. However, major paradigm shifts are involved [23]. The classical approach in decision and control system design is the instrumentation, data collection, modeling and optimization sequence shown on the left of the diagram of Figure 10:

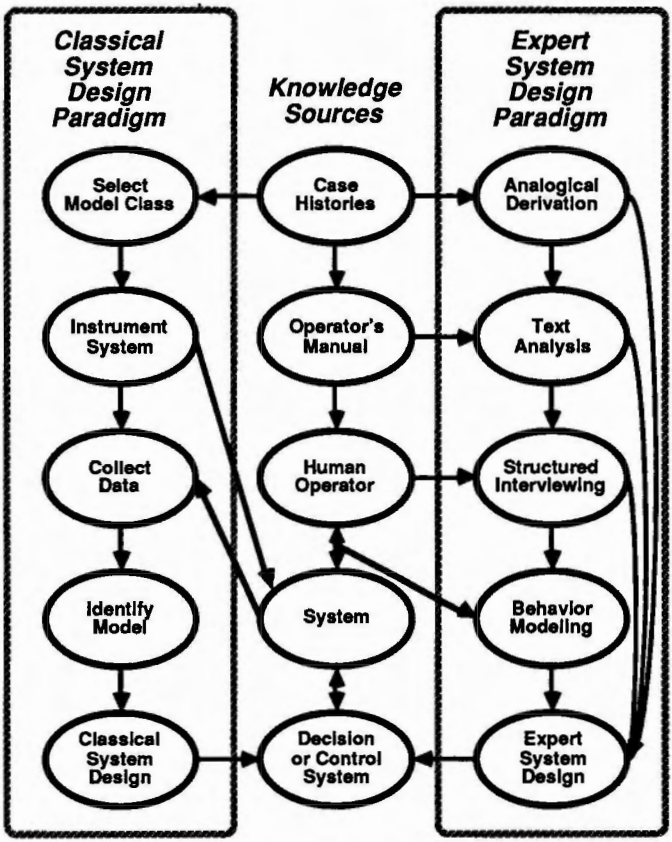


Fig.10 Classical and knowledge-based system design paradigms

- Knowledge of past case histories is used to select a class of system models;
- The information required to discriminate within this class determines how the system should be instrumented for data acquisition;
- Data is collected from the system through the instrumentation;
- A model is identified from the model class which best fits the data;
- This model is used to design a decision or control system for optimal performance.

This approach to system design underlies the methodologies of the physical sciences and technologies based on them. It has the merit that it has been extremely successful in engineering much of the technological infrastructure of our current society including our manufacturing industries. However, this approach is successful only to the extent that the systems under consideration are amenable to instrumentation and modeling. Its greatest successes have been where this amenability can be achieved normatively, that is in cases where the system to be controlled is itself a human artifact. For example, linear systems theory has become a major tool in systems engineering not because most natural systems are linear—they are not. The implication is in the opposite direction: that linear systems are mathematically tractable and that we design artificial systems to be linear so that we may model them readily.

The knowledge-based systems paradigm may be seen to be particularly applicable when it is not possible to model the system but there is an alternative source of data available because human operators are able to performance the decision or control task. The right hand column of Figure 10 shows the use of knowledge sources in the expert system design paradigm:

- Structured interviewing may be used to acquire knowledge directly from the operators;
- Behavior modeling may be used to identify the operators' strategies even if they are unaware of them or give incorrect ones in interviews;
- Text analysis may be used with instructional material such as the operators' manuals;
- Reasoning by analogy may be used based on the case histories without the data collection of the classical system design paradigm.

Note that the classical and knowledge-based system design paradigms, and the various acquisition techniques for knowledge-based systems, need not be regarded as competitive. They may all be used in the system design, and it may be that a collection of weak methods is necessary because there is no overall strong method that can be applied.

The integration of the classical systems analysis with expertise transfer techniques is a major task. The primary problems currently are in the widely different forms of knowledge representation usually involved, and these will be noted in a later section in terms of integration of conventional and knowledge-based system components. However, at a knowledge acquisition level the problem can be seen as one of the imbalance in the types of knowledge structures studied in depth to date. There has been an imbalance favoring symbolic knowledge to redress the past emphasis on quantitative methods. This needs redressing with the knowledge structures underlying arithmetic being integrated with those underlying non-arithmetic symbol manipulation.

7 KNOWLEDGE ACQUISITION TOOLS

A number of tools have been developed for the rapid prototyping of knowledge-based systems automating some or all of the methodologies on the right of Figure 10. The objective of systems such as PLANET [36, 37, 38], ETS [3, 4], MOLE [13], SALT [29], KITTEN [39], KNACK [28], KRITON [12], OPAL [33], AQUINAS [6], C4.5 [34] and KSSO [16,24] is to expedite the process of knowledge acquisition and transfer to knowledge-based systems.

Boose [5] has surveyed a very wide range of knowledge acquisition tools and classified them along a variety of dimensions, analyzing the applicability of tools to tasks. He distinguishes three classes of tasks:-

- Analysis tasks
  - —Classification: categorizing based on observables;
  - —Debugging: prescribing remedies for malfunctions;
  - —Diagnosis: inferring system malfunctions from observable;
  - —Interpretation: inferring situation descriptions from sensor data.
- Synthesis tasks
  - —Configuration: configuring collections of objects under constraints in relatively small search spaces;
  - —Design: configuring collections of objects under constraints in relatively large search spaces;
  - —Planning: designing actions;
  - —Scheduling: planning with strong time and/or space constraints.
- Tasks combining analysis and synthesis
  - —Command and control: ordering and governing overall system control;
  - —Instruction: diagnosing, debugging and repairing student behavior;
  - —Monitoring: comparing observations to expected outcomes;
  - —Prediction: inferring likely consequences of given situations;
  - —Repair: executing plans to administer prescribed remedies.

Boose notes that most knowledge acquisition tools currently are best suited to analysis tasks and are weak in acquiring expertise in synthesis.

The general requirements for a system to automate knowledge acquisition for knowledge-based systems are [39]:

- The knowledge acquisition system tools should be domain independent;
- The tools should be directly applicable by experts without intermediaries;
- The tools should be able to access a diversity of knowledge sources including text, interviews with experts, and observations of expert behavior;
- The knowledge acquisition system should be able to encompass a diversity of perspectives including partial or contradictory input from different experts;
- The system should be able to encompass a diversity of forms of knowledge and relationships between knowledge;
- The system should be able to present knowledge from a diversity of sources with clarity as to its derivation, consequences and structural relations;
- Users of the knowledge acquisition system should be able to apply the knowledge in a variety of familiar domains and freely experiment with its implications;
- The system should make provision for algorithmic expression of knowledge where appropriate;
- The system should make provision for validation studies;
- As much of the operation of the system as possible should be founded on well-developed and explicit theories of knowledge acquisition, elicitation and representation;
- As the overall knowledge acquisition develops it should converge to an integrated system.

It is particularly important in describing tools that they are presented in the context of user interaction, since their primary function is to operate directly with experts. It is impossible to describe all the knowledge acquisition tools currently in the literature, and the references cited at the beginning of this section and specialist collections should be consulted for details and examples of use [7, 8, 22, 30].

8 CONCLUSIONS: IMPEDIMENTS TO SYSTEM DEVELOPMENT

Expert systems technology, like many other computing developments, has been 'oversold', and the expectations of what may be achieved through the use of knowledge-based system technology have been raised too high. The 'bottleneck' of knowledge engineering discussed in Section 4 still remains. The quotation at the head of that section dates from 1983 and meanwhile major effort has swung into research on knowledge acquisition techniques and there has been a substantial increase in the experience of developing expert systems. Notwithstanding this research and experience, there are still major problems in the development of knowledge-based systems, and it is appropriate to conclude this chapter by summarizing these problems—Figure 11 gives a synopsis of the major impediments to knowledge-based system development.

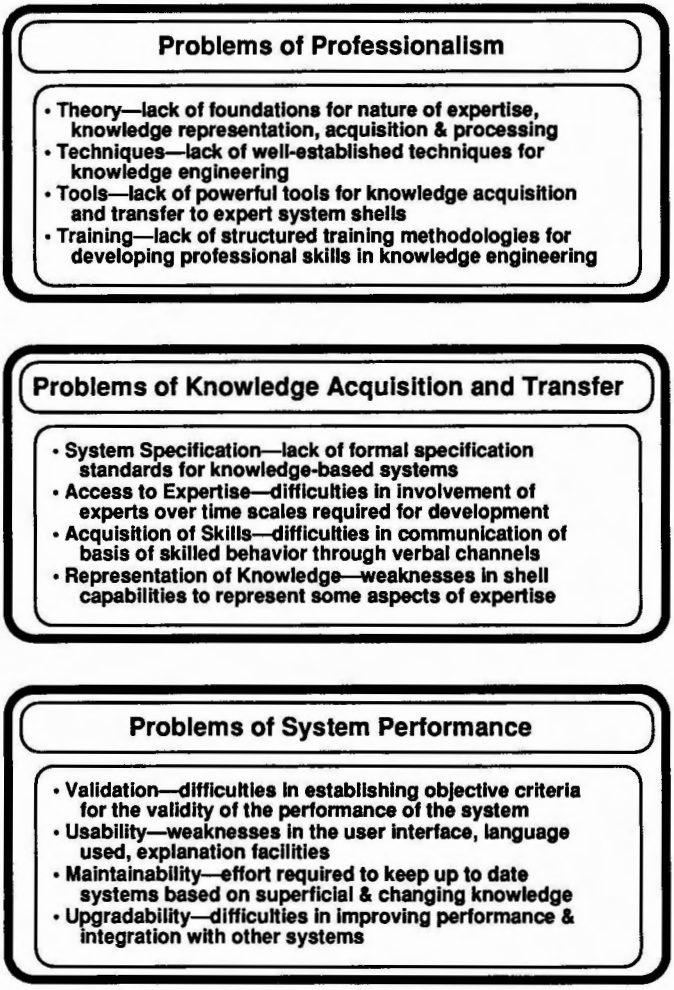


Fig.11 Impediments to knowledge-based system development

The top box in Figure 11 summarizes what may be termed the problems of *professionalism* in developing knowledge-based systems:

- *Theory*: the lack of scientific and engineering foundations for knowledge-based systems is a major impediment to the development of professional standards for expert system development. We understand remarkably little about the nature of expertise even though it one of the most valued commodities in our society. We also have little fundamental understanding of knowledge, the ways in which it is represented in human thought processes, the ways in which it is used in human problem solving, and the ways in which it is transmitted between people.
- *Techniques*: the lack of well-established techniques for knowledge engineering is a major problem for new development teams. There is some folklore, generally mutually contradictory, a few rules of thumb based on experience, but so far no in-depth studies of what is actually done and how successful it has been.
- *Tools*: the lack of commercially available tools for knowledge acquisition and transfer is also a major problem. There are now many research reports of such tools but few of those developed are available as products.
- *Training*: the training situation is improving all the time, largely because it has become the major secondary industry to the supply of expert system shells. A range of courses is being offered by commercial and university organizations. It is still a problem to get good quality courses targeted to specific requirements and on the schedules required.

The middle box in Figure 11 summarizes problems of *knowledge acquisition and transfer* in developing knowledge-based systems. Knowledge acquisition and transfer processes in human society are complex and poorly understood. Neurology, psychology, linguistics, education, sociology, anthropology, philosophy and systems theory all have significant contributions to make. However, integrating them, disambiguating jargon, and combining different objectives and perspectives are major problems. There is a variety of short-term problems arising from these weaknesses in foundations:

- *System Specification*: the formal specification of software that has made software engineering a professional discipline which can ensure quality within well-defined budgets and schedules is notably lacking for expert system developments. We seem at the stage of conventional software development of twenty years ago. This should change as expert system developments move out of the laboratories into the hands of experienced software development groups trained in software management. There is little in the literature to support and expedite this change currently, and we need to develop specification disciplines for expert systems.
- *Access to Expertise*: the problems of getting experts to make the time commitments necessary to the development of expert systems should never be underestimated. Successful projects inevitably seem to have involved the experts in such a way that they become personally committed to the success of the system.
- *Acquisition of Skills*: communicating the basis of skilled performance between people, and between people and computers, is very difficult.
- *Representation of Knowledge*: encoding knowledge within the representation schemes of current shells can present major difficulties. Production rules as used in early expert systems offer very limited facilities for encoding simple knowledge structures at a single level. Frames and their generalization through objects allow conceptual hierarchies to be better expressed. Neither rules nor frames are adequate to encode the procedural or temporal knowledge which is often a key component of expertise.

The bottom box in Figure 11 summarizes problems of *system performance* in developing knowledge-based systems. Rapid prototyping is a significant practical objective and techniques for accelerated initial development without loss of long-term quality are now common in conventional data processing. However, expert system development shows

weaknesses in some areas of performance which we tend to take for granted in standard information technology. There are problems with:

- *Validation*: because we can rarely define and quantify expertise, the validation of expert systems can be very difficult. Do we subjectively validate against the opinions of the expert or do we attempt objective validation of performance against performance standards, accepting that the experts may not themselves satisfy such criteria. How, do we determine whether the knowledge base captures the expert's relevant knowledge, accurately, consistently and completely? There are very few validation studies in the literature.
- *Usability*: the usability of expert systems raises many problems. If we accept the jargon of the expert and the rules of inference based on it, how do we ensure that the operators will use this jargon in the same way as the expert so that the inferences remain valid. It is such considerations that make nonsense of the popular stereotype that access to encoded expertise can make experts of all of us. Understanding the questions, advice and explanations from an expert system generally requires an operator with the same professional background as the expert.
- *Maintainability*: the superficiality of the knowledge encoded in current expert systems raises major problems of maintenance. Keeping the knowledge base up to date may require an ongoing effort at the same level as the original development, a situation very different from most conventional data processing. The usual problems of extensive maintenance not introducing problems of its own are at least as severe for expert systems as for standard data processing.
- *Upgradability*: many expert systems, and the shells supporting them have been designed as stand-alone packages with little scope for integration into large-scale data processing environments. These systems will be difficult to upgrade and integrate with other software.

This emphasis on problems impeding knowledge-based system development is not intended to be negative. In any major technological advance there are significant problems and it is the recognition and classification of these that indicates the growing maturity of the field. There is intensive, world-wide research in all the problem areas noted above, and growing experience in how to overcome them. By recognizing the problems project managers may be able to target their objectives to avoid some, and allocate their resources to overcome others.

ACKNOWLEDGEMENTS

Financial assistance for this work has been made available by the Natural Sciences and Engineering Research Council of Canada. We are grateful to many colleagues for discussions over the years that have influenced this paper. In particular we would like to thank John Boose, Jeff Bradshaw, Alain Rappaport and Brian Woodward for access to their own research and many stimulating discussions.

REFERENCES

[1] Benbasat, I. & Dhaliwal, J.S. (1989) A framework for the validation of knowledge acquisition. **Knowledge Acquisition 1**(2), 215-233 (June).

[2] Blume, S.S. (1974) **Towards a Political Sociology of Science**. New York: Free Press.

[3] Boose, J.H. (1984) Personal construct theory and the transfer of human expertise. **Proceedings AAAI-84**, 27-33. California: American Association for Artificial Intelligence.

[4] Boose, J. H. (1986) **Expertise Transfer for Expert System Design**. New York: Elsevier.

[5] Boose, J. H. (1989) A survey of knowledge acquisition techniques and tools. **Knowledge Acquisition 1**(1), 3-37 (March).

[6] Boose, J.H. & Bradshaw, J.M. (1987) Expertise transfer and complex problems: using AQUINAS as a knowledge acquisition workbench for knowledge-based systems. **International Journal of Man-Machine Studies(26),** 3-28.

[7] Boose, J.H. & Gaines, B.R., Eds. (1988) **Knowledge Acquisition Tools for Expert Systems**. London, Academic Press.

[8] Boose, J.H. & Gaines, B.R., Eds. (1990) **The Foundations of Knowledge Acquisition**. London, Academic Press.

[9] Breuker, J. & Wielinga, B. (1987) Use of models in the interpretation of verbal data. Kidd, A. (Ed) **Knowledge Elicitation for Expert Systems: A Practical Handbook.** Plenum.

[10] Chandrasekaran, B. (1988) Generic tasks as building blocks for knowledge-based systems: the diagnosis and routine design examples. **The Knowledge Engineering Review, 3**(3), 183-211.

[11] Clancey, W. (1989) Viewing knowledge bases as qualitative models. **IEEE Expert, 4**(2), 9-23.

[12] Diederich, J., Ruhmann, I. & May, M. (1987) KRITON: A knowledge acquisition tool for expert systems. **International Journal of Man-Machine Studies, 26**(1), 29-40.

[13] Eshelman, L., Ehret, D., McDermott, J. & Tan, M. (1987) MOLE: A tenacious knowledge acquisition tool. **International Journal of Man-Machine Studies, 26**(1), 41-54

[14] Feigenbaum, E.A. (1980) Knowledge Engineering: the Applied Side of Artificial Intelligence. **Report STAN-CS-80-812.** Department of Computer Science, Stanford University.

[15] Gaines, B.R. (1987) An overview of knowledge acquisition and transfer. **International Journal of Man-Machine Studies 26**(4), 453-472 (April).

[16] Gaines, B.R. (1987) Rapid prototyping for expert systems. Oliff, M.D., Ed. **Intelligent Manufacturing: Proceedings from First International Conference on Expert Systems and the Leading Edge in Production Planning and Control**. pp.45-73. Menlo Park, California, Benjamin Cummins.

[17] Gaines, B.R. (1988) Second generation knowledge acquisition systems. **Proceedings of the Second European Workshop on Knowledge Acquisition for Knowledge-Based Systems (EKAW'88)**. GMD-Studien Nr.143. pp.17-1-17-14. Bonn, Germany: Gellsellschaft für Mathematik und Datenverarbeitung (June).

[18] Gaines, B.R. (1988) Positive feedback processes underlying the formation of expertise. **IEEE Transactions on Systems, Man & Cybernetics SMC-18**(6), 1016-1020 (November).

[19] Gaines, B.R. (1988) Software engineering for knowledge-based systems. **Proceedings of CASE'88, Second International Workshop on Computer-Aided Software Engineering**. pp. 14-1-14-7.

[20] Gaines, B.R. (1989) Social and cognitive processes in knowledge acquisition. **Knowledge Acquisition 1**(1), 39-58 (March).

[21] Gaines, B.R. (1989) Integration issues in knowledge support systems. **International Journal of Man-Machine Studies 31**(5), 495-515 (November).

[22] Gaines, B.R. & Boose, J.H., Eds. (1988) **Knowledge Acquisition for Knowledge-Based Systems**. London, Academic Press.

[23] Gaines, B.R. & Shaw, M.L.G. (1985) From fuzzy sets to expert systems. **Information Science 36** (1-2), 5-16 (July).

[24] Gaines, B.R. & Shaw, M.L.G. (1987) Knowledge support systems. **ACM MCC-University Research Symposium.** Austin, Texas: MCC. 47-66.

[25] Hagstrom, W.O. (1965) **The Scientific Community**. New York: Basic Books.

[26] Hawkins, D. (1983) An analysis of expert thinking. **International Journal of Man-Machine Studies, 18**(1), 1-47 (January).

[27] Hayes-Roth, F., Waterman, D.A. & Lenat, D.B., Eds. (1983) **Building Expert Systems**. Reading, Massachusetts: Addison-Wesley.

[28] Klinker, G., Bentolila, J., Genetet, S., Grimes, M. & McDermott, J. (1987) KNACK—report-driven knowledge acquisition. **International Journal of Man-Machine Studies, 26**(1), 65-79.

[29] Marcus, S. (1987) Taking backtracking with a grain of SALT. **International Journal of Man-Machine Studies 26**(4), 383-398 (April).

[30] Marcus, S. (Ed.) (1988) **Automating Knowledge Acquisition for Expert Systems**. New York: Kluwer.

[31] McDermott, J. (1988) Preliminary steps toward a taxonomy of problem solving methods. Marcus, S. (Ed) **Automating Knowledge Acquisition for Expert Systems,** Kluwer: Boston.

[32] Merton, R.K. (1973) **The Sociology of Science: Theoretical and Empirical Investigations**. Chicago: University Press.

[33] Musen, M.A. (1989) **Automated Generation of Model-Based Knowledge Acquisition Tools.** London: Pitman.

[34] Quinlan, J.R. (1987) Simplifying decision trees. **International Journal of Man-Machine Studies 27** (3), 221-234 (September).

[35] Schön, D.A. (1983) **The Reflective Practitioner**. New York: Basic Books.

[36] Shaw, M.L.G. & Gaines, B.R. (1983) A computer aid to knowledge engineering. **Proceedings of British Computer Society Conference on Expert Systems**, 263-271 (December) Cambridge.

[37] Shaw, M.L.G. & Gaines, B.R. (1986) Interactive elicitation of knowledge from experts. **Future Computing Systems, 1**(2), 151-190.

[38] Shaw, M.L.G. & Gaines, B.R. (1987) An interactive knowledge elicitation technique using personal construct technology. Kidd, A., Ed. **Knowledge Eliciation for Expert Systems: A Practical Handbook**. pp.109-136. Plenum Press.

[39] Shaw, M.L.G. & Gaines, B.R. (1987) KITTEN: Knowledge Initiation & Transfer Tools for Experts & Novices. **International Journal of Man-Machine Studies 27**(3), 251-280 (September).

[40] Shaw, M.L.G. & Gaines, B.R. (1989) A methodology for recognizing conflict, correspondence, consensus and contrast in a knowledge acquisition system. **Knowledge Acquisition 1**(4), 341-363 (December).

[41] Shaw, M.L.G. & Woodward, J.B. (1988) Validation in a knowledge support system: construing and consistency with multiple experts. **International Journal of Man-Machine Studies, 29,** 329-350.

[42] Stich, S.P. & Nisbett, R.E. (1984) Expertise, justification and the psychology of inductive reasoning. Haskell, T.L., Ed. **The Authority of Experts**. pp. 226-241. Bloomington, Indiana: Indiana University Press.

Knowledge Acquisition Tools, Methods, and Mediating Representations

John H. Boose
Boeing Advanced Technology Center, Boeing Computer Services
P.O. Box 24346, Seattle, WA, 98124, USA

1.0 Knowledge Acquisition

Acquiring and modeling knowledge effectively can be the most time-consuming portion of the knowledge engineering process. Little methodology is practiced beyond unstructured interviewing. Automated methods are, for the most part, still in the research stage.

Several well-known factors contribute to interviewing problems:

- o Plausible lines of reasoning can have little to do with actual problem-solving.
- o Academic (novice) knowledge is often obtained in place of compiled (expert) knowledge.
- o Experts may be insecure. They could be afraid of losing their jobs; they may not want computers encroaching on their "private domain;" they may not want to expose their problem-solving methods to the scrutiny of colleagues or of the general public.
- o Interpersonal interviewing problems can result when knowledge engineers are not trained in interviewing techniques.
- o There are inherent problems in introspection.
- o Protocol analysis (discussed below) is labor intensive, error-prone, and results in a series of random behavior samples that must be synthesized by the knowledge engineer.

Clancey summarized the problem as follows (1986b):

> "Knowledge acquisition is not just a problem of accessing and translating what is already known, but the familiar scientific and engineering problem of formalizing models for the first time."

This section discusses the role of modeling languages, and manual, semi-automated (interactive), and automated (machine learning) knowledge acquisition techniques.

2.0 Modeling Expertise - Mediating Representations

Many knowledge acquisition research ideas and experiments for semi-automated techniques have been tested in the last several years. Successful approaches seem to rely on effective *mediating representations* - problem modeling languages that help bridge the gap between experts and computer implementations (Figure 1). These representations may be implemented with paper and pencil or in computer systems. Effective mediating representations obviate the need to build and maintain systems in lower-level programming languages. They empower effective communication between experts, knowledge engineers, and efficient implementation. The mediating representation (1) should closely match the representation language the expert normally uses to describe and solve problems, or (2) should be easily learned and intuitive to use, so that the expert has little difficulty transforming his normal language into the new representation.

Some of these representations are at higher levels of abstraction than others (Figure 2). Lower-level representations are often used as building blocks for higher-level representations. Higher-level representations are often transformed into lower-level representations by a computer to perform inference (for instance, decision trees might be transformed into rules). Some representation systems provide a loose framework for collections of other representations (for example, hypertext). Different types of knowledge may be more easily used in one form of representation than another. For instance, tables may emphasize entity-attribute information while semantic networks emphasize the relationships between objects.

The use of a spreadsheet is analogous to the use of mediating representations in knowledge acquisition tools. In fact, a spreadsheet could be classified as an effective mediating representation. A spreadsheet provides users with an efficient representation language for building business models. Previously such models were programmed directly in languages such as Basic or COBOL. Many more users can build business models on computers because spreadsheets are available. The emphasis is on *business modeling* skill, not *programming* skill (Figure 3). It can be argued that spreadsheets help improve the *quality* of business models because users (1) are not burdened with computer programming, (2) concentrate on important aspects of the model directly, (3) can easily change the model, and (4) can more easily maintain the model.

Effective mediating representations allow experts and knowledge engineers to concentrate on important aspects of problem-solving. Rules and other lower-level knowledge representations were a step forward from LISP or FORTRAN, and higher-level representations are a step forward from rules.

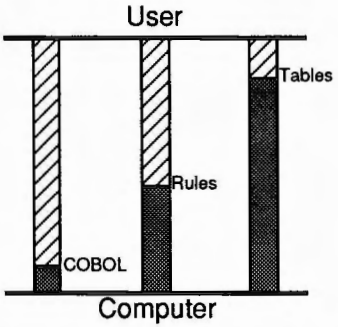


Figure 1. Mediating representations (such as tables) help bridge the gap between the expert (or knowledge engineer) and the computer.

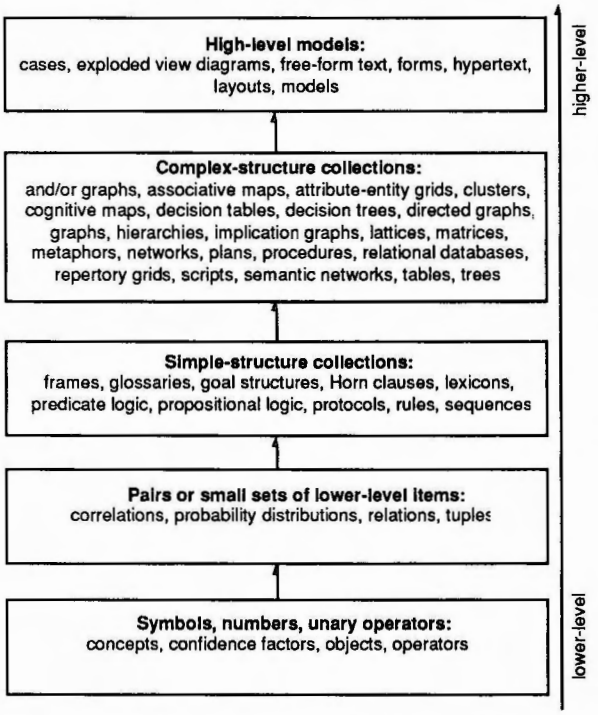


Figure 2. Example mediating representations for knowledge-based systems.

There seems to be a basic trade-off in a representation language between *acquirability* and *expressive power* (Gruber, 1989a). Programming languages are a powerful methods for implementing ideas on computers, but are probably the most difficult to use, especially for nonspecialists. High-level representations such as forms may be intuitive to use but are very rigid and this limits their applicability. Some representations make it easier to acquire knowledge for specialized applications,

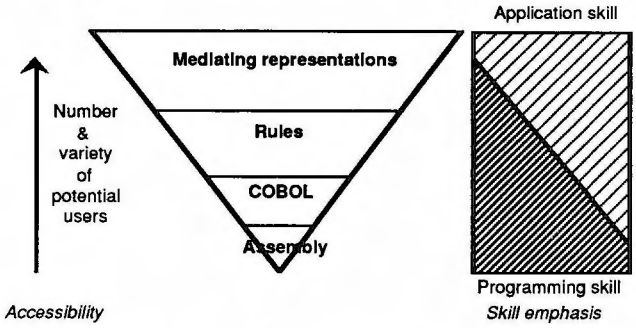


Figure 3. Higher-level mediating representations enable access.

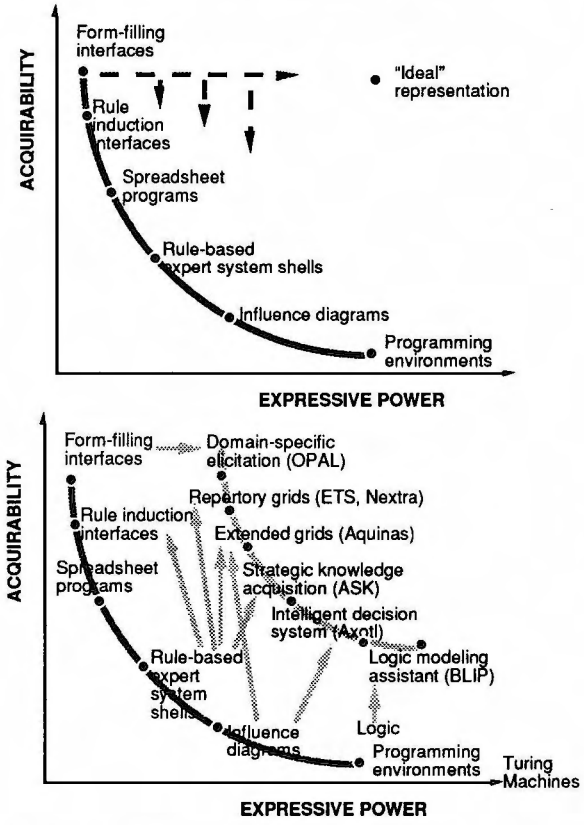


Figure 4. There is a basic tradeoff between knowledge acquirability and expressive power (Gruber, 1989). The more powerful the representation, the more difficult it is to maintain a high level of acquirability. Mediating knowledge representations in knowledge elicitation tools can act to pull the curve upward to reduce the effects of the tradeoff (Bradshaw, Russo, Covington, and Boose, 1990).

some representations have more expressive power. The dotted arrows in the first part of Figure 4 show the dilemma of the knowledge elicitation tool implementor trying to design an "ideal" representation that combines the naturalness of form-filling interfaces with the power and flexibility of a Turing

machine: the more powerful the representation, the more difficult it is to maintain a high level of acquirability. Likewise, developers of programming environments try to represent programming constructs through higher-level languages and development environments such as CASE tools. But, inevitably, as representations become more powerful, expressiveness and flexibility are lost. Mediating knowledge representations in knowledge elicitation tools can act to pull the trade-off curve upward to reduce the effects of the tradeoff (Bradshaw, Russo, Covington, and Boose, 1990). References for the tools appear later in this section.

3.0 Knowledge Acquisition Tools and Techniques

Both manual and computer-based knowledge acquisition techniques and tools are described below (Figure 5). First, a short catalog of manual techniques is listed. Next, computer-based knowledge acquisition tools and techniques are classified and briefly described. Computer-based techniques are divided into interactive (semi-automated) and automated (machine learning) techniques.

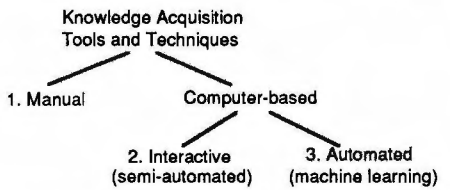


Figure 5. Elicitation tools and techniques are classified by manual, interactive, and automated methods.

3.1 Manual Methods

Representative manual knowledge elicitation techniques are described below.

Brainstorming

Crawford Slip Method - rapidly generate a large number of ideas (Rusk and Krone, 1984)

Interviewing

Unstructured Interview - ask general questions and hope for the best, recording as much as possible (Brown, B., 1989), (Kidd and Cooper, 1985), (Freeman, 1985), (von Martial and Victor, 1988), (Trimble and Cooper, 1987), (Waldron, 1989), (Welbank, 1987a)

Semi-Structured Interview - interview with open questions and a list of topics to cover (Bradshaw, 1991), (Crandall, 1989), (Forsythe and Buchanan, 1988, 1989), (LaFrance, 1987), (Welbank, 1987b)

Structured Interview - interview with strict agenda and list of specific questions relating to features of system (Becker and Balasubramanian, 1989), (Bradshaw, 1991), (Clarke, 1987), (De La Garza and Ibbs, 1989), (Freiling, Alexander, Messick, Rehfuss, and Shulman, 1985), (Haers and d'Ydewalle, 1989), (Klein, Calderwood, and MacGregor, 1989), (Shah, 1989), (Slocombe, Moore, and Zelouf, 1986), (Tournat, 1989)

Teachback Interview - knowledge engineer demonstrates understanding of expertise by paraphrasing or solving a problem (Johnson and Johnson, 1987), (Johnson and Tomlinson, 1988)

Neurolinguistic Programming - observe physical cues (eye movement, body language) to enhance communication with expert (Micciche and Lancaster, 1989)

Tutorial Interview - expert delivers a lecture (Welbank, 1987b)

Interviewing Process Issues - (Fredman, 1987)

Knowledge Organization Techniques

Card Sorting - sort objects on cards to help structure knowledge (Burton, Shadbolt, Hedgecock, and Rugg, 1987), (Gammack and Young, 1984)

Ethnoscience Techniques - use anthropological techniques to elicit names of items and categories in a semantic domain (Benfer and Furbee, 1989)

Knowledge Analysis - systematically analyze and document knowledge during early phases of knowledge engineering based on mediating representations (Johnson, Tomlinson, and Johnson, 1988), (Regoczei and Hirst, 1988)

Mediating Representations - work with representations that mediate between the expert and the implementation - (Johnson, 1987), (Johnson, Tomlinson, and Johnson, 1988), (Young and Gammack, 1987)

Overcoming Bias - recognize and correct bias from knowledge sources (Cleaves, 1987), (Meyer and Booker, 1989), (Meyer, Mniszewski, and Peaslee, 1989), (Moray, 1985), (Stephanou, 1987), (Tolcott, Marvin, and Lehner, 1989)

Psychological Scaling (including multidimensional scaling) - use scaling techniques to help structure knowledge (Burton, Shadbolt, Hedgecock, and Rugg, 1987), (Saaty, 1981), (Golden, Wasil, and Harker, 1989), (Williams and Thomas, 1987), (Young and Gammack, 1987)

Uncertain Information Elicitation and Representation - expert encodes uncertainty about the problem (Beyth-Marom and Dekel, 1985), (Dean and Kanazawa, 1989), (Fischhoff, 1989), (Hink and Woods, 1987), (Kahneman, Slovic, and Tversky, 1982), (Levi, 1989), (Mullin, 1989), (Pearl, 1986), (Shafer and Tversky, 1985), (Spetzler and Stael von Holstein, 1983), (Stael von Holstein and Matheson, 1978), (Tversky and Kahneman, 1974, 1981), (Tversky, Sattah, and Slovic, 1987), (Wallsten and Budescu, 1983), (Yang and Okrent, 1989)

Protocol Analysis Techniques

Participant Observation - knowledge engineer becomes an apprentice or otherwise participates in the expert's problem-solving process (Welbank, 1987b)

Protocol Analysis (Case Walk-Through / Eidetic Reduction / Observation / Process-Tracing) - record and analyze transcripts (from experts thinking aloud during tasks or other sources) (Belkin, Brooks, and Daniels, 1987), (Breuker and Wielinga, 1987a,b), (Burton, Shadbolt, Hedgecock, and Rugg, 1987), (Clarke, 1987), (Cordingley, 1987), (Crandall, 1989), (De La Garza and Ibbs, 1989), (Ericsson and Simon, 1984), (Eskenazi, Faure, Frediani, and Saitta, 1989), (Gammack and Young, 1984), (Grover, 1983), (Haers and d'Ydewalle, 1989), (Hausen and Neusser, 1987), (Johnson, Zualkernan, and Garber, 1987), (Killin and Hickman, 1986), (Kwasnik, 1989), (Laske, 1987), (Laskey, Cohen, and Martin, 1989), (Littman, 1987), (Waldron, 1985), (Wetter and Schmalhofer, 1988)

User Interface Techniques

Wizard of Oz Technique - an expert simulates the behavior of a future system (Sandberg, Winkels, and Breuker, 1988)

Several taxonomies exist that classify manual knowledge elicitation methods. Alternate classification schemes include Hoffman's breakdown including the analysis of familiar tasks, interviewing, special tasks, and constrained processing tasks (Hoffman, 1989); interviewing breakdowns by Shaw and Woodward (1989); and approaches classified by knowledge elicitation task (Morik, 1989). Bradshaw presents a detailed interviewing methodology for classification problems, emphasizing debiasing techniques (1991). These schemes and others are described in more detail in in Boose and Gaines (1990).

3.2 Interactive (Semi-Automated) Methods

3.2.1 Classification Framework

Interactive knowledge elicitation tools can be associated with knowledge-based application *problems* and *problem-solving methods*. This descriptive approach provides a framework for analyzing and comparing elicitation tools, and focuses the task of building knowledge-based systems on the knowledge elicitation process and accompanying knowledge representations.

Several taxonomies exist for categorizing knowledge-based application problems. One common scheme, illustrated below, divides them into *analysis* (interpretation) problems and *synthesis* (construction) problems (Clancey, 1986a). Generally, analysis problems involve identifying sets of objects based on their features. One characteristic of analysis problems is that a complete set of solutions can be enumerated and included in the system. Synthesis (generative, or constructive) problems require that a solution be built up from component pieces or subproblem solutions. In synthesis problems there are too many potential solutions to enumerate and include explicitly in the system.

Analysis and synthesis problems can be broken down into sub-problem areas. The following classification is used in the remainder of this section, although the same knowledge elicitation tool mapping idea can be applied to other problem taxonomies (adapted from Hayes-Roth, Lenat, and Waterman, 1983). Alternate taxonomies are described in Boose and Gaines (1990).

Analysis problems

- o Classification - categorizing based on observables.
- o Debugging - prescribing remedies for malfunctions.
- o Diagnosis - inferring system malfunctions from observables.
- o Interpretation - inferring situation descriptions from sensor data.

Synthesis (construction) problems

- o Configuration - configuring collections of objects under constraints in relatively small search spaces.
- o Design - configuring collections of objects under constraints in relatively large search spaces.
- o Planning - designing actions.
- o Scheduling - planning with strong time and/or spatial constraints.

Analysis and Synthesis Problems

- o Command and control - ordering and governing overall system control.
- o Instruction - diagnosing, debugging and repairing student behavior.
- o Monitoring - comparing observations to expected outcomes.
- o Prediction - inferring likely consequences of given situations.
- o Repair - executing plans to administer prescribed remedies.

Relationships exist between problems and problem-solving methods. For instance, the *heuristic classification* problem-solving method has been used for many knowledge-based systems that solve analysis problems (Clancey, 1986a), and is employed in a variety of knowledge-based system development tools, or "shells" (S.1, M.1, EMYCIN, TI-PC, and so on). In heuristic classification, data is abstracted up through a problem hierarchy, problem abstractions are mapped onto solution abstractions, and solution abstractions are refined down through the solution hierarchy into specific solutions.

General methods for solving synthesis problems are sparse; Clancey classified these methods under *heuristic construction*. Usually, a specific method is developed to solve a particular problem (such as SALT's *propose-and-revise* method or OPAL's *skeletal-plan-refinement* method), but it may be difficult to generalize the method. Some form of directed backtracking or cyclic constraint exploration is often used to explore the problem space.

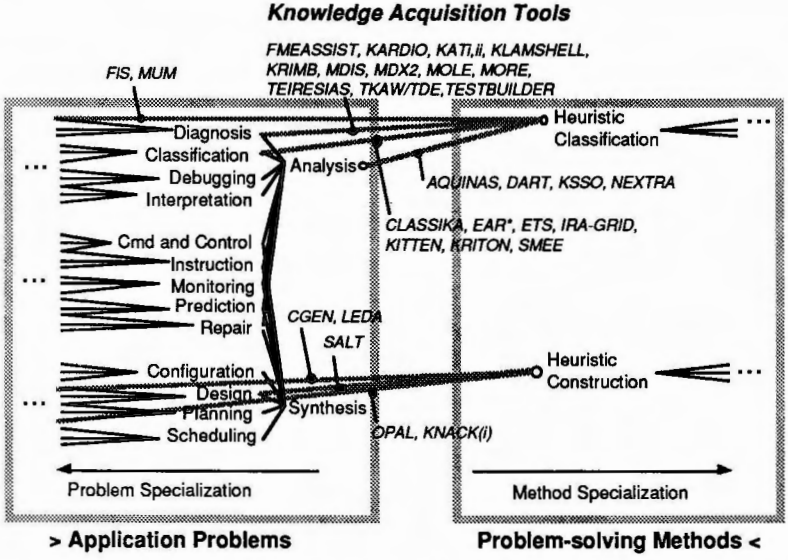


Figure 6. Knowledge elicitation tools may be associated with relationships between application problems and problem-solving methods. Representative tools are shown.

Many problems require a combination of analysis and synthesis problem-solving methods. For instance, Clancey outlines a maintenance cycle requiring monitoring, prediction, diagnosis, and modification; this combines aspects of heuristic classification and heuristic construction.

Musen et al. proposed that knowledge elicitation tools could be associated with specific problems or specific problem-solving methods (Musen, Fagan, Combs, and Shortliffe, 1987). In a related manner, we propose to classify tools with problems *and* problem-solving methods, since most problems are strongly linked to certain types of problem-solving methods. Consequently, certain types of domain knowledge and possibly control knowledge should be acquired to build the corresponding knowledge-based system. This idea was discussed at the First AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop held in Banff, Canada, in November, 1986 (Gaines and Boose, 1990). Builders of interactive knowledge elicitation tools were asked to try and classify their research and the research of others in terms of these relationships. Figure 6 shows a possible mapping of such relationships at a high level in the problem classification hierarchy and a problem-solving method classification hierarchy. Lower levels in the problem hierarchy would be sub-problems (i.e., trouble shooting and symptom analysis would be found under diagnosis), and the leaves of the problem hierarchy would be specific application problems to be solved.

Knowledge elicitation tool research falls into several categories. Descriptions and references for the tools mentioned here are given below.

3.2.2 Semi-Automated Tools - Research Strategies

Research strategy 1: Problem - Method Relationships

Find and clarify knowledge elicitation strategies for a problem-to-method relationship (usually a domain specific problem employing a highly specialized method using much domain knowledge, or a general problem employing a general method with little domain knowledge).

Examples for specific problem domains ("bottom up") include:
ASKE, FIS, MOLTKE, MUM, OPAL, PROTOGÉ, SMAC, STUDENT

Examples for general analysis problems ("top down") include:
AQUINAS, CLASSIKA, DART, EAR*, ETS, FLEXIGRID, IRA-GRID, KITTEN, KRITON, NEXTRA, KSS0, SMEE

Pick a problem, find and develop knowledge elicitation strategies for an applicable method, and then see if the method and strategies will generalize to another problem.

Examples ("middle out") include :
3DKAT, ASKE, BLIP, CANARD, CGEN, DACRON, FMEASSIST, GEN-X, INFORM, KARDIO, KAT (i), KAT (ii), KLAMSHELL, KNACK (i), KNACK (ii), KRIMB, LEDA, MDIS, MDX2, MOLE, MORE, ONTOS, ROGET, QUIZ, SALT, SMAC, TEIRESIAS, TESTBENCH, TESTBUILDER, TKAW/TDE, VIEWPOINT

Research strategy 2: Task and Problem Languages.

Develop representation languages for defining and describing problems and methods.

To some extent, all successful knowledge elicitation tools and methods must employ representations that knowledge engineers or experts can understand. Examples that place particular emphasis on these languages include:
ASTEK, CODE, KADS, MAC
Addis and Bull, 1988 (concept language); Alexander, Freiling, Shulman, Rehfuss, and Messick, 1987 (ontolgy); Bylander and Mittal, 1986 (task primitives); Bylander and Chandrasekaran, 1987 (task primitives); Chandrasekaran, 1988 (task primitives); Davis and Bonnell, 1989 (knowledge dictionary); Freiling, 1988 (ontology); Freiling and Jacobson, 1989 (ontology); Gruber and Cohen, 1987 (task language); Hayward, Wielinga, and Breuker, 1987 (domain ontology); Jacobson and Freiling, 1989 (ontology); Jansen and Compton, 1988, 1989 (knowledge dictionary integration); Johnson, 1987 (mediating representations); Johnson, Tomlinson, and Johnson, 1988 (mediating representations); Rochowiak and Mosley, 1989 (knowledge dictionary); Schreiber, Breuker, Bredeweg, and Wielinga, 1988 (domain ontology); Young and Gammack, 1987 (intermediate representations)

Research strategy 3: Intelligent Editors, Hypermedia.

Build intelligent editors to help AI programmers construct large knowledge bases.

Examples include:
Editors: APPRENTICE, CYC, GKE, KET, KREME, PRED, SEEGRAPH
Hypermedia: ACQUIST, CAMEO, COGNOSYS, KSS0, ONTOS, SMAC

3.2.3 Patterns in Semi-Automated Tools

Figure 7 plots elicitation tools on domain and problem axes. There is a strong concentration of knowledge elicitation tools for diagnostic problems, but few knowledge elicitation tools exist for synthesis problems.

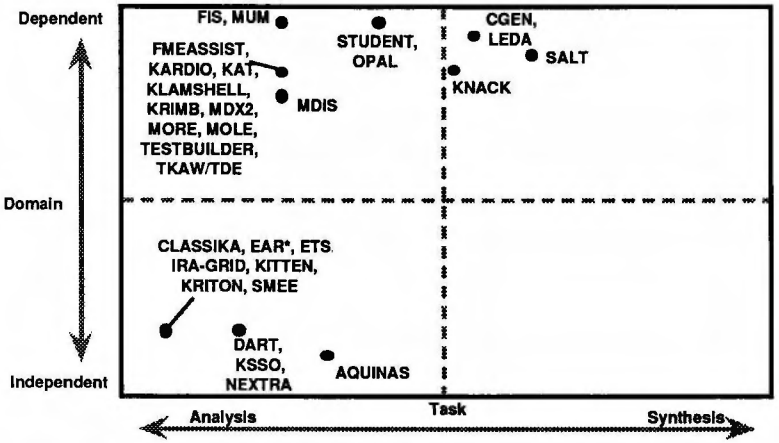


Figure 7. Domain independence vs. problem class table showing representative interactive tools.

Other patterns in the tools are apparent. For instance, some tools try to draw power using strong specific domain knowledge (ASKE, FIS, GKE, LAS, LEAP, MOLTKE, MUM, OPAL, PROTOGÉ, SMAC, STUDENT); other tools try to address a broader range of problems at the expense of built-in domain-specific problem solving power (AQUINAS, CLASSIKA, DART, EAR*, ETS, FLEXIGRID, IRA-GRID, KITTEN, KRITON, NEXTRA, KSS0, SMEE). The few tools that address synthesis problems are domain dependent (CGEN, LEDA, SALT). Few strong commercial tools are available (commercial tools include AutoIntelligence, GeneR, KAT (iii), NEXTRA, TestBuilder/Testbench). Most researchers seem to be interested in applying their tools to more domain independent and/or harder tasks.

3.2.4 Semi-Automated Tool Methods

Computer-based tools are described below, indexed by type of method (an alphabetical, referenced catalog appears later). When specific tools implement these methods, the name of the tool is listed. Work describing methods not implemented as computer-based tools, and tools without names, are listed and referenced as a "Method." Many tools employ multiple methods and are indexed in several categories.

Psychology-Based and Interviewing Methods

Automated or Mixed-initiative Interviewing - *the tool interviews the expert*
AQUINAS, ARK, ASK, COGNOSYS, DART, EAR*, ELI, ETS, IRA-GRID, KAT (i), KITTEN, KNACK (i), KRIMB, KRITON, KSS0, MDIS, MOLE, MORE, NEXTRA, ODYSSEUS, PLANET, PROTOKI, PROTOS, ROGET, SALT, TEIRESIAS, TKAW/TDE
Methods - (Furuya and Hattori, 1989)

Protocol Analysis (Case Walk-Through / Eidetic Reduction / Observation / Process-Tracing) - *record and analyze transcripts from experts thinking aloud about tasks*
CODE, COGNOSYS, KRITON, LAPS, MACAO, MEDKAT

Psychological Scaling (including multidimensional scaling) - *use scaling techniques to help structure knowledge*

AQUINAS, DART, EAR*, IRA-GRID, FLEXIGRID, KITTEN, KRITON, KSS0, NEXTRA, PATHFINDER, PLANET, SMEE

Methods - (Adelman, 1989), (Butler and Corter, 1986), (Furuya and Hattori, 1989), (Gaines and Shaw, 1981)

Repertory Grids and PCP - *use personal construct psychology and related methods to elicit and analyze knowledge*

AQUINAS, AUTOINTELLIGENCE, CANARD, DART, EAR*, ETS, FLEXIGRID, IRA-GRID, KITTEN, KRITON, KSS0, NEXTRA, PLANET, SMEE

Methods - (Ford, Adams-Webber, Petry, and Chang, 1989), (Ford and Petry, 1989), (Furuya and Hattori, 1989), (Kelly, 1955)

Problem, Method, Performance Exploitation

Domain Problem Exploitation (Single Application) - *rely heavily on the domain for knowledge acquisition guidance*

ASKE, FIS, LEAP, MOLTKE, OPAL, PROTOGÉ, SMAC, STUDENT

Methods - (Boy and Caminel, 1989), (Lee, 1989)

Problem-Solving Method Exploitation - *use information about the problem-solving method to guide knowledge acquisition*

AQUINAS, ASKE, CANARD, CGEN, CLASSIKA, DACRON, DART, FMEASSIST, GEN-X, KARDIO, KAT (i), KAT (ii), KLAMSHELL, KNACK (i), KNACK (ii), LEDA, MDIS, MDX2, MOLE, MORE, SALT, SMAC, SMEE, TEIRESIAS, TKAW, VIEWPOINT

Methods - (Chu, 1989), (Furuya and Hattori, 1989)

Performance System (direct link or embedded) - *generate knowledge that may be directly tested by an embedded inference engine or by a separate shell*

APPRENTICE, AQUINAS, ASK, ASKE, BLIP, CGEN, DART, ETS, GEN-X, IVY, IRA-GRID, KAE, KAT (ii), KLAMSHELL, KNACK (i), KNACK (ii), KRITON, KSS0, LAPS, LEAP, MDIS, MOLE, MOLTKE, MORE, MUM, NEXTRA, ODYSSEUS, OPAL, PROTOGÉ, ROGET, TEIRESIAS, TESTBENCH, TESTBUILDER, TKAW, SALT

Methods - (Boy and Caminel, 1989), (Boy and Delail, 1988), (Brown, D., 1989), (Loftin, Saito, Wang, and Baffes, 1989)

Verification, Validation, and Maintenance - *verify and validate knowledge acquisition techniques; support testing, delivery, and maintenance of KBS's; help with necessity and sufficiency measuring for the knowledge-based system life cycle*

AQUINAS, TESTBED

Methods - (Baum, Shema, Boose, and Bradshaw, 1989). (Benbaset and Dhaliwal, 1988, 1989a,b), (Chadha, Mazlack, and Pick, 1989), (Dhaliwal and Benbaset, 1989), (Enand, Kahn, and Mills, 1988), (Gale, 1988), (Kellogg, Gargan, Mark, McGuire, Pontecorvo, Schlossberg, Sullivan, Genesereth, and Singh, 1989), (Lehner, 1989), (Levi, 1989), (Shadbolt and Burton, 1989), (Shaw, 1988), (Shaw and Woodward, 1987)

Modeling

Causal Modeling - *build cause/effect models of domain entities and relationships*

CMET, FIS, MOLE

Methods - (Bradshaw and Young, 1988)

Cognitive Modeling - *model thought process and human problem solving*

BDM-KAT, COGNOSYS, ONTOS

Methods - (Boy, 1989: situation recognition and reasoning patterns), (Boy and Caminel, 1989: situation recognition and reasoning patterns), (Boy and Delail, 1988: situation recognition and reasoning patterns), (Boy and Nuss, 1988: situation recognition and reasoning patterns), (LaFrance, 1989: cognitive modeling), (Nobel, 1989: cognitive modeling), (Rappaport, 1988: cognitive primitives), (Rouse, Hammer, and Lewis, 1989: signal and symbol processing), (Shalin, Bloomfield, and Bullemer, 1989: cognitive modeling), (Shaw and Woodward, 1989: mental models), (Woods and Hollnagel, 1987: cognitive modeling)

Conceptual Modeling - *build graphical or other multiple level models*

APPRENTICE (graphical planning models), BLIP (top-down, bottom-up modeling), GEKATOO (syntactic conceptual models), KET (graphical models), METAMATH (multiple level knowledge modeling), PROTOGÉ (conceptual models of knowledge acquisition tools), ROGET (conceptual domain structures), SEEGRAPH (graphic query), SORTAL (conceptual modeling)

Methods - (Addis, 1987, 1989a: theories, models, and representations), (Berg-Cross and Price, 1989: conceptual graphs), (Brulé, 1989: knowledge templates), (Eklund and Kellett, 1989: conceptual graphs), (Jansson and Rudström, 1989: semantic modeling), (Morik, 1987b: top-down, bottom-up modeling), (Pau and Nielsen, 1989: conceptual graphs), (Regoczei and Hirst, 1988, 1989a,b: conceptual modeling), (Regoczei and Plantinga, 1987: conceptual modeling), (Ribar, 1989: graphical interface for decision trees), (Twine, 1988, 1989: knowledge analysis model), (Yoshi, 1989: rule editor)

Consistency Analysis, Completeness Checking - *analyze knowledge for consistency or completeness*

ANALYZER, BLIP, FIS, KAT (ii), KNAC, KNACK (ii), LAPS, LÉZARD, MUM, QMOD, TEIRESIAS, VIEWPOINT

Methods - (Chen, 1989), (Polat and Güvenir, 1989)

Decision Analysis - *perform probabilistic inference and planning using influence diagrams and related techniques*

AXOTL, CANARD, INFORM

Methods - (Adelman, 1989), (Bradshaw and Boose, 1990), (von Winterfeldt and Edwards, 1986)

Domain Modeling - *build rich domain models*

3DKAT (design models), ASKE (domain exemplars), BLIP (domain modeling), CANARD (design alternatives and constraints), CGEN (specification models), FMEASSIST (failure modes and effects), INDE (deep modeling), KARDIO (diagnostic models), KAT (diagnostic models), KRIMB (diagnostic models), LÉZARD (deep modeling), ONTOS (domain and cognitive models), ROGET (conceptual domain structures), QUIZ (domain models from text), TESTBENCH (failure modes models), TESTBUILDER (failure modes models)

Methods - (Araya, 1988: diagnostic models), (Boose, Bradshaw, Shema, and Covington, 1989: design knowledge), (Friel, Lockledge, Hart, Mayer, Sanders, and Shashank, 1989: design knowledge), (Krishnamurthi and Underbrink, 1989: diagnostic modeling), (Linster, 1987: diagnostic models)

Ontological Modeling, Linguistic Modeling - *build language-based models*

ASTEK (ontology), CODE (concepts and terms), KADS (domain ontology), MAC (domain ontology)

Methods - (Addis and Bull, 1988: concept language), (Alexander, Freiling, Shulman, Rehfuss, and Messick, 1987: ontolgy), (Berry-Rogghe and Kaplan, 1986: text models), (Davis and Bonnell, 1989: knowledge dictionary), (Freiling and Jacobson, 1988, 1989: ontology), (Hayward, Wielinga, and Breuker, 1987: domain ontology), (Jacobson and Freiling, 1989: ontology), (Jansen and Compton, 1988, 1989: knowledge dictionary integration), (Regoczei and Plantinga, 1987), (Rochowiak and Mosley, 1989: knowledge dictionary), (Schreiber, Breuker, Bredeweg, and Wielinga, 1988: domain ontology)

Simulation - *run simulations to verify knowledge bases or produce rules*

3DKAT, MDIS, QUMAS, SIMULA

Methods - (Becker, Bartlett, and Soroushian, 1989: generate diagnostic rules from simulations)

Knowledge Browsers

Graphical Browsers - *provide knowledge browsers and editors as a primary elicitation tool*

APPRENTICE, CYC, GKE, KET, KREME, NEXPERT, PRED, SEEGRAPH

Hypermedia - *use hypermedia to capture and document knowledge*

ACQUIST, CAMEO, COGNOSYS, KAT (iii), KSS0, ONTOS, SMAC

Methods - (Boy, 1989), (Gaines and Sharp, 1987), (Kellogg, Gargan, Mark, McGuire, Pontecorvo, Schlossberg, Sullivan, Genesereth, and Singh, 1989), (Lee, 1989), (Lindsay, Cameron, Fugure, and Niem, 1988), (Rantanen, 1989), (Wells, 1989)

Multiple Experts

Delphi - *gather information from people independently*

MEDKAT

Multiple Source - *elicit and analyze knowledge from multiple sources separately and combine for use and further analysis*

ANALYZER, AQUINAS, CARTER, DART, ETS, GKE, MEDKAT, KITTEN, KSS0, NEXTRA

Methods - (Gaines, 1987a, 1987b), (LeClair, 1989), (Liou, Weber, and Nunamaker, 1989), (Loftin, Saito, Wang, and Baffes, 1989), (Mittal and Dym, 1985), (Wolf, 1989)

Other Sources of Knowledge

CAD Database, Database Management Systems - *acquire knowledge directly from computer-aided design data or other databases*

CMET

Methods - (Myler and Gonzalez, 1989)

Textual Analysis, Natural Language Analysis - *generate knowledge directly by analyzing text*

ACQUIST, KADS, KALEX, KRITON, KSS0, KBAM, LUKES, PETRARCA, PROPOS/EPISTOS, QUIZ, SORTAL, WASTL

Methods - (Allgayer, Harbusch, Kobsa, Reddig, Reithinger, and Schmauks, 1989), (Arinze, 1989), (Berry-Rogghe and Kaplan, 1986), (Cordingley, 1987), (De La Garza and Ibbs, 1989), (Fass, 1989), (Gomez, 1989), (Kornecki, 1989), (Kwasnik, 1989), (McHugh, 1987), (Slator, 1988, 1989), (Tsui, 1988), (Velardi, 1989)

3.3 Automated (Machine Learning) Tools and Techniques

3.3.1 Interactions Between Automated and Semi-Automated Methods

Automated (machine learning) methods typically apply algorithms to make generalizations or induce knowledge from examples. The interaction between interactive and machine learning methods is discussed in (Boose, Shema, and Bradshaw, 1989; Gaines, 1989b; Kodratoff, 1989). Several tools have combined interactive methods with machine learning methods (for example, AQUINAS, BLIP, DISCIPLE, ELI, IVY, KARDIO, KRITON, KSS0, LEDA, NEXTRA, ODYSSEUS, TEIRESIAS). Typically in such a combined system, information is elicited from the expert and analyzed. Then the tool generates information using a machine learning technique. The expert may use this information to further refine and debug the knowledge base, or the information might be used directly by an expert system "shell."

Gaines (1989a) has studied how the amount of knowledge available from the expert affects the amount of data required for effective empirical induction. He took a set of cases originally defined by Cendrowska (1987) as a test of empirical induction and generated a range of data sets from it by making incorrect decisions with known probability and adding a prescribed number of irrelevant attributes with random values. He then applied an empirical induction algorithm, INDUCT, to the data sets to determine the average amount of data required to obtain a correctly performing set of rules. The original data set can be correctly modeled with 6 rules involving 4 attributes.

Figure 8 shows some of Gaines' results as a plot of the data required on average for empirical induction against the type of knowledge available from the simulated expert. The trade off between data and expertise is clearly shown, and what is also apparent is the continuum between dependence on empirical induction from cases on the left and dependence on elicitation of expert knowledge on the right. The results validate the claims of those with tools aimed primarily at eliciting relevant attributes and critical cases from experts that this is an efficient approach. They also show that empirical induction and expertise elicitation can be combined to produce a more powerful and versatile acquisition tool than either alone. In the future methodologies and tools will increasingly support a combination of techniques.

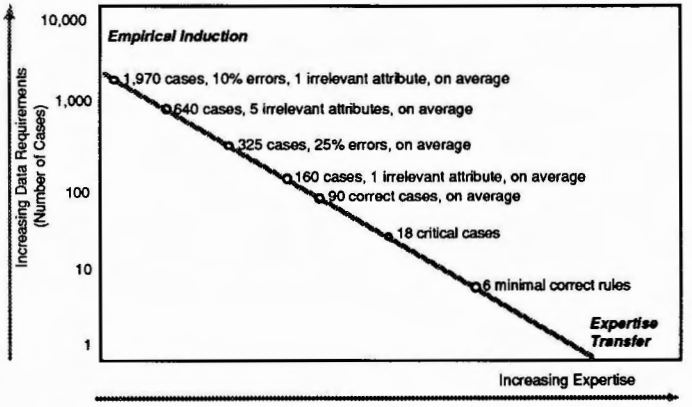


Figure 8. Knowledge/data tradeoff in expertise transfer and empirical induction (Gaines, 1989b).

3.3.2 Automated Tools and Techniques

Representative machine learning tools appear below, indexed by technique. Alternate classification schemes include Kodratoff's breakdown into specification-guided and example-guided deductive processes, and inductive learning from examples and observations (1989, adapted from Michalski, Carbonell, and Mitchell, 1983, 1986; Kodratoff and Michalski, 1989); a division into logical, functional, and procedural methods (MacDonald and Witten, 1989); Michalski's traditional classification of rote learning, learning from instruction, by deduction, analogy, example, and discovery (Michalski, Carbonell, and Mitchell, 1986); and a classification by representation generation agent, experience generation agent, and representation evaluator (Shalin, Wisniewski, Levi, and Scott, 1988). These schemes and others are detailed in Boose and Gaines (1990).

Analogy - *apply knowledge from old situations in similar new situations*
CYC, TEIRESIAS
Methods: (Adelson, 1989), (Leishman, 1989)

Apprenticeship Learning - *learn by watching experts solve problems*
CLINT, DISCIPLE, ISG, LEAP, LEDA, MALEFIZ, METAMOUSE, ODYSSEUS, PROTOS
Methods - (Yamada and Tsuji, 1989)

Case-Based Learning - *reason directly from a set of stored cases*
AQUINAS, CREEK, ELI, IVY, MOLTKE, PETRARCA
Methods - (Becker, 1988), (Chadha, Mazlack, and Pick, 1989), (Hardt, 1988)

Decision Tree Induction / Analysis; Question Scheduling - *generate, analyze decision trees*
CART, ID3, KATE
Methods - (Bramer, 1987), (Cox, 1988), (González and Alcaraz, 1989), (Goodman and Smyth, 1987a,b), (Mellis, 1989), (Merrem, 1989), (Pettit and Pettit, 1987), (Ruberg, Cornick, and James, 1988)

Discovery - *Learn by experimentation and observation*
EURISKO, LEX
Methods - (Thanassas, 1989)

Example Selection - *select an appropriate set of examples for various learning techniques*
Methods - (Blythe, Corsi, and Needham, 1987), (Rissland, 1987)

Explanation-Based Learning - *deduce a general rule from a single example by relating it to an existing theory*
ACES, EPSILON, INDE, IVY, LAS, LBUE, LEAP, OCCAM, ODYSSEUS, SRAR
Methods - (Kodratoff, 1987), (Levi, Perschbacher, and Shalin, 1988), (Zinßmeister and Hoppe, 1989)

Function Induction - *learn functions from input data*
BACON, MARVIN, NODDY

Genetic Algorithm - *genetic operators (crossing-over, mutation, inversion) are used to adapt a system's behavior*
Methods - (Pettit and Pettit, 1987)

Induction of Models from Experience
AM, ATOM

Neural Networks
Methods - (Daley, 1989), (Hayslip and Eilbert, 1989), (Oosthuizen, 1987), (Tani, Hirobe, Niida, Koshijima, and Murakami, 1989)

Performance Feedback - *performance feedback is used to reinforce behavior*
ACM, AQUINAS, CHECKERS, MOLE, PROTOS, STELLA

Rule / Knowledge Induction and Modification - *generate rules and other forms of knowledge*
AQ, AQUINAS, BLIP, CHARADE, EPSILON, ETS, KSS0, INDUCE, INDUCT, INSTIL, ISOLDE, ITRULE, KARDIO, KSS0, LEW, NEXTRA, PRISM, TLTS, VERSION SPACES
Methods - (Bergandano, Matwin, Michalski, and Zhang, 1989), (Buntine, 1987), (Cleary, 1987), (Delgrande, 1987), (Furuya and Hattori, 1989), (González and Alcaraz, 1989), (Goodman and Smyth, 1987b), (Handa and Ishizaki, 1988, 1989), (Liang, 1989), (MacDonald and Witten, 1989), (Maler, 1987), (Rissland, 1987), (Ruberg, Cornick, and James, 1988), (Sebag and Schoenauer, 1988), (Subramanian and Freuder, 1989), (Suwa and Motoda, 1989a,b), (Witten and MacDonald, 1988)

Similarity-Based Learning - *learn similarities from sets of positive examples and differences from sets of negative examples*
BLIP, GINESYS, ID3, ILROD, INC2, INDE, INSTIL
Methods - (Becker and Balasubramanian, 1989), (Matheus, 1989), (Schröder, Niemann, and Sagerer, 1988)

Systemic Principles Derivation - *use general principles to derive specific laws*
OBJ

4.0 Computer-Based Tools Catalog

The tools mentioned above are briefly described and referenced here.

3DKAT - *acquire models for design-oriented applications* (Dieng and Trousse, 1988)

ACES - *learn heuristics for fault diagnosis from device descriptions using explanation-based learning* (Pazzani, 1987)

ACQUIST - *use a hypertext-based tool to acquire knowledge for KEATS* (Motta, Eisenstadt, Pitman, and West, 1988), (Motta, Rajan, and Eisenstadt, 1988, 1989)

AM - *induce models from experience* (Davis and Lenat, 1982)

ANALYZER - *elicit software requirements through a system of distributed cooperating agents* (Easterbrook, 1989)

APPRENTICE - *use a graphical language to acquire knowledge about object manipulation for planning domains* (Joseph, 1989)

AQ - *induce rules from sets of positive and negative training examples* (Michalski, 1983)

AQUINAS - *elicit and model information using a knowledge acquisition workbench including hierarchically-structured repertory grid-based interviewing and testing and other methods* (Baum, Shema, Boose, and Bradshaw, 1989), (Boose, 1988), (Boose, 1989), (Boose and Bradshaw, 1987a,b), (Boose, Bradshaw, and Shema, 1988), (Boose, Bradshaw, Kitto, and Shema, 1989), (Boose, Shema, and Bradshaw, 1989), (Bradshaw and Boose, 1990), (Bradshaw, Covington, Russo, and Boose, 1989), (Kitto, 1988), (Kitto and Boose, 1987, 1989), (Schuler, Russo, Boose, and Bradshaw, 1988), (Shema and Boose, 1988), (Shema, Boose, and Bradshaw, 1988), (Shema, Bradshaw, Covington, and Boose, 1989)

ARK - *ask experts about the consequences of events* (Tonn, Arrowood, Goeltz, and Hake, 1989)

ASK - *acquire strategic knowledge from experts using a justification language* (Gruber, 1988a,b, 1989a,b,c), (Gruber and Cohen, 1989)

ASKE - *use exemplars from similar domains and task models to elicit problem solving expertise from the expert* (Patel, 1988, 1989)

ASTEK - *combine multiple paradigms for knowledge editing in a natural language discourse framework* (Jacobson and Freiling, 1988, 1989)

ATOM - *induce models from experience* (Gaines, 1977)

AXOTL - *use influence diagrams and other decision analysis techniques to represent process and probabilistic knowledge for problems with large amounts of uncertainty and complex tradeoffs* (Bradshaw, Boose, Covington, Russo, 1988), (Bradshaw, Covington, Russo, and Boose, 1989)

AUTOINTELLIGENCE - *elicit knowledge using repertory grid techniques* (commercial tool from IntelligenceWare, California)

BACON - *discover empirical scientific laws by inducing functions that account for observed behavior* (Langley, Bradshaw, and Simon, 1983), (Langley, Zytkow, Simon, and Bradshaw, 1986)

BDM-KAT - *provide an environment for knowledge engineers and experts to discuss, manipulate, and record objects and relationships* (Lancaster, Westphal, and McGraw, 1989), (McGraw, 1989), (McGraw and Lancaster, 1989)

BLIP - *construct organized domain models automatically by learning from sloppy models* (Kietz, 1988), (Morik, 1987a, 1989), (Wrobel, 1988)

CAMEO - *use hypertext to model and view expertise from multiple perspectives* (Jones, 1989)

CARTER - *recognize and repair discrepancies in knowledge bases from multiple experts* (Trice and Davis, 1989)

CANARD - *provide an exploratory environment for acquiring and narrowing design alternatives and constraints* (Shema, Bradshaw, Covington, and Boose, 1989)

CART - *employ cross-validation to produce appropriately-sized decision trees* (Crawford, 1989)

CGEN - *use specifications and generalization to design computers* (Birmingham, 1989), (Birmingham and Siewiorek, 1988)

CHARADE - *use inductive learning and empirical generalization for preventive maintenance* (Dupas and Millot, 1989)

CHECKERS - *learn variable weights by trial and error for playing checkers* (Samuel, 1963).

CLASSIKA - *use expert-directed techniques to capture aspects of classification problem-solving* (Gappa, 1988, 1989a,b), (Gappa and Puppe, 1989)

CLINT - *learn concepts from incomplete, weak examples* (De Raedt and Bruynooghe, 1989)

CMET - *acquire causal models from test and display databases* (Schaefer, 1989)

CODE - *perform conceptual and terminological analysis* (Skuce, 1989), (Skuce, Shenkang, Beauvillé, 1989)

COGNOSYS - *using a hypertext-based system acquire domain definition knowledge* (Woodward, 1988)

CREEK - *continuously update diagnostic knowledge bases using case-based learning* (Aamodt, 1989)

CYC - *acquire and use knowledge through the use of analogy and a large existing knowledge base* (Lenat, Prakash and Shepard, 1986)

DACRON - *acquire knowledge for situation based calculus planners* (Mahling and Croft, 1988)

DART - *NASA tool roughly based on Aquinas, tailored for engineering trade studies*, NASA Space Station Freedom Level 2 Program Office (Boose, Shema, and Bradshaw, 1990).

DISCIPLE - *integrate various machine learning techniques to adopt to available theories* (Kodratoff and Tecuci, 1988)

EAR* - *build knowledge-level structures for heuristic classification using repertory grids and interviewing methods* (Plaza and de Màntaras, 1989)

ELI - *interview experts and store knowledge in rule-oriented AND/OR graphs* (Silverman, Wenig, and Wu, 1989)

EPSILON - *perform knowledge acquisition by abductive and inductive explanation-based generalization* (Taki, 1989), (Taki and Fujii, 1989)

ETS - *interview experts using repertory grid-based methods and test the knowledge* (Boose, 1984, 1985, 1986a, b), (Boose, Bradshaw, Kitto, and Shema, 1989)

EURISKO - *learn heuristics and concepts by discovery* (Lenat, 1983)

FIS - *tie knowledge acquisition closely to the fault diagnosis domain* (De Jong, 1987)

FLEXIGRID - *use repertory grid techniques to elicit and analyze knowledge* (Tschudi, 1988)

FMEASSIST - *acquire failure characteristics for failure modes and effects analysis* (Carnes, 1989)

GEKATOO - *acquire conceptual models, checking for syntactic completeness* (Bonarini, Gallo, and Guida, 1989)

GENER - *acquire object hierarchies from experts and help generate if/then rules* (commercial tools from Cognisys; Mychaltchouk, Fayad, and Talaslian, 1989)

GEN-X - *acquire decision trees, AND/OR graphs, and if/then rule tables for diagnosis* (Crapo and Shah, 1989)

GINESYS - *use confirmation rules, a form of redundant knowledge, to learn in noisy domains* (Gams, 1988).

GKE - *decrease representational complexity by supporting multiple viewpoints and structured system design in a graphical framework* (Kontio and Lounamaa, 1989)

ID3 - *learn similarities and differences from training sets by optimizing global parameters* (Quinlan, 1983, 1987)

ILROD - *perform logic-based induction on Horn clauses to learn knowledge of relevance* (Dutta, 1988)

INC2 - *perform learning by observation using hill-climbing through a space of hierarchical classification schemes* (Hadzikadic, 1988)

INDE - *generate rules on the basis of counterexamples combining explanation-based learning and similarity-based learning* (Terpstra and van Someren, 1988)

INDUCE - *induce knowledge that mimics an expert's behavior* (Michalski and Chilausky, 1980)

INDUCT - *induce rules from a training set* (Gaines, 1989a,b), (Gaines and Rappaport, 1989)

INFORM - *elicit knowledge using decision analysis techniques* (Moore and Agogino, 1987)

INSTIL - *acquire knowledge using similarity-based learning combining aspects of both numeric and symbolic approaches* (Kodratoff and Manago, 1987a)

IRA-GRID - *use repertory grids to acquire knowledge for general classification problems* (Linster, 1989a)

ISG - *link evidence to situations by synthesizing rules from interesting situations using an apprenticeship learning approach* (Wisniewski, Winston, Smith, and Kleyn, 1987)

ISOLDE - *learn organic chemistry through induction* (Rose and Gelernter, 1989)

ITRULE - *induce rules using an information-theoretic metric* (Goodman and Smyth, 1987a)

IVY - *find gaps in a current knowledge base and formulate plans to fill the gaps* (Hunter, 1988)

KADS - *elicit and model knowledge decoupled from the design and implementation of the system* (Anjewierden, 1987), (Breuker and Wielinga, 1987a,b), (de Greef and Breuker, 1989), (Jansen-Winkeln, 1988), (Konrad and Tong, 1989), (Karbach, Voß, and Tong, 1988), (Konrad and Tong, 1989), (Schreiber, Breuker, Bredeweg, and Wielinga, 1988), (Tong, 1989), (Tong and Karbach, 1988), (Valtorta, 1989), (Wielinga, Akkermans, Schreiber, and Balder, 1989)

KAE - *capture scene analysis expertise* (Tranowski, Levitt, and Riley, 1988)

KALEX - *translate sentences from experts into an internal representation* (Schmidt and Wetter, 1989)

KARDIO - *generate diagnostic rules from models* (Lavrac and Mozetic, 1989)

KAT (i) - *acquire knowledge category information for diagnosis* (D'Ambrosio, 1989)

KAT (ii) - *acquire diagnostic information for a diagnostic refinement language* (Bansal and Posco, 1989)

KAT (iii) - *acquire knowledge for LEVEL5 using a hypertext editor* (commercial tool from Ribar, NY)

KATE - *given examples of an expert's diagnosis and background domain knowledge, generate a decision tree and rules* (Rouveirol and Manago, 1988)

KBAM - *use natural language explanations to construct a domain-specific knowledge base* (Silvestro, 1988)

KET - *provide a graphical interface and analyze relationships to help experts write rules* (Esfahani and Teskey, 1987, 1988)

KITTEN - *interview experts using repertory grid-based methods* (Shaw and Gaines, 1987b), (Shaw and Woodward, 1988)

KLAMSHELL - *acquire decision trees for maintenance and troubleshooting* (Cochran, 1988)

KNAC - *use acquired assimilation knowledge to help enter new knowledge in a knowledge base* (Lefkowitz and Lesser, 1988)

KNACK (i) - *elicit and use knowledge about evaluation report generation* (Kitto, 1988), (Klinker, 1989), (Klinker, Bentolila, Genetet, Grimes, and McDermott, 1987), (Klinker, Boyd, Dong, Maiman, McDermott, and Schnelback, 1989), (Klinker, Genetet, and McDermott, 1988)

KNACK (ii) - *acquire information from knowledge engineers and experts, allow browsing and editing for classification, diagnosis and repair* (Hsieh, Hayes, Hofmann, Patrick, and Mallory, 1988)

KREME - *include multiple-representations in a knowledge editing environment* (Abrett and Burstein, 1987)

KRIMB - *interview experts and build diagnostic domain models* (Cox and Blumenthal, 1987)

KRITON - *combine repertory grid interviewing and protocol analysis to build knowledge at an intermediate level* (Diederich, Linster, Ruhmann, and Uthmann, 1987), (Diederich, Ruhmann, and May, 1987), (Linster, 1988a,b, 1989, a,b)

KSS0 - *elicit knowledge with a repertory grid-based interviewing tool including text analysis, behavior induction, and psychological scaling techniques* (Gaines, 1987a,b, 1988a,b), (Gaines and Rappaport, 1989), (Gaines, Rappaport, and Shaw, 1989), (Gaines and Sharp, 1987), (Shaw, 1988, 1989a,b), (Shaw and Gaines, 1987a, 1988, 1989)

LAPS - *interweave protocol analysis with completeness querying* (di Piazza, 1988), (di Piazza and Helsabeck, 1989)

LAS - *use apprenticeship learning to learn by watching experts solve problems* (Smith, Winston, Mitchell, and Buchanan, 1985)

LBUE - *combine explanation-based learning and causal model learning* (Martin and Redmond, 1989)

LEAP - *use apprenticeship learning to learn steps in VLSI design by watching experts solve problems* (Mitchell, Mahadevan, and Steinberg, 1985), (Smith, Winston, Mitchell, and Buchanan, 1985)

LEDA - *acquire knowledge for chip architecture design by interactively generalizing design plans* (Hermann, 1989), (Herrmann and Franzke, 1988)

LEW - *learn inductively from examples of problem-solution or question-answer pairs* (Constant, Matwin, and Oppacher, 1988), (Matwin and Oppacher, 1989)

LEX - *acquire problem solving heuristics for symbolic integration* (Mitchell, Utgoff, and Banerji, 1982)

LÉZARD - *find ambiguities and incompleteness from deep knowledge and ask the expert about them* (Charlet and Gascuel, 1989)

LUKES - *perform natural language understanding and knowledge acquisition based on concept cluster attachment* (Regoczei and Hirst, 1989a)

MAC - *use a model-directed approach to acquire inference structures and successive refinements processes within the KADS framework* (de Greef and Breuker, 1989), (Konrad and Tong, 1989), (Tong, 1989)

MACAO - *model expert knowledge based on empirical and conceptual schemes* (Aussenac, Frontin, Riviere, and Soubie, 1989), (Aussenac, Frontin, and Soubie, 1988)

MALEFIZ - *use apprenticeship learning to acquire geometric knowledge for a complex design task* (Herrmann and Beckmann, 1989)

MARVIN - *learn hierarchical structures of concepts or functions* (Sammut and Banerji, 1983, 1986)

MDIS - *interview experts to elicit mechanisms in a top-down structured manner for diagnostic problems* (Antonelli, 1983)

MDX2 - *exploit diagnostic problem-solving for clinical medicine* (Sticklen, Chandrasekaran, and Bond, 1989)

MEDKAT - *automate the Delphi technique to gather information from multiple experts* (Jagannathan and Elmaghraby, 1985).

METAMATH - *perform multiple level knowledge modeling for knowledge engineering* (Nordbø, Vestli, and Sølvberg, 1989)

METAMOUSE - *learn procedural descriptions of graphical concepts, such as object structures, spatial relations and transformations* (Maulsby, James, and Witten, 1989), (Maulsby and Witten, 1988)

MOLE - *exploit information about how problems are solved to elicit scarce diagnostic knowledge and use feedback to fine tune the knowledge* (Eshelman, 1988), (Eshelman, Ehret, McDermott, and Tan, 1987), (Green and Eshelman, 1989)

MOLTKE - *acquire knowledge from diagnostic cases for computerized numerical control applications* (Althoff, Kockskämper, Traphöner, Wernicke, and Faupel, 1989)

MORE - *exploit information about how problems are solved to elicit extensive diagnostic knowledge* (Kahn, Nowlan, McDermott, 1985a,b)

MUM - *evidential combination knowledge and control knowledge are elicited for medical problems* (Gruber and Cohen, 1987)

NEXTRA - *commercial version of KSSO*, (from Neuron Data, California).(Rappaport and Gaines, 1988)

NODDY - *induce functions from traces of robot program executions* (Andreae, 1984a,b)

OBJ - *use general principles to derive specific laws* (Goguen and Meseguer, 1983)

OCCAM - *learn to predict outcomes of economic sanction episodes using explanation-based learning* (Pazzani, 1987)

ODYSSEUS - *refine and debug knowledge using apprenticeship learning techniques* (Chachere, 1989), (Wilkens, Clancey, and Buchanan, 1987)

ONTOS - *build domain models using cognitive and linguistic factors* (Monarch, Kaufmann, and Subramanian, 1989), (Monarch and Nirenburg, 1987), (Nirenburg, Monarch, Kaufmann, Nirenburg, and Carbonell, 1988)

OPAL - *tie knowledge acquisition closely to the cancer treatment domain* (Combs, Tu, Musen, and Fagan, 1988), (Musen, 1988a,b), (Musen, Fagan, Combs, and Shortliffe, 1987)

PATHFINDER - *use psychological scaling techniques to help structure knowledge hierarchically* (Cooke and McDonald, 1987)

PETRARCA - *acquire a case-based semantic dictionary from text* (Velardi, Pazienza, and Magrini, 1989)

PLANET - *use repertory grids for psychological interviewing and analysis* (Shaw, 1984), (Gaines and Shaw, 1986)

PM - *acquire a library of plans for programming using a metrics-driven approach* (Reynolds, Maletic, and Porvin, 1989)

PRED - *help experts create frame-based windowed knowledge acquisition interfaces* (Xie, Dumaresq, and Winne, 1988)

PRISM - *induce rules from a training set* (Cendrowska, 1987)

PROPOS/EPISTOS - *transform text into a meaning representation and then perform epistemological analysis using pragmatic fields* (Möller, 1988)

PROTOGÉ - *develop and edit the conceptual model of another knowledge acquisition tool (such as OPAL) for skeletal plan refinement tasks* (Musen, 1988a,b, 1989a,b,c)

PROTOKI - *critique new knowledge and integrate it into an existing knowledge base* (Murray, 1989), (Murray and Porter, 1988, 1989)

QMOD - *represent hypotheses graphically for consistency checking* (Feldman, Compton, Smythe, 1989)

QUIZ - *semi-automatically process technical text and incrementally produce a domain conceptual model* (Szpakowicz, 1988)

QUMAS - *run simulations on partially supplied models and learn new parts of the model* (Lavrac and Mozetic, 1989)

ROGET - *interview experts and produce conceptual structures of the domain* (Bennet, 1985)

SALT - *elicit and deliver knowledge for constructive constraint satisfaction tasks* (Marcus, 1987, 1988a,b, 1989), (Marcus, McDermott, and Wang, 1985), (Stout, Caplain, Marcus, and McDermott, 1988)

SEEGRAPH - *use a graphic query language to manipulate and store information in a network database* (Kopec and Latour, 1989)

SIMULA - *use basic laws to derive physical models through simulation* (Nygaard and Dahl, 1981)

SMAC - *use hypercards to allow entry of domain and problem-solving knowledge* (objects and links) (Moulin, 1989)

SMEE - *combine repertory grid methods and Newell and Simon's problem space concept for manufacturing problems* (Garg-Janardan, 1988), (Garg-Janardan and Salvendy, 1987)

SOAR - *learn for planning and problem solving, producing probability distributions and operator applicability conditions* (Laird, Rosenbloom, and Newell, 1986)

SORTAL - *perform meaning-triangle-based sortal analysis* (Regoczei and Hirst, 1989a,b)

SRAR - *use explanation-based learning techniques to develop intelligent tutoring systems* (Boy and Nuss, 1988).

STELLA - *performance feedback is used to reinforce behavior* (Gaines and Andreae, 1966)

STRIPS - *learn macro functions for planning and problem solving* (Fikes, Hart, and Nilsson, 1972)

STUDENT - *tie knowledge acquisition closely to the statistical consulting domain* (Gale, 1987)

TEIRESIAS - *model existing knowledge to monitor refinements and help debug consultations* (Davis and Lenat, 1982)

TESTBENCH, TESTBUILDER - *acquire knowledge for fault tree diagnosis* (commercial tool from Carnegie Group, Inc.) (Enand, Kahn, and Mills, 1988), (Lewis and Kahn, 1988)

TKAW/TDE - *exploit information about how problems are solved to elicit trouble-shooting knowledge* (Kahn, Breaux, Joeseph, and DeKlerk, 1987)

TLTS - *learn new knowledge sources for a blackboard system using redesign knowledge* (Simoudis, 1988, 1989)

VERSION SPACES - *find all descriptions that are consistent with a set of positive and negative examples* (Mitchell, 1982)

VIEWPOINT - *acquire troubleshooting knowledge* (Caviedes and Reed, 1989)

5.0 References

Aamodt, A. (1989). Towards robust expert systems that learning from experience - an architectural framework, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.311-326.

Abrett, G., Burstein, M. H. (1987).The KREME Knowledge Editing Environment, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 4, *International Journal of Man-Machine Studies*, Vol. 27, No. 2, pp. 103-126; also in Boose, J. H., and Gaines, B. R. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 1-24.

Addis, T. R. (1987). A framework for knowledge elicitation, *Proceedings of the first European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, U.K., September, pp. A1.1-16.

Addis, T.R. (1989a). Knowledge for Design, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 1.1-12.

Addis, T.R. (1989b). The Science of Knowledge: A Research Program for Knowledge Engineering, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.49-59.

Addis, T. R., Bull, S. P. (1988). A Concept Language for Knowledge Elicitation, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. A1.1-16.

Adelman, L. (1989). Measurement Issues in Knowledge Engineering, special issue of *IEEE Transactions on System, Man, and Cybernetics*, Vol. 19, No. 3, pp. 483-488.

Adelson, B. (1989). Learning by Analogy: A Cognitive Theory, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 23-27.

Alexander, J. H., Freiling, M. J., Shulman, S. J., Rehfuss, S., Messick, S. L. (1987). Ontological Analysis: An Ongoing Experiment, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 3, *International Journal of Man-Machine Studies*, Vol. 26, No.4, pp. 473-486; also in Boose, J. H., and Gaines, B. R. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 25-38.

Allgayer, J. Harbusch, K., Kobsa, A., Reddig, C., Reithinger, N., Schmauks, D. (1989). XTRA: a natural-language access system to expert systems, *International Journal of Man-Machine Studies*, Vol. 31, pp. 161-196.

Althoff, K. D., Kockskamper, S., Traphöner, R., Wernicke, W., Faupel, B. (1989). Knowledge Acquisition in the Domain of CNC Machining Centers: the MOLTKE Approach, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.180-195.

Andreae, P. M. (1984a). Constraint limited generalization: acquiring procedures from examples, *Proceedings of the National Conference on Artificial Intelligence* (AAAI-84), Austin, TX, June.

Andreae, P. M. (1984b). Justified generalization: Acquiring procedures from examples, PhD Thesis, Department of Electrical Engineering and Computer Science, MIT.

Anjewierden, A. (1987). The KADS System, *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, September, pp. E2.1-12.

Antonelli, D. (1983). The Application of Artificial Intelligence to a Maintenance and Diagnostic Information System (MDIS), in the *Proceedings of the Joint Services Workshop on Artificial Intelligence in Maintenance*, Boulder, CO.

Araya, A. (1988). Acquisition of generic knowledge in the context of a performance system, *Proceedings of the AAAI-88 Integration of Knowledge Acquisition and Performance Systems Workshop*, St. Paul, August, npp.

Arinze, B. (1989). A natural language front-end for knowledge acquisition, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 106-114.

Aussenac, N., Frontin, J., Riviere, M. H., Soubie, J. L. (1989). A Mediating Representation to assist Knowledge acquisition with MACAO, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.516-529; and *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 2.1-17.

Aussenac, N., Frontin, J., Soubie, J.-L. (1988). Macao: A Knowledge Acquisition Tool for Expertise Transfer, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 8.1-12.

Bansal, R., Posco, P. (1989). Knowledge Acquisition Tool Promulgating Expert Systems in a Manufacturing Environment, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 129-132.

Bareiss, E. R. (1988). Protos: A unified approach to concept representation, classification, and learning, Technical Report AI88-83, AI Laboratory, Univ. of Texas, Austin, August.

Bareiss, E. R. (1989a). *Exemplar-based Knowledge Acquisition: A Unified Approach to Concept Representation, Classification, and learning*. Academic Press.

Bareiss, E. R. (1989b). Experimental Evaluation of the PROTOS Knowledge Acquisition System, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp.186-188, and *IJCAI-89 Workshop on Knowledge Acquisition: Practical Tools and Techniques*, Detroit, August, npp.

Bareiss, E. R., Porter, B. W., Wier, C. C. (1988). Protos: An Exemplar-Based Learning Apprentice, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, Vol. 29, No. 5, pp. 549-562.

Baum, L.S., Shema, D.B., Boose, J.H., Bradshaw, J.M. (1989). Acquiring and Verifying Control Knowledge for a Blackboard System, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 3.1-15.

Becker, B. (1988). Towards a Case-Oriented Concept for Knowledge Acquisition, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, p.9.1-12.

Becker, L.A., Balasubramanian, N. (1989). An Efficient Knowledge Acquisition Methodology for Process Planning Expert Systems, *IJCAI-89 Workshop on Knowledge Acquisition: Practical Tools and Techniques*, Detroit, August, npp.

Becker, L.A., Bartlett, R., Soroushian, F. (1989). Using simulation to compile diagnostic rules from a manufacturing process representation, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 172-173.

Belkin, N. J., Brooks, H. M., Daniels, P. J. (1987). Knowledge Elicitation Using Discourse Analysis, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 4, *International Journal of Man-Machine Studies*, Vol. 27, No. 2, pp. 127-144; also in Gaines, B. R., and Boose, J. H. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 107-124.

Benbasat, I., Dhaliwal, J.S. (1988). A framework for the validation of knowledge acquisition. Boose, J.H. & Gaines, B.R. (Eds) *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*. pp.1-1-1-18. Banff (November).

Benbaset, I., Dhaliwal, J.S. (1989a). A framework for the validation of knowledge acquisition, *Knowledge Acquisition: An International Journal*, Vol. 1, No. 2, pp. 215-233.

Benbasat, I., Dhaliwal, J. S. (1989b). The Validation of Knowledge Acquisition: Methodology and Techniques, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, pp.60-74.

Benfer, R. A., Furbee, L. (1989). Knowledge acquisition in the Peruvian Andes, *AI Expert*, November, pp. 22-29.

Bennet, J. S. (1985). A Knowledge-Based System for Acquiring the Conceptual Structure of a Diagnostic Expert System, *Journal of Automated Reasoning*, 1, pp. 49-74.

Berg-Cross, G., Price, M.E. (1989). Acquiring and Managing Knowledge Using a Conceptual Structures Approach: Introduction and Framework, special issue of *IEEE Transactions on System, Man, and Cybernetics*, Vol. 19, No. 3, pp. 513-527.

Bergadano, F., Matwin, S., Michalski, R.S., Zhang, J. (1989). Learning Flexible Concepts through a Search for Simpler but Still Accurate Descriptions, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 4.1-10.

Berry-Rogghe, G., and Kaplan, R. M. (1986). On the structure of expository text: Preliminaries in building an expert system from manuals, *Proceedings of the First Knowledge Acquisition for Knowledge-Based System Workshop*, Banff, November, pp. 4.0-7.

Beyth-Marom, R. and Dekel, S. (1985). *An Elementary Approach to Thinking Under Uncertainty*, translated and adapted by S. Lichtenstein, B. Marom, and R. Beyth-Marom, Hillsdale, NJ.: Lawrence Erlbaum.

Birmingham, W. P., Siewiorek, D. P. (1988). Automated Knowledge Acquisition for a Computer Hardware Synthesis System, *Proceedings of the Third Knowledge Acquisition for Knowledge-*

Based Systems Workshop, Banff, November, pp. 2.1-20.

Birmingham, W. P. (1989). Automated Knowledge Acquisition for Computer-Aided Design Tools, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp.101-104.

Blythe, J., Corsi, P., and Needham, D. (1987). An Experimental Protocol for the Acquisition of Examples for Learning, *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, September, pp. F1.1-13.

Bonarini, A., Gallo, M. C., Guida, M. (1989). GEKATOO, A General Support Tool for Knowledge Acquisition, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 28-33.

Boose, J. H. (1984). Personal Construct Theory and the Transfer of Human Expertise, in the *Proceedings of the National Conference on Artificial Intelligence (AAAI-84)*, Austin, Texas, pp. 27-33.

Boose, J. H. (1985). A Knowledge Acquisition Program for Expert Systems Based On Personal Construct Psychology, *International Journal of Man-Machine Studies*, Vol. 23, p.495-525.

Boose, J. H. (1986a). *Expertise Transfer for Expert System Design*, New York: Elsevier.

Boose, J. H. (1986b). Rapid Acquisition and Combination of Knowledge from Multiple Experts in the Same Domain, *Future Computing Systems Journal*, Vol. 1, No. 2, pp. 191-216.

Boose, J. H. (1988). Uses of Repertory Grid-Centred Knowledge Acquisition Tools for Knowledge-Based Systems, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, Vol. 29, No. 3, pp. 287-310.

Boose, J. H. (1989). Design Knowledge Capture for a Corporate Memory Facility, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 5-6.

Boose, J. H., Bradshaw, J. M. (1987a). Expertise Transfer and Complex Problems: Using Aquinas as a Knowledge Acquisition Workbench for Expert Systems, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 1, *International Journal of Man-Machine Studies*, Vol. 26, No.1, p.3-28; also in Boose, J. H., and Gaines, B. R. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 39-64.

Boose, J. H., Bradshaw, J. M. (1987b). AQUINAS: A Knowledge Acquisition Workbench for Building Knowledge-Based Systems, *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, September, pp. A6.1-6.

Boose, J. H., Bradshaw, J. M., Kitto, C. M., Shema, D. B. (1989). From ETS to Aquinas: Six years of knowledge acquisition tool development, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, pp.502-516; and *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 5.1-17.

Boose, J. H., Bradshaw, J. M., Shema, D. B. (1988). Recent Progress in Aquinas; A Knowledge Acquisition Workbench, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 2.1-15.

Boose, J. H., Bradshaw, J.M., Shema, D.B., Covington, S.P. (1989). Design Knowledge Capture for a Corporate Memory Facility, *IJCAI-89 Workshop on Knowledge Acquisition: Practical Tools and Techniques*, Detroit, August, 1989; *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 5-6.

Boose, J. H., and Gaines, B. R. (1990). Knowledge Acquisition for Knowledge-Based Systems, Tutorial Notes, *Proceedings of the Artificial Intelligence Systems in Government*, Wash., DC, May, 1990.

Boose, J. H., Shema, D.B., and Bradshaw, J.M. (1989). Recent progress in Aquinas: A knowledge acquisition workbench, *Knowledge Acquisition: An International Journal*, Vol. 1, No. 2, pp. 185-214.

Boose, J. H., Shema, D.B., and Bradshaw, J.M. (1989). Design Knowledge Capture for a Corporate Memory Facility, *5th Annual AI in Space Conference*, Huntsville, May, 1990.

Boy, G. (1989). The Block Representation in Knowledge Acquisition for Computer Integrated Documentation, *IJCAI-89 Workshop on Knowledge Acquisition: Practical Tools and Techniques*, Detroit, August, 1989, and *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 6.1-26.

Boy, G., Caminel, T. (1989). Situations Patterns Acquisition. Improves the Control of Complex Dynamic Systems, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.160-168.

Boy, G. A., Delail, M. (1988). Knowledge acquisition by specialization / structuration: A space telemanipulation application, *Proceedings of the AAAI-88 Integration of Knowledge Acquisition and Performance Systems Workshop*, St. Paul, August, npp.

Boy, G., Nuss, N. (1988). Knowledge Acquisition by Observation: Application to Intelligent Tutoring Systems, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 11.1-14.

Bradley, J., Harbison-Briggs, K. (1989). The symptom-component approach to knowledge acquisition, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 70-76.

Bradshaw, J., Young, R. (1988). Shared Causal Knowledge as a Basis for Communication Between Expert and Knowledge Acquisition System, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 12.1-6.

Bradshaw, J. M. (1991). Strategies for Selecting and Interviewing Experts, Boeing Computer Services Technical Report, in preparation.

Bradshaw, J. M., Boose, J. H. (1990). Decision Analysis Techniques for Knowledge Acquisition: Combining Situation and Preference Models Using Aquinas and Axotl, *International Journal of Man-Machine Studies*, Vol. 32, No. 2, pp. 121-186.

Bradshaw, J. M., Boose, J. H., Covington, S. P., and Russo, P. J. (1989). How to do with grids what people say you can't: The application of decision analysis methods in Axotl and personal construct methods in Aquinas to design problems, *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 4.1-15; *Knowledge Acquisition Journal*, [in press].

Bradshaw, J. M., Russo, P. J., Covington, S. P., Boose, J. H. (1989). Knowledge acquisition and decision analysis; La folie a deux, *Proceedings of the AAAI Uncertainty Workshop*, Detroit, August.

Bradshaw, J. M., Covington, S. P., Russo, P. J., and Boose, J. H. (1990). KWACQ'ing with DDUCKS: Knowledge Acquisition Techniques for Decision Analysis, Using Axotl and Aquinas, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 7.1-12; *Knowledge Acquisition Journal*, [in press].

Bramer, M. (1987). Automatic Induction of Rules from Examples: A Critical Analysis of the ID3 Family of Rule Induction Systems, *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, September, pp. F2.1-31.

Breuker, J., Wielinga, B. (1987a). Knowledge Acquisition as Modeling Expertise: The KADS Methodology, *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, September, pp. B1.1-8.

Breuker, J., Wielinga, B. (1987b). Use of Models in the Interpretation of Verbal Data, in A. Kidd, *Knowledge Elicitation for Expert Systems: A Practical Handbook*, Plenum Press.

Brown, B. (1989). The taming of an expert: An anecdotal report, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 133-135.

Brown, D. (1989). A knowledge acquisition tool for decision support systems. *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 93-97.

Brulé, J. F. (1989). Knowledge Templates: A Knowledge Acquisition Tool with Organizational Significance, *IJCAI-89 Workshop on Knowledge Acquisition: Practical Tools and Techniques*, Detroit, August, npp.

Buntine, W. (1987). Induction of Horn Clauses: Methods and the Plausible Generalization Algorithm, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 3, *International Journal of Man-Machine Studies*, Vol. 26, No.4, pp. 499-520; also in Gaines, B. R., and Boose, J. H. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 275-298.

Burton, A. M., Shadbolt, N. R., Hedgecock, A. P., Rugg, G. (1987). A Formal Evaluation of Knowledge Elicitation Techniques for Expert Systems: Domain 1, *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, September, pp. D3.1-21.

Butler, K. A., Corter, J. E. (1986). Use of Psychometric Tools for Knowledge Acquisition: A Case Study, in W. A. Gale, *Artificial Intelligence and Statistics*, Menlo Park: Addison-Wesley, pp. 295-320.

Bylander, T., and Mittal, S. (1986). CSRL: A Language for Classificatory Problem-Solving and Uncertainty Handling, *AI Magazine*, August.

Bylander, T., Chandrasekaran, B. (1987). Generic Tasks in Knowledge-Based Reasoning: The 'Right' Level of Abstraction for Knowledge Acquisition, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems

Workshop, 1986, Part 2, *International Journal of Man-Machine Studies*, Vol. 26, No.2, pp. 231-244; also in Gaines, B. R., and Boose, J. H. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 65-80.

Carnes, J.R. (1989). FMEAssist: A Tool for Acquisition, Engineering, and Operations Using Failure Knowledge, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 151-153.

Caviedes, J. E., Reed, M. K. (1989). ViewPoint: A troubleshooting-specific knowledge acquisition tool, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 155.

Cendrowska, J. (1987) An algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27 (4), 349-370 (October).

Chachere, L. F. (1989). A Knowledge-based system that supports explanation-based learning applications, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.327-337.

Chadha, S.R., Mazlack, L.J., Pick, R.A. (1989). Acquiring and Validating Knowledge for an Expert System Using A Case-Based Approach, *IJCAI-89 Workshop on Knowledge Acquisition: Practical Tools and Techniques*, Detroit, August, npp.

Chandrasekaran, B. (1988). Generic tasks as building blocks for knowledge-based systems: the diagnosis and routine design examples, *The Knowledge Engineering Review*, 3(3), pp. 183-211.

Charlet, J., Gascuel, O. (1989). Knowledge Acquisition by Causal Model and Meta-Knowledge, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.212-226.

Chen, Z. (1989). Integrating Knowledge Pieces in Knowledge Acquisition, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 149-150.

Chu, W.-H. (1989). Knowledge Acquisition for Diagnostic Reasoning, *IJCAI-89 Workshop on Knowledge Acquisition: Practical Tools and Techniques*, Detroit, August, npp.

Clancey, W. (1986a). Heuristic Classification, in J. Kowalik (ed.), *Knowledge-Based Problem-Solving*, New York: Prentice-Hall.

Clancey, W. (1986b). Knowledge acquisition, presentation at the First Knowledge Acquisiton for Knowledge-Based Systems Workshop, Banff, October.

Clarke, B. (1987). Knowledge acquisition for real-time knowledge-based systems, *Proceedings of the first European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, U.K., September, pp. C2.1-7.

Cleary, J. G. (1987). Acquisition of Uncertain Rules in a Probabilistic Logic, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 4, *International Journal of Man-Machine Studies*, Vol. 27, No. 2, pp. 145-154; also in Gaines, B. R., and Boose, J. H. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 323-332.

Cleaves, D. A. (1987). Cognitive Biases and Corrective Techniques; Proposals for Improving Elicitation Procedures for Knowledge-Based Systems, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 4, *International Journal of Man-Machine Studies*, Vol. 27, No. 2, pp. 155-156; also in Gaines, B. R., and Boose, J. H. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 23-34.

Cochran, E. L. (1988). KLAMShell™: A domain-specific knowledge acquisition shell, *Proceedings of the AAAI-88 Integration of Knowledge Acquisition and Performance Systems Workshop*, St. Paul, August, npp.

Combs, D. M., Tu, S. W., Musen, M. A., Fagan, L. M. (1988). From expert models to expert systems: Translation of an intermediate knowledge representation, *Proceedings of the AAAI-88 Integration of Knowledge Acquisition and Performance Systems Workshop*, St. Paul, August, npp.

Constant, P., Matwin, S., Oppacher, F. (1988). LEW: Learning by Watching, Technical Report, Department of Computer Science, George Mason Univ., July.

Cooke, N. M., McDonald, J. E. (1987). The Application of Psychological Scaling Techniques to Knowledge Elicitation for Knowledge-Based Systems, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 3, *International Journal of Man-Machine Studies*, Vol. 26, No.4, pp. 533-549; also in Boose, J. H., and Gaines, B. R. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 65-82.

Cordingley, E. S. (1987). Knowledge acquisition for the Advice System, *Proceedings of the first European Workshop on Knowledge Acquisition for*

Knowledge-Based Systems, Reading University, U.K., September, pp. C3.1-21.

Cox, L. A. (1988). Designing Interactive Expert Classification System that Acquires Knowledge 'Optimally,' *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 13.1-16.

Cox, L. A., Blumenthal, R. (1987). KRIMB: An Intelligent Knowledge Acquisition and Representation Program for Interactive Model Building, *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, September, pp. E3.1-17.

Crandall, B.W. (1989). A comparative study of think-aloud and critical decision knowledge elicitation methods, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 144-146.

Crapo, A., Shah, R. (1989). GEN-X: A Graphical Tool for Knowledge Acquisition, *IJCAI-89 Workshop on Knowledge Acquisition: Practical Tools and Techniques*, Detroit, August, npp.

Crawford, S. L. (1989). Extensions to the CART Algorithm, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, Vol. 31, pp. 197-218.

D'Ambrosio, A. (1989). Acquiring Knowledge of Real-World Domains: Is the Paradigm the Key? *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 120-125.

Daley, P. C. (1989). Using Neural Nets to Automate Knowledge Engineering, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 1-4.

Davis, J.P., Bonnell, R.D. (1989). Concept Abstraction and the Knowledge Dictionary: A Framework for Acquisition and Maintenance of Knowledge Schemata, *IJCAI-89 Workshop on Knowledge Acquisition: Practical Tools and Techniques*, Detroit, August, npp.

Davis, R., Lenat, D. B. (1982). *Knowledge-Based Systems in Artificial Intelligence*, New York: McGraw-Hill.

De Greef, P., Breuker, J. (1989). A Methodology for Analysing Modalities of System/User Cooperation for KBS, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.462-473.

De Jong, K. (1987). Knowledge Acquisition for Fault Isolation Expert Systems, special issue on the 1st AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 4, *International Journal of Man-Machine Studies*, Vol. 27, No. 2.

De Raedt, L., Bruynooghe, M. (1989). On Explanation and Bias in Inductive Concept-Learning, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.338-353.

De la Garza, J., Ibbs, C.W. (1989). A Limited Information Knowledge Elicitation Experiment, *IJCAI-89 Workshop on Knowledge Acquisition: Practical Tools and Techniques*, Detroit, August, npp.

Dean, T., Kanazawa, K. (1989). Persistence and Probabilistic Projection, special issue of *IEEE Transactions on System, Man, and Cybernetics*, Vol. 19, No. 3, pp. 574-585.

Delgrande, J. P. (1987). A Formal Approach to Learning From Examples, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 2, *International Journal of Man-Machine Studies*, Vol. 26, No.2, pp. 123-142; also in Gaines, B. R., and Boose, J. H. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 163-182.

di Piazza, J. S. (1988). Laps: An Assistant for Debriefing Experts, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 14.1-17.

di Piazza, J.S., Helsabeck, F. A. (1989). LAPS: From Cases to Models to Complete Expert Systems, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 126-128.

Diaper, D. (1987). POMESS: A people orientated methodology for expert system specification, *Proceedings of the first European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, U.K., September, pp. D4.1-14.

Diederich, J., Linster, L., Ruhmann, I., and Uthmann, T. (1987). A Methodology for Integrating Knowledge Acquisition Techniques, *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, University of Reading, September, pp. E4.1-11.

Diederich, J., Ruhmann, I., and May, M. (1987). KRITON: A Knowledge Acquisition Tool for Expert Systems, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 1, *International Journal of Man-Machine Studies*, Vol. 26, No.1, pp. 29-40; also in Boose, J. H., and Gaines, B. R. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition*

for Knowledge-Based Systems, New York: Academic Press, 1988, pp. 83-94.

Dieng, R., Trousse, B. (1988). 3DKAT, a dependency-driven dynamic knowledge acquisition tool, *Proceedings of the AAAI-88 Integration of Knowledge Acquisition and Performance Systems Workshop*, St. Paul, August, npp.

Dupas, R., Millot, P. (1989). A Learning Expert System for Preventive Maintenance, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.354-364.

Dutta, S. (1988). Domain Independent Inductive Learning of Relevance, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 15.1-13.

Dvorak, D. L. (1987) Guide to CL-PROTOS: An Exemplar-Based Learning Apprentice, Technical Report AI88-87, AI Laboratory, Univ. of Texas at Austin, September.

Easterbrook, S. M. (1989). Distributed Knowledge Acquisition as a Model for Requirements Elicitation, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.530-543.

Eklund, P., Kellett, J. (1989). Prospects for Conceptual Graphs in Acquisition Interfaces, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.169-179.

Enand, R., Kahn, G. S., Mills, R. A. (1988). A Methodology for Validating Large Knowledge Bases, *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 5.1-11.

Ericsson, K. A., Simon, H. A. (1984). Protocol Analysis: Verbal Reports as Data, MA: The MIT Press.

Esfahani, L., Teskey, F. N. (1987). KET, A Knowledge Encoding Tool, *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, September, pp. E5.1-11.

Esfahani, L., Teskey, F. N. (1988). A Self-Modifying Rule-Eliciter, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 16.1-16.

Eshelman, L. (1988). MOLE: A Knowledge Acquisition Tool That Buries Certainty Factors, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, Vol. 29, No. 5, pp. 563-578.

Eshelman, L, Ehret, D., McDermott, J., and Tan, M. (1987). MOLE: A Tenacious Knowledge Acquisition Tool, special issue on the 1st AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 1, *International Journal of Man-Machine Studies*, Vol. 26, No.1, pp. 41-54; also in Boose, J. H., and Gaines, B. R. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 95-108.

Eskenazi, M., Faure, C., Frediani, S., Saitta, L. (1989). Knowledge Elicitation for Image Interpretation, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.544-553.

Fass, L.F. (1989). Learnability of CFLS: Inferring syntactic models from constituent structure, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 175-176.

Feldman, B.Z., Compton, P.J., Smythe, G.A. (1989). Hypothesis Testing: An Appropriate Task for Knowledge-Based Systems, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 10.1-20.

Fikes, R. E., Hart, P. E., Nilsson, N. J. (1972). Learning and executing generalized robot plans, *Artificial Intelligence*, 3, pp. 251-288.

Fischoff, B. (1989). Eliciting Knowledge for Analytical Representation, special issue of *IEEE Transactions on System, Man, and Cybernetics*, Vol. 19, No. 3, pp. 448-461.

Ford, K.M., Adams-Webber, J.R., Petry, F.E., Chang, P.J. (1989). An Approach to Knowledge Acquisition Based on the Structure of Personal Construct Systems, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 11.1-20.

Ford, K.M., Petry, F.E. (1989). Knowledge acquisition from repertory grids using a logic of confirmation, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 146-147.

Forsythe, D. E., Buchanan, B. G. (1988). Knowledge Elicitation in Theory and Practice: An Integrated Program of Research and Training, *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 6.1-7.

Forsythe, D.E., Buchanan, B.G. (1989). Knowledge Acquisition for Expert Systems: Some Pitfalls and Suggestions, special issue of *IEEE Transactions on System, Man, and Cybernetics*, Vol. 19, No. 3, pp. 435-442.

Fredman, N. E. (1987). Knowledge acquisition in a secure environment, *Proceedings of the Second Knowledge Acquisition for Knowledge-Based System Workshop*, Banff, October, pp. 7.0-8.

Freeman, P. (1985). Knowledge Elicitation - A Commercial Perspective, *Proceedings of the 1st International Expert System Conference*, London, October.

Freiling, M. J. (1988). Synthesizing an Effective Control Strategy from Acquired Domain Knowledge, *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 7.1-20.

Freiling, M., Alexander, J., Messick, S., Rehfuss, S., and Shulman, S. (1985). Starting a knowledge engineering project: A step-by-step approach. *AI Magazine*, Fall, 150-164.

Freiling, M.J., Jacobson, C.E. (1989). A new look at inference engine synthesis, *Knowledge Acquisition: An International Journal*, Vol. 1, No. 3, pp. 235-254.

Friel, P.G., Lockledge, J.C., Hart, P., Mayer, R.J., Sanders, L., Shashank, J. (1989). An Integrated Knowledge Capture / Design Analysis Assistant for Engineering, *IJCAI-89 Workshop on Knowledge Acquisition: Practical Tools and Techniques*, Detroit, August, npp.

Furuya, H., Hattori, F. (1989). Knowledge acquisition system for hierarchical classification problems, *Proceedings of COMPCON-89*, spring.

Gaines, B. R. (1977). System Identification Approximation and Complexity, *International Journal of General Systems*, 3, pp. 145-174.

Gaines, B. R. (1987a). An Overview of Knowledge Acquisition and Transfer, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 3, *International Journal of Man-Machine Studies*, Vol. 26, No.4, pp. 453-472; also in Gaines, B. R., and Boose, J. H. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 3-22.

Gaines, B. R. (1987b). Knowledge Acquisition for Expert Systems, *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, September, pp. A3.1-4.

Gaines, B. R. (1988a). Advanced Expert System Support Environments, *Proceedings of the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop*, October, pp. 8.0-14.

Gaines, B. R. (1988b). Second Generation Knowledge Acquisition Systems, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 17.1-14.

Gaines, B. R. (1989a) An Ounce of Knowledge is Worth a Ton of Data: Quantitative Studies of the Trade-Off between Expertise and Data based on Statistically Well-Founded Empirical Induction. KSI-89-2, Knowledge Science Institute, University of Calgary.

Gaines, B. R. (1989b). Knowledge Acquisition : the continuum linking machine learning and expertise transfer, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.90-101.

Gaines, B. R., Andreae, J. H. (1966). A Learning Machine in the Context of the General Control Problem, *Proceedings of the 3rd Congress of the International Federation for Automatic Control*, London: Butterworths.

Gaines, B. R., Boose, J. H. (1990). A Summary of the AAAI-sponsored Knowledge Acquisition for Knowledge-Based System Workshops, *AI Magazine*, [in press].

Gaines, B. R., Rappaport, A.T. (1989). The Automatic Generation of Classes, Objects, and Rules at the Interface Between Knowledge Acquisition Tools and Expert System Shells, *IJCAI-89 Workshop on Knowledge Acquisition: Practical Tools and Techniques*, Detroit, August, npp.

Gaines, B. R., Rappaport, A.T., Shaw, M.L.G. (1989). A Heterogeneous Knowledge Support System, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 13.1-20.

Gaines, B. R., Sharp, M. (1987). A Knowledge Acquisition Extension to Notecards, *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, September, pp. C1.1-7.

Gaines, B. R., Shaw, M. L. G. (1981). New Directions in the Analysis and Interactive Elicitation of Personal Construct Systems, in M. L. G. Shaw (Ed.), *Recent Advances in Personal Construct Technology*, New York: Academic Press.

Gaines, B. R., Shaw, M. L. G. (1986). Interactive Elicitation of Knowledge from Experts, *Future Computing Systems*, Vol. 1, No. 2, pp. 151-190.

Gale, W. A. (1987). Knowledge-Based Knowledge Acquisition for a Statistical Consulting System, special issue on the 1st AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 1, *International Journal of Man-Machine Studies*, Vol. 26, No.1, pp. 55-

64; also in Boose, J. H., and Gaines, B. R. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 109-118.

Gale, W. A. (1988). An Interactive Performance Based Knowledge Acquisition Methodology, *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 9.1-17.

Gammack, J. G., and Young, R. M. (1984). Psychological Techniques for Eliciting Expert Knowledge, in *R&D in Expert Systems, Proceedings of the 4th Expert System Conference*, Warwick, ed Max Bramer, CUP.

Gams, M. (1988). A New Breed of Knowledge Acquisition Systems uses Redundant Knowledge?!, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 18.1-7.

Gappa, U. (1988). Classika: A Knowledge Acquisition System Facilitating the Formalization of Advanced Aspects in Heuristic Classification, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 19.1-16.

Gappa, U. (1989a). CLASSIKA: A Knowledge Acquisition Tool for Use by Experts, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 14.1-15.

Gappa, U. (1989b). The Knowledge Acquisition Tool CLASSIKA, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 65-67.

Gappa, U., Puppe, F. (1989). The Knowledge Acquisition Tool CLASSIKA, *IJCAI-89 Workshop on Knowledge Acquisition: Practical Tools and Techniques*, Detroit, August, npp.

Garg-Janardan, C. (1988). Integration issues in knowledge acquisition for building expert systems, *Proceedings of the AAAI-88 Integration of Knowledge Acquisition and Performance Systems Workshop*, St. Paul, August, npp.

Garg-Janardan, C., and Salvendy, G. (1987). A Conceptual Framework for Knowledge Elicitation, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 3, *International Journal of Man-Machine Studies*, Vol. 26, No.4, pp. 521-532; also in Boose, J. H., and Gaines, B. R. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 119-130.

Goguen, J. A., and Meseguer, J. (1983). Programming with Parametrized Abstract Objects in *OBJ, Theory and Practice of Programming Technology*, Amsterdam: North-Holland.

Golden, B., Wasil, E., and Harker, P. (eds.) (1989). *The Analytic Hierarchy Process: Applications and Studies*, Springer-Verlag, September.

Gomez, F. (1989). Knowledge Acquisition From Natural Language For Expert Systems Based On Classification, Problem-Solving Methods, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 15.1-18.

Gonzalez, J. C., Alcaraz, F. (1989). A Learning System for INFERNO Inference Networks, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.365-379.

Goodman, R. M. F., Smyth, P. (1987a). ITRULE: An information theoretic rule-induction algorithm, *Proceedings of the first European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, U.K., September, pp. F3.1-8.

Goodman, R. M. F., Smyth, P. (1987b). Learning from Examples Using Information Theory, *Proceedings of the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop*, October, pp. 9.0-18.

Green, M., Eshelman, L. (1989). Knowledge Acquisition for Causal Reasoning, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 68-70.

Grover, M. D. (1983). A Pragmatic Knowledge Acquisition Methodology, *Proceedings of the Eighth International Conference of Artificial Intelligence*, Karlsrue, West Germany, pp. 436-438.

Gruber, T. R. (1988a). Acquiring Strategic Knowledge from Experts, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, Vol. 29, No.5, pp. 579-598.

Gruber, T. R. (1988b). A Method for Acquiring Strategic Knowledge, *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 10.1-21.

Gruber, T. R. (1989a). The acquisition of strategic knowledge. Ph.D. dissertation, Department of Computer and Information Science, University of Massachusetts.

Gruber, T. R. (1989b). *The Acquisition of Strategic Knowledge*. Academic Press.

Gruber, T. R. (1989c). A method for acquiring strategic knowledge, *Knowledge*

Acquisition: An International Journal, Vol. 1, No. 3, pp. 255-278.

Gruber, T. R., Cohen, P. R. (1987). Design for Acquisition: Principles of Knowledge System Design to Facilitate Knowledge Acquisition, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 2, *International Journal of Man-Machine Studies*, Vol. 26, No.2, pp. 143-160; also in Boose, J. H., and Gaines, B. R. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 131-148.

Gruber, T. R., Cohen, P. R. (1989). The design of an automated assistant for acquiring strategic knowledge, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 147-151.

Hadzikadic, M. (1988). Inc2: A Prototype-Based Incremental Conceptual Clustering System, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 20.1-13.

Haers M., d'Ydewalle G. (1989). The problem of knowledge extraction with human experts : A case of Expert Verbalizations, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.19-33.

Handa, K., Ishizaki, S. (1988). Learning Importance of Concepts: Construction of Representative Network, *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 11.1-15.

Handa, K., Ishizaki, S. (1989). Acquiring Knowledge about A Relation between Concepts, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.380-390.

Hardt, S. L. (1988). The roles of core knowledge and case history in the integration of knowledge acquisition and performance in the PACKS project, *Proceedings of the AAAI-88 Integration of Knowledge Acquisition and Performance Systems Workshop*, St. Paul, August, npp.

Hausen, H. L., Neusser, H. J. (1987). Knowledge acquisition for method and tool selection, *Proceedings of the first European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, U.K., September, pp. D5.1-12.

Hayes-Roth, F., Lenat, D. B., Waterman, D. A. (eds.) (1983). *Building Expert Systems*, Addison-Wesley: Reading, MA.

Hayslip, I. C., Eilbert, J. L. (1989). Capturing Air Combat Maneuvering Knowledge In A Mixed Representation Containing Neural Networks, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 87-90.

Hayward, S. A., Wielinga, B. J., Breuker, J. A. (1987). Structured Analysis of Knowledge, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 3, *International Journal of Man-Machine Studies*, Vol. 26, No.4, pp. 487-499; also in Boose, J. H., and Gaines, B. R. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 149-160.

Herrmann, J. (1989). Computer Aided Acquisition of Procedural Knowledge About VLSI-Design, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 56-59.

Herrmann, J., Beckmann, R. (1989). Malefiz - A Learning Apprentice System that Acquires Geometrical Knowledge about a Complex Design Task, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.391-405.

Hermann, J., Franzke, H. (1988). Requirements for computer-aided knowledge acquisition in an ill-structured domain: A case study, *Proceedings of the Second European Knowledge Acquisition Workshop*, Bonn, June, pp. 21.1-13.

Hink, R. F., Woods, D. L. (1987). How humans process uncertain knowledge: An introduction for knowledge engineers. *AI Magazine*, 8:3, pp. 41-53.

Hoffman, R. (1989). A brief survey of methods for extracting the knowledge of experts, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 19-27.

Holtzman, S. H. (1989). *Intelligent Decision Systems*, Reading, MA, Addison-Wesley.

Hsieh, L., Hayes, R., Hofmann, M., Patrick, A., Mallory, J. (1988). A knowledge acquisition kit (KNACK), Technical Report, IBM Palo Alto Scientific Center.

Hunter, L. (1988). Knowledge Acquisition Planning, *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 12.1-18.

Jacobson, C., Freiling, M. J. (1988). ASTEK: A Multi-Paradigm Knowledge Acquisition Tool for Complex Structured Knowledge, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, Vol 29, No. 3, pp. 311-328.

Jacobson, C., Freiling, M. J. (1989). Knowledge Dependencies, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 16.1-19.

Jagannathan, V., and Elmaghraby, A. S. (1985). MEDKAT: Multiple Expert Delphi-Based Knowledge Acquisition Tool, *Proceedings of the ACM NE Regional Conference*, Boston, October, pp. 103-110.

Jansen, B., Compton, P. (1988). The Knowledge Dictionary: An application of software engineering techniques to the design and maintenance of expert systems, *Proceedings of the AAAI-88 Integration of Knowledge Acquisition and Performance Systems Workshop*, St. Paul, August, npp.

Jansen, B., Compton, P. (1989). The Knowledge Dictionary: Storing Different Knowledge Representations, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.448-461.

Jansen-Winkeln, R. M. (1988). WASTL: An Approach to Knowledge Acquisition in the Natural Language Domain, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 22.1-15.

Jansson, C., and Rudström, Å. (1989). Re-Use of Problem-Solving Experience for the Weekly Planning of School Meals, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 145-148.

Johnson, N. E. (1987). Mediating Representations in Knowledge Elicitation, *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, September, pp. A2.1-10.

Johnson, L., Johnson, N. E. (1987). Knowledge Elicitation Involving Teachback Interviewing, in A. Kidd (ed.), *Knowledge Elicitation for Expert Systems: A Practical Handbook*, New York: Plenum Press.

Johnson, N. E., Tomlinson, C. M. (1988). Knowledge Elicitation for Second Generation Expert Systems, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 23.1-10.

Johnson, N. E., Tomlinson, C. M., Johnson, L. (1988). Knowledge Representation for Knowledge Elicitation, *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 13.1-9.

Johnson, P. E., Zaulkernan, I., Garber, S. (1987). Specification of Expertise: Knowledge Acquisition for Expert Systems, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 2, *International Journal of Man-Machine Studies*, Vol. 26, No.2, pp. 161-182; also in Gaines, B. R., and Boose, J. H. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 125-146.

Jones, W. P. (1989). Bringing Corporate Knowledge Into Focus with CAMEO, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 17.1-23.

Joseph, R. L. (1989). Graphical Knowledge Acquisition, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 18.1-16.

Kahn, G. S., Breaux, E. H., Joeseph, R. L., and DeKlerk, P. (1987). An Intelligent Mixed-Initiative Workbench for Knowledge Acquisition, special issue on the 1st AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 4, *International Journal of Man-Machine Studies*, Vol. 27, No. 2, pp. 167-180; also in Boose, J. H., and Gaines, B. R. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 161-174.

Kahn, G., Nowlan, S., McDermott, J. (1985a). Strategies for Knowledge Acquisition, *IEEE Transactions of Pattern Analysis and Machine Intelligence*, PAMI-7 (3), pp. 511-522.

Kahn, G., Nowlan, S., McDermott, J. (1985b). MORE: An Intelligent Knowledge Acquisition Tool, in the *Proceedings of the Ninth Joint Conference on Artificial Intelligence*, Los Angeles, CA, August, pp. 581-584.

Kahneman, D., Slovic, P., Tversky, A. (Eds.) (1982). *Judgment Under Uncertainty: Heuristics and Biases*. Cambridge: Cambridge University Press.

Karbach, W., Voß, A., Tong, X. (1988). Filling in the knowledge acquisition gap: via KADS' models of expertise to ZDEST-2's expert systems, *Proceedings of the Second European Knowledge Acquisition Workshop*, Bonn, June, pp. 31.1-17.

Kellogg, C., Gargan, R.A.Jr., Mark, W., McGuire, J.G., Pontecorvo, M., Schlossberg, J.L., Sullivan, J.W., Genesereth, M.R., Singh, N. (1989). The acquisition, verification, and explanation of design knowledge, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 163-165.

Kelly, G. A. (1955). *The Psychology of Personal Constructs*, New York: Norton.

Kidd, A. L. and Cooper, M. B. (1985). Man-Machine Interface Issues in the

Construction and Use of an Expert System, *International Journal of Man-Machine Studies*, vol. 22.

Kietz, J.-U. (1988). Incremental and Reversible Acquisition of Taxonomies, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 24.1-11.

Killin, J. L., and Hickman, F. R. (1986). The Role of Phenomenological Techniques of Knowledge Elicitation in Complex Domains, *Proceedings of Expert Systems '86*, Brighton.

Kitto, C. M. (1988). Progress in Automated Knowledge Acquisition Tools: How Close Are We to Replacing the Knowledge Engineer?, *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 14.1-13.

Kitto, C. M., and Boose, J. H. (1987). Heuristics for Expertise Transfer: The Automatic Management of Complex Knowledge Acquisition Dialogs, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 2, *International Journal of Man-Machine Studies*, Vol. 26, No. 2, pp. 183-202; also in Boose, J. H., and Gaines, B. R. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 175-194.

Kitto, C. M., and Boose, J. H. (1989). Selecting Knowledge Acquisition Tools and Strategies Based on Application Characteristics, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, Vol. 31, No. 2.

Klein, G.A., Calderwood, R., MacGregor, D. (1989). Critical Decision Method for Eliciting Knowledge, special issue of *IEEE Transactions on System, Man, and Cybernetics*, Vol. 19, No. 3, pp. 462-472.

Klinker, G. (1989). A Framework for Knowledge Acquisition, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.102-116.

Klinker, G., Bentolila, J., Genetet, S., Grimes, M., and McDermott, J. (1987). KNACK: Report-Driven Knowledge Acquisition, special issue on the 1st AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 1, *International Journal of Man-Machine Studies*, Vol. 26, No.1, pp. 65-80; also in Boose, J. H., and Gaines, B. R. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 195-210.

Klinker, G., Boyd, C., Dong, D., Maiman, J., McDermott, J., Schnelbach, R. (1989). Building expert systems with KNACK, *Knowledge Acquisition: An International Journal*, Vol. 1, No. 3, pp. 299-320.

Klinker, G., Genetet, S., McDermott, J. (1988). Knowledge Acquisition for Evaluation Systems, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, Vol. 29, No. 6, pp. 715-732.

Kodratoff, Y. (1987). Machine Learning and Explanations, *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, September, pp. A5.1-9.

Kodratoff, T. (1989). Characterizing Machine Learning Programs: A European Compilation, working paper, CNRS & Université Paris Sud, April.

Kodratoff, Y., Manago, M. (1987). Generalization in a Noisy Environment: The Need to Integrate Symbolic and Numeric Techniques in Learning, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 4, *International Journal of Man-Machine Studies*, Vol. 27, No. 2, pp. 181-204; also in Gaines, B. R., and Boose, J. H. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 299-322.

Kodratoff, Y., Michalski, R. S. (eds.) (1989). *Machine Learning: An Artificial Intelligence Approach, Volume III*, Morgan Kaufmann.

Kodratoff, Y., Tecuci, G. (1988). Learning at Different Levels of Knowledge, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 3.1-17.

Konrad, E., Tong, X. (1989). Model Directed Inference Structure Acquisition, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.128-138.

Kontio, J., Lounamaa, P. (1989). A Graphical Framework for knowledge Acquisition and representation, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.490-501.

Kopec, D., Latour, L. (1989). Towards an expert/novice learning system with application to infectious disease, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 140-143.

Kornecki, A.J. (1989). Operational knowledge acquisition problems for air traffic expert controller, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 165-166.

Krishnamurthi, M., Underbrink, A.J. (1989). Knowledge acquisition in a machine fault diagnosis shell, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 84-92.

Kwasnik, B.H. (1989). Extraction of Knowledge About the Cognitive Process of browsing from discourse and Thinking-Out-Loud Protocols, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 75-77.

LaFrance, M. (1986). The knowledge acquisition grid: A method for training knowledge engineers, *Proceedings of the First Knowledge Acquisition for Knowledge-Based System Workshop*, Banff, November, pp. 26.0-14.

Laird, J. E., Rosenbloom, P. S., Newell, A. (1986). Chunking in SOAR: The anatomy of a general learning mechanism, *Machine Learning*, 1, pp. 11-46.

Lancaster, J., Westphal, C.R., McGraw, K.L. (1989). A cognitively valid knowledge acquisition tool, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 152-154.

Langley, P., Bradshaw, G. L., Simon, H. A. (1983). Rediscovering chemistry with the BACON system, in R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (eds.), *Machine Learning Vol 2*, Los Altos: Morgan-Kaufmann, pp. 425-469.

Langley, P., Zytkow, J. M., Simon, H. A., and Bradshaw, G. L. (1986). Rediscovering chemistry with the BACON system, in R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (eds.), *Machine Learning: An Artificial Intelligence Approach, Vol. 3*, Palo Alto: Tioga, pp. 307-329.

Laske, O. E. (1987). A decompilation approach to knowledge acquisition: Using concurrent expert reports as the basis for automated knowledge modeling, *Proceedings of the first European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, U.K., September, pp. D8.1-13.

Laskey, K.B., Cohen, M.S., Martin, A.W. (1989). Representing and Eliciting Knowledge for Uncertain Evidence and Its Implications, special issue of *IEEE Transactions on System, Man, and Cybernetics*, Vol. 19, No. 3, pp. 536-545.

Lavrac, N., Mozetic, I. (1989). Methods for Knowledge Acquisition and Refinement in Second Generation Expert Systems, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.554-564.

LeClair, S.R. (1989). Interactive learning: A multiexpert paradigm for acquiring new knowledge, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 34-44.

Lee, J. (1989). Task-Embedded Knowledge Acquisition through a Task-Specific Language, *IJCAI-89 Workshop on Knowledge Acquisition: Practical Tools and Techniques*, Detroit, August, npp.

Lefkowitz, L. S., Lesser, V. R. (1988). Knowledge Acquisition as Knowledge Assimilation,, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, Vol. 29, No. 2, pp. 215-226.

Lehner, P.E. (1989). Toward an Empirical Approach to Evaluating the Knowledge Base of an Expert System, special issue of *IEEE Transactions on System, Man, and Cybernetics*, Vol. 19, No. 3, pp. 658-662.

Leishman, D. (1989). Analogy in Knowledge Acquisition: Description of an Analogical Tool, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 19.1-13.

Lenat, D. B. (1983). EURISKO: A program that learns new heuristics and domain concepts, *Artificial Intelligence*, 21, pp. 61-98.

Lenat, D. B., Prakish, M., Shepard, M. (1986). CYC: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks, *AI Magazine* 6(4), pp. 65-85.

Levi, K. (1989). Expert Systems Should be More Accurate than Human Experts: Evaluation Procedures from Human Judgment and Decision Making, special issue of *IEEE Transactions on System, Man, and Cybernetics*, Vol. 19, No. 3, pp. 647-657.

Levi, K., Perschbacner, D., Shalin, V. (1988). Learning Plans and Information Requirements for Pilot Aiding, *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 16.1-13; and *Proceedings of the AAAI-88 Integration of Knowledge Acquisition and Performance Systems Workshop*, St. Paul, August, npp.

Lewis, K. A., Kahn, G. S. (1988). TestBuilder™: An integrated expert system development environment, *Proceedings of the AAAI-88 Integration of Knowledge Acquisition and Performance Systems Workshop*, St. Paul, August, npp.

Liang, T.-P. (1989). Empirical Knowledge Refinement in Noisy Domains, *Proceedings of the Fourth Knowledge*

Acquisition for Knowledge-Based Systems Workshop, Banff, October, pp. 20.1-15.

Lindsay, E., Cameron, R., Fugure, J., Niem, L. (1988). Knowledge acquisition for Canadian armed forces army battlefield decision support system, *Proceedings of the AAAI-88 Integration of Knowledge Acquisition and Performance Systems Workshop*, St. Paul, August, npp.

Linster, M. (1987). On Structuring Knowledge for Incremental Knowledge Acquisition, *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, September, pp. D1.1-8.

Linster, M. (1988a). Kriton: A Knowledge Elicitation Tool for Expert Systems, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 4.1-9.

Linster, M. (1988b) A critical look at KRITON. Boose, J.H. & Gaines, B.R. (Eds) *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*.. pp.17-1-17-19. Banff (November).

Linster, M. (1989a). Integrating Acquisition, Representation and Interpretation, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.565-577.

Linster, M. (1989b). Towards a second generation knowledge acquisition tool, *Knowledge Acquisition: An International Journal*, Vol. 1, No. 2, pp. 163-184.

Liou, Y.I., Weber, E.S., Nunamaker, J.F. (1989). A Methodology for Knowledge Acquisition In A Group Decision Support System Environment, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 21.1-19.

Littman, D. C. (1987). Modeling Human Expertise in Knowledge Engineering: Some Preliminary Observations, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 1, *International Journal of Man-Machine Studies*, Vol. 26, No.1, pp. 81-92; also in Gaines, B. R., and Boose, J. H. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 93-106.

Loftin, B.R., Saito, T., Wang, L., Baffes, P. (1989). Automating Knowledge Acquisition for Intelligent Training Systems, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 80-82.

MacDonald, B.A., Witten, I.H. (1989). A Framework for Knowledge Acquisition Through Techniques of Concept Learning, special issue of *IEEE Transactions on System, Man, and Cybernetics*, Vol. 19, No. 3, pp. 499-512.

Mahling, D. E., Croft, W. B. (1988). Knowledge Acquisition for Planners, *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 18.1-18.

Maler, O. (1987). Student modeling: Learning a function from examples while knowing a close function, *Proceedings of the first European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, U.K., September, pp. F4.1-14.

Marcus, S. (1987). Taking Backtracking with a Grain of SALT, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 3, *International Journal of Man-Machine Studies*, Vol. 26, No. 4, pp. 383-398; also in Boose, J. H., and Gaines, B. R. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 211-226.

Marcus, S. L. (1988a). Understanding Subtasks from a Piecemeal Collection of Knowledge, *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 19.1-22.

Marcus, S. L. (1988b). *Automating Knowledge Acquisition for Expert Systems*, Kluwer: Boston.

Marcus, S. (1989). Understanding decision ordering from a piece-meal collection of knowledge, *Knowledge Acquisition: An International Journal*, Vol. 1, No. 3, pp. 279-298.

Marcus, S., McDermott, J., and Wang, T. (1985). Knowledge Acquisition for Constructive Systems, in the *Proceedings of the Ninth Joint Conference on Artificial Intelligence*, Los Angeles, CA, August, pp. 637-639.

Martial, F. von, Victor, F. (1988). Collaborative construction of a office knowledge base: Experiences and suggestions, *Proceedings of the Second European Knowledge Acquisition Workshop*, Bonn, June, pp. 32.1-14.

Martin, J.D., Redmond, M. (1989). Acquiring knowledge by explaining observed problem solving, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 77-82.

Matheus, C.J. (1989). Similarity-Based Learning, Feature Construction, and Construction Induction, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 49-51.

Matwin, S., Oppacher, F. (1989) Knowledge Acquisition Using a Machine Learning System with User-Limited Autonomy, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 133-136.

Maulsby, D. L., James, G. A., Witten, I. H. (1989). Evaluating Interaction in Knowledge Acquisition: A Case Study, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.406-419.

Maulsby, D. L., Witten, I. H. (1988). Acquiring Graphical Know-How: An Apprenticeship Model, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 34.1-16.

McGraw, K. L. (1989). Developing a Cognitively-Based Toolkit for Knowledge Acquisition, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 7-9.

McGraw, K. L., Lancaster, J.S. (1989). An Integrated Toolkit for Cognitively-Based Knowledge Acquisition, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 22.1-10.

McHugh, B. (1987). Knowledge acquisition and nuclear safety advisors, *Proceedings of the first European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, U.K., September, pp. C4.1-10.

Mellis, W. (1989). A General Approach to the Use of Background Knowledge in a Numerical Induction Algorithm, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.420-434.

Merrem, F.H. (1989). Automatic generation of knowledge structures, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 160-162.

Meyer, M.A., Booker, J.M. (1989). A Practical Program For Handling Bias In Knowledge Acquisition, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 23.1-19.

Meyer, M. A., Mniszewski, S. M., Peaslee, Jr., A. T. (1989). Using three minimally biasing elicitation techniques for knowledge acquisition, *Knowledge Acquisition: An International Journal of Knowledge Acquisition for Knowledge-Based Systems*, Vol. 1,. No. 1, pp. 59-72.

Micciche, P.F., Lancaster, J.S. (1989). Applications of neurolinguistic techniques to knowledge acquisition, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 28-33.

Michalski, R. S. (1983). Theory and Methodology of Inductive Learning, in R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (eds.), Machine Learning, An Artificial Intelligence Approach, Tioga Publishing Company.

Michalski, R. S., Chilausky, R. L. (1980). Knowledge Acquisition by Encoding Expert Rules versus Computer Induction from Examples - A Case Study Involving Soybean Pathology. *International Journal of Man-Machine Studies*, Vol. 12, pp. 63-87.

Michalski, R.S., Carbonell, J.G., Mitchell, T. M. Eds. (1983). *Machine Learning: An Artificial Intelligence Approach*. Palo Alto, California: Tioga.

Michalski, R.S., Carbonell, J.G., Mitchell, T. M., Eds. (1986). *Machine Learning: An Artificial Intelligence Approach Volume II*. Los Altos, California: Morgan Kaufmann.

Mitchell, T. M. (1982). Generalization as search, *Artificial Intelligence*, 18, pp. 203-226.

Mitchell, T. M., Mahadevan, S., and Steinberg, Louis I. (1985). LEAP: A Learning Apprentice for VLSI Design, *Proceedings of the International Conference on Artificial Intelligence*, 1985, Los Angeles.

Mitchell, T.M., Utgoff, P. E., Banerji, R. (1982). Learning by experimentation: Acquiring and modifying problem-solving heuristics, Technical Report LCSR-TR-31, Rutgers University.

Mittal, S., Dym, C. (1985). Knowledge acquisition from multiple experts. *AI Magazine*, Summer, pp. 32-36.

Möller, J. U. (1988). Knowledge Acquisition from Texts, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 25.1-16.

Monarch, I., Kaufmann, T., Subramanian, E. (1989). Knowledge-based Dynamic Hypertext to Guide Decision-Making and Rule Modification in Public Policy Law and Administration, *IJCAI-89 Workshop on Knowledge Acquisition: Practical Tools and Techniques*, Detroit, August, npp.

Monarch, I., Nirenburg, S. (1987). The Role of Ontology in Concept Acquisition for Knowledge-Based Systems, *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, September, pp. E1.1-14.

Moore, E. A., Agogino, A. M. (1987). INFORM: An Architecture for Expert-Directed Knowledge Acquisition, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 2, *International Journal of Man-Machine Studies*, Vol. 26, No.2, pp. 213-

230; also in Boose, J. H., and Gaines, B. R. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 227-244.

Moray, N. (1985). Sources of Bias and Fallibility in Humans, *Workshop on Knowledge Engineering in Industry*, University of Toronto, May.

Morik, K. (1987a). Acquiring Domain Models, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 1, *International Journal of Man-Machine Studies*, Vol. 26, No.1, pp. 93-104; also in Boose, J. H., and Gaines, B. R. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 245-256.

Morik, K. (1987b). Knowledge Acquisition and Machine Learning - The Issue of Modeling, *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, September, pp. A4.1-4.

Morik, K. (1989). Integration issues in knowledge acquisition systems, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 124-131.

Motta, E., Eisenstadt, M., Pitman, K., West, M. (1988). Support for knowledge acquisition in the Knowledge Engineer's Assistant (KEATS). *Expert Systems*, Vol. 5, No. 1, pp. 6-28.

Motta, E., Rajan, T., Eisenstadt, M. (1988). A Methodology and Tool for Knowledge Acquisition in KEATS-2, *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 21.1-20.

Motta, E., Rajan, T., Eisenstadt, M. (1989). Knowledge Acquisition as a Process of Model Refinement, HCRL Technical Report No. 40, Milton Keynes, U.K., January.

Moulin, S. (1989). SMAC: a prototype of a knowledge modeling and acquisition tool for the expert, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.579-591.

Mullin, T. (1989). Experts' Estimation of Uncertain Quantities and Its Implications for Knowledge Acquisition, special issue of *IEEE Transactions on System, Man, and Cybernetics*, Vol. 19, No. 3, pp. 616-625.

Murray, K. S. (1989). A Reactive Approach to Knowledge Integration, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 41-43.

Murray, K. S., Porter, B. W. (1988). Developing a Tool for Knowledge Integration: Initial Results, *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 22.1-12.

Murray, K.S., Porter, B.W. (1989). Controlling Search for the Consequences of New Information During Knowledge Integration, *IJCAI-89 Workshop on Knowledge Acquisition: Practical Tools and Techniques*, Detroit, August, npp.

Musen, M. A. (1988a). Conceptual Concepts of Interactive Knowledge Acquisition Tools, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 26.1-15.

Musen, M. A. (1988b). An Editor for the Conceptual Models of Interactive Knowledge-Acquisition Tools, *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 23.1-20.

Musen, M. A. (1989a). *Automated Generation of Model-based Knowledge-acquisition Tools*. Pitman.

Musen, M. A. (1989b).Knowledge acquisition at the metalevel: Creation of custom-tailored knowledge acquisition tools, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 45-55.

Musen, M. A. (1989c). Conceptual models of interactive knowledge acquisition tools, *Knowledge Acquisition: An International Journal of Knowledge Acquisition for Knowledge-Based Systems*, Vol. 1,. No. 1, pp. 73-88.

Musen, M. A., Fagan, L. M., Combs, D. M., and Shortliffe, E. H. (1987). Use of a Domain Model to Drive an Interactive Knowledge-Editing Tool, special issue on the 1st AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 1, *International Journal of Man-Machine Studies*, Vol. 26, No.1, pp. 105-121; also in Boose, J. H., and Gaines, B. R. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 257-274.

Mychaltchouk, P., Fayad, F., Talaslian, A. (1989). Towards Automating Knowledge Engineering: A Rule Generator: GeneR, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 112-114.

Myler, H.R., Gonzalez, A.J. (1989). Automated design data capture using relaxation techniques, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 169-170.

Nirenburg, S., Monarch, I., Kaufmann, T., Nirenburg, I., Carbonell (1988). Acquisition of Very Large Knowledge Bases: Methodology, Tools and Applications, *Proceedings of the Third Knowledge Acquisition for Knowledge-*

Based Systems Workshop, Banff, November, pp. 24.1-26.

Nobel, D. (1989). Schema-Based Knowledge Elicitation for Planning and Situation Assessment Aids, special issue of *IEEE Transactions on System, Man, and Cybernetics*, Vol. 19, No. 3, pp. 473-482.

Nordbø, I., Vestli, M., Sølvberg, I. (1989). METAMETH. Methodology for Knowledge Engineering, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.226-238.

Nygaard, K., Dahl, O. J. (1981). The Development of the SIMULA Languages, in R. L. Wexelblat (ed.), *History of Programming Languages*, 439-480, New York: Academic Press.

Oosthuizen, G. D. (1987). Graph induction, *Proceedings of the first European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, U.K., September, pp. F5.1-13.

Patel, J. (1988). ASKE: Towards an automated knowledge acquisition system, HCRL Technical Report No. 36, Milton Keynes, U.K., October.

Patel, J. (1989). ASKE: Towards Automatic Knowledge Acquisition, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.474-489.

Pau, L.F., Nielsen, S.S. (1989). Conceptual graphs as a visual language for knowledge acquisition in architectural expert systems, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 151.

Pazzani, M. J. (1987). Explanation-Based Learning for Knowledge-based Systems, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 3, *International Journal of Man-Machine Studies*, Vol. 26, No.4, pp. 413-434; also in Gaines, B. R., and Boose, J. H. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 215-238.

Pearl, J. (1986). Fusion, Propagation and Structuring in Belief Networks, Technical Report CSD-850022, R-42-VI-12, Cognitive Systems Laboratory, Computer Science Department, University of California, Los Angeles, CA, April.

Pettit, E. J., Pettit, M. J. (1987). Analysis of the Performance of a Genetic Algorithm-Based System for Message Classification in Noisy Environments, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 4, *International Journal of Man-Machine Studies*, Vol. 27, No. 2, pp. 205-220; also in Gaines, B. R., and Boose, J. H. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 333-350.

Plaza, E., Mantàras, R.L. (1989). Model-based knowledge acquisition for heuristic classification systems, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 98-105.

Polat, F., Güvenir, H. A. (1989). Completeness and Consistency Checking in Rule-Based Expert Systems, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 162-164.

Quinlan, J. R. (1983). Learning Efficient Classification Procedures and Their Application to Chess End Games, in R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (eds.), *Machine Learning, An Artificial Intelligence Approach*, Tioga Publishing Company.

Quinlan, J. R. (1987). Simplifying Decision Trees, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 5, *International Journal of Man-Machine Studies*, Vol 27, No. 3, pp. 221-234; also in Gaines, B. R., and Boose, J. H. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 239-252.

Rantanen, J. (1989). Hypermedia in Knowledge Acquisition and Specification of User Interface for KBS: An Approach and a Case Study, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 24.1-20.

Rappaport, A. (1988). Cognitive Primitives, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, Vol. 29, No.6, pp. 733-747.

Rappaport, A., Gaines, B. R. (1988). Integration of Acquisition and Performance Systems, *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 25.1-12; and *Proceedings of the AAAI-88 Integration of Knowledge Acquisition and Performance Systems Workshop*, St. Paul, August, npp.

Regoczei, S., Hirst, G. (1988). The Meaning Triangle as a Tool for the Acquisition of Abstract, Conceptual Knowledge, *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 26.1-19.

Regoczei, S., Hirst, G. (1989a). On 'Extracting knowledge from text': Modelling the architecture of language

users, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.196-211.

Regoczei, S., Hirst, G. (1989b). Sortal Analysis with SORTAL, a Software Assistant for Knowledge Acquisition, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 25.1-18.

Regoczei, S., Plantinga, E. P. O. (1987).Ontology and Inventory: A Foundation for a Knowledge Acquisition Methodology, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 5, *International Journal of Man-Machine Studies*, Vol. 27, No. 3, pp. 235-250.

Reynolds, R.G., Maletic, J.I., Porvin, S.E. (1989) PM: A Metrics Driven Plan Compiler, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 26.1-10; *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 141-144.

Ribar, G.S. (1989). Graphical Knowledge Acquisition, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 44-48.

Rissland, E. L. (1987). The Problem of Intelligent Example Selection, *Proceedings of the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop*, pp. 16.0-25.

Rose, J. R., Gelernter, H. (1989). ISOLDE: A System for Learning Organic Chemistry through Induction, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.297-310.

Rouse, W.B., Hammer, J.M., Lewis, C.M. (1989). On Capturing Humans Skulls and Knowledge: Algorithmic Approaches to Model Identification, special issue of *IEEE Transactions on System, Man, and Cybernetics*, Vol. 19, No. 3, pp. 558-573.

Rouveirol, C., Manago, M. (1988). Widening the Knowledge Acquisition Bottleneck, *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 27.1-18.

Ruberg, K., Cornick, S. M., James, K. A. (1988). House Calls: Building and Maintaining a Diagnostic Rule-Base , *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 28.1-20.

Rusk, R. A., Krone, R. M. (1984). The Crawford Slip Method (CSM) as a Tool for extraction of Expert Knowledge, in G. Salvendy (ed.), *Human-Computer Interaction*, New York: Elsevier, pp. 279-282.

Saaty, T. L. (1981). *The Analytic Hierarchy Process*, New York: McGraw-Hill.

Sammut, C., Banerjii, R. (1983). Hierarchical memories: An aid to concept learning, *Proceedings of the International Machine Learning Workshop*, Monticello, IL, June, pp. 22-24.

Sammut, C., Banerjii, R. (1986). Learning concepts by asking questions, in R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (eds.), *Machine Learning Vol. 2*, Los Altos, CA: Morgan-Kaufmann.

Samuel, A. L. (1963). Some studies in machine learning using the game of checkers, in E. A. Feigenbaum and J. Feldman (eds.), *Computers and Thought*, New York: McGraw-Hill.

Sandberg, J., Winkels, R., Breuker, J. (1988). Knowledge Acquisition for Intelligent Tutoring System, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 27.1-12.

Schaefer, P. (1989). Automated Knowledge Acquisition of Higher-Level Causal Models, *IJCAI-89 Workshop on Knowledge Acquisition: Practical Tools and Techniques*, Detroit, August, npp.

Schmidt, G., Wetter, T. (1989). Towards Knowledge Acquisition in Natural Language Dialogue, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.239-252.

Schreiber, G., Breuker, J., Bredeweg, B, Wielinga, B. (1988). Modeling in KBS Development, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 7.1-15.

Schröder, S., Niemann, H., Sagerer, G. (1988). Knowledge Acquisition for a Knowledge Based Image Analysis System, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 29.1-15.

Schuler, D. C., Russo, P. J., Boose, J. H., Bradshaw, J. M. (1988). Using Personal Construct Techniques for Collaborative Evaluation, *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 29.1-15.

Sebag, M., Schoenauer, M. (1988). Generation of Rules with Certainty and Confidence Factors from Incomplete and Incoherent Learning Bases, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 28.1-20.

Shadbolt, N. and Burton, M. (1989). The empirical study of knowledge elicitation techniques, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 15-18.

Shafer, G., Tversky, A. (1985). Languages and designs for probability judgment. *Cognitive Science*, Vol. 9, pp. 309-339.

Shah, R.P. (1989). A Framework for Efficient Knowledge Acquisition, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 137-140.

Shalin, V. L., Bloomfield, J.R., Bullemer, P.T. (1989). Knowledge Acquisition as Cognitive Modeling, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 27.1-15.

Shalin, V. L., Wisniewski, E. J., Levi, K. R., Scott, P. D. (1988). A Formal Analysis of Machine Learning Systems for Knowledge Acquisition, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, Vol. 29, No. 4, pp. 429-446.

Sharp, M. (1989). Knowledge Acquisition for Advice Systems: FUSION, A Network Cognitive Support System, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 28.1-19.

Shaw, M. L. G. (1984). Interaction Knowledge Elicitation, *Proceedings of the Canadian Information Processing Society Annual Conference*, Calgary, Canada.

Shaw, M. L. G. (1988). Problems of Validation in a Knowledge Acquisition System Using Multiple Experts, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 5.1-15.

Shaw, M. L. G. (1989a). A grid-based tool for knowledge acquisition: validation with multiple experts, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 168-169.

Shaw, M. L. G. (1989b). A Grid-Based Tool for Knowledge Acquisition, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 19-22.

Shaw, M. L. G., Gaines, B. R. (1987a). Techniques for Knowledge Acquisition and Transfer, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 5, *International Journal of Man-Machine Studies*, Vol 27., No. 3, pp. 251-280.

Shaw, M. L. G., Gaines, B. R. (1987b) KITTEN: Knowledge Initiation & Transfer Tools for Experts & Novices. *International Journal of Man-Machine Studies* 27(3), 251-280 (September).

Shaw, M. L. G., Gaines, B. R. (1988). A Methodology for Recognizing Conflict, Correspondence, Consensus and Contrast in a Knowledge Acquisition System, *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 30.1-19.

Shaw, M. L. G., Gaines, B. R. (1989). Knowledge Acquisition: some foundations, manual methods and future trends, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.3-18.

Shaw, M. L. G., Woodward, J. B. (1987) Validation of a knowledge support system. Boose, J.H. & Gaines, B.R. (Eds) *Proceedings of the Second Knowledge Acquisition for Knowledge-Based Systems Workshop*.. pp.18-0-18-15. Banff (October).

Shaw, M. L. G., Woodward, J. B. (1988). Validation in a Knowledge Support System: Construing Consistency with Multiple Experts, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, Vol. 29, No. 3, pp. 329-350.

Shaw, M. L. G., Woodward, J. B. (1989). Mental Models in the Knowledge Acquisition Process, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 29.1-24.

Shema, D. B., Boose, J. H. (1988). Refining Problem-Solving Knowledge in Repertory Grids Using a Consultation Mechanism, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, Vol. 29, p.447-460.

Shema, D. B., Boose, J. H., Bradshaw, J. M. (1988). Recent Progress in AQUINAS: A Knowledge Acquisition Workbench, *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 31.1-21.

Shema, D.B., Bradshaw, J.M., Covington, S.P., Boose, J.H. (1989). Design Knowledge Capture and Alternative Generation Using Possibility Tables in CANARD, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 30.1-19.

Silverman, B.G., Wenig, R.G., Wu, T. (1989). COPEing with ongoing knowledge acquisition from collaborating hierarchies of experts, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 170-171.

Silvestro, K. (1988). Using Explanations for Knowledge Base Acquisition, special issue

on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, Vol. 29, No. 2, pp. 159-170.

Simoudis, E. (1988). Learning Redesign Knowledge, *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 32.1-20.

Simoudis, E. (1989). Learning Redesign Knowledge, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 52-55.

Skuce, D. (1989). A Generic Knowledge Acquisition Environment Integrating Natural Language and Logic, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 165-169.

Skuce, D., Shenkang, W., Beauvillé, Y. (1989). A Generic Knowledge Acquisition Environment for Conceptual and Ontological Analysis, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 31.1-20.

Slator, B.M. (1988) Extracting lexical knowledge from dictionary text. Boose, J.H. & Gaines, B.R. (Eds) *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop..* pp.33-1-33-18. Banff (November), and *Knowledge Acquisition: An International Journal of Knowledge Acquisition for Knowledge-Based Systems*, Vol. 1,. No. 1, pp. 89-112.

Slator, B.M. (1989). Extracting lexical knowledge from dictionary text, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 173-174.

Slocombe, S., Moore, K. D. M., and Zelouf, M. (1986). Engineering Expert System Applications, *Proceedings of Expert Systems-86*, December, Brighton.

Smith, R. G., Winston, H. A., Mitchell, T. M., and Buchanan, B. G. (1985). Representation and Use of Explicit Justifications for Knowledge Base Refinements, *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA.

Spetzler, C., Stael von Holstein, C. (1983). Probability Encoding in Decision Analysis, in R. Howard and J. Matheson (eds.), *Readings on the Principles and Applications of Decision Analysis, Volume 2*, Palo Alto, CA: Strategic Decisions Group, pp. 601-626.

Stael von Holstein, C., Matheson, J.E. (1978). *A manual for encoding probability distributions.* Menlo Park, CA: SRI International.

Stephanou, H. (1987). Perspectives on imperfect information processing. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 17, pp. 780-798.

Sticklen, J., Chandrasekaran, B., Bond, W.E. (1989). Distributed causal reasoning, *Knowledge Acquisition: An International Journal*, Vol. 1, No. 2, pp. 139-162.

Stout, J., Caplain, G., Marcus, S., McDermott, J (1988). Toward Automating Recognition of Differing Problem-Solving Demands, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, Vol. 29, No. 5, pp. 599-611.

Subramanian, S., Freuder, E.C. (1989). Compiling rules from constraint satisfaction problem solving, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 177-178.

Suwa, M., Motoda, H. (1989a). Understanding Metaphors By Frustration-Based Learning Method, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 33.1-17.

Suwa, M., Motoda, H. (1989b). Acquisition of associative knowledge by the frustration-based learning method in an auxiliary-line problem, *Knowledge Acquisition: An International Journal of Knowledge Acquisition for Knowledge-Based Systems*, Vol. 1,. No. 1, pp. 113-137.

Szpakowicz, S. (1988) Semi-automatic acquisition of conceptual structure from technical texts. Boose, J.H. & Gaines, B.R. (Eds) *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop..* pp.35-1-35-16. Banff (November).

Taki, H. (1989). Knowledge Acquisition By Abductive And Inductive Explanation, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 34.1-19.

Taki, H., Fujii, Y. (1989). Operation Presumption: Knowledge Acquisition by Induction, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.34-48.

Tani, J., Hirobe, T., Niida, K., Koshijima, I., Murakami, H. (1989). New Learning Algorithm For Rule Extraction By Neural Network And Its Application, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 35.1-16.

Tanimoto, S. (1987). *The Elements of Artificial Intelligence.* Rockville, Md: Computer Science Press.

Terpstra, P. P., van Someren, M. W. (1988). Inde: A System for Knowledge Refinement and Machine Learning, *Proceedings of the Second European Knowledge Acquisition*

Workshop (EKAW-88), Bonn, June, pp. 30.1-8.

Thanassas, D. (1989). Knowledge Acquisition for Knowledge Discovery: A Theoretical Perspective, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 176-178.

Tolcott, M.A., Marvin, F.F., Lehner, P.E. (1989). Expert Decision Making in Evolving Situations, special issue of *IEEE Transactions on System, Man, and Cybernetics*, Vol. 19, No. 3, pp. 606-615.

Tong, X. (1989). MAC: Acquiring Abstract Knowledge, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 38-40.

Tong, X., Karbach, W. (1988). Filling in the Knowledge Acquisition Gap: via Kads' Models of Expertise to Zdest-2's Expert Systems, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 31.1-17.

Tournat, K. (1989). Comparing Structured Interview Techniques for Initial Expertise Elicitation, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 60-64.

Tranowski, D., Levitt, T., Riley, K. (1988). Performance-Based Knowledge Acquisition: A Structural Foundation for Traditional Introspective Knowledge Acquisition, *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 36.1-3.

Trice, A., Davis, R. (1989). Consensus Knowledge Acquisition, Technical Report, MIT School of Management, April.

Trimble, G., Cooper, C. N. (1987). Experience of Knowledge Acquisition for Expert Systems in Construction, *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, September, pp. C5.1-14.

Tschudi, F. (1988). Matrix representation of expert systems. *AI Expert*, October, pp. 44-53.

Tsui, C. (1988). A tool for assisting booklore acquisition. Technical Report, Chinese Academy of Sciences, August.

Tversky, A., Kahneman, D. (1974). Judgement under uncertainty, in Kahneman, D. (eds.), *Judgement Under Uncertainty: Heuristics and Biases*. Cambridge: Cambridge University Press.

Tversky, A., Kahneman, D. (1981). The framing of decisions in the psychology of choice, *Science*, 211, pp. 453-458.

Tversky, A., Sattah, S., Slovic, P. (1987). Contingent weighting in judgment and choice. Menlo Park, California: Stanford University.

Twine, S (1988). From information Analysis Toward Knowledge Acquisition, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 6.1-15.

Twine, S. (1989). A Model for the Knowledge Analysis Process, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, pp.253-268.

Valtorta, M. (1989). KADS vs. KEATS, *IIJCAI-89 Workshop on Knowledge Acquisition: Practical Tools and Techniques*, Detroit, August, npp.

Velardi, P., Pazienza, M. T., Magrini, S. (1989). Acquisition of semantic patterns from a natural corpus of texts, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 115-123.

von Winterfeldt, D., and Edwards, W. (1986). *Decision Analysis and Behavioral Research*. Cambridge: Cambridge University Press.

Waldron, V.R. (1985). Process Tracing as a Method for Initial Knowledge Acquisition, *Proceedings of the 2nd IEEE Conference on AI Applications*, Florida, December.

Waldron, V.R. (1989). Investigating the communication problems encountered in knowledge acquisition, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 143-144.

Wallsten, T., and Budescu, D. (1983). Encoding Subject Probabilities: A Psychological and Psychometric Review, *Management Science*, Vol. 29, No. 2, pp. 151-173.

Welbank, M. (1987a). Knowledge Acquisition: A Survey and British Telecom Experience, *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, September, pp. C6.1-9.

Welbank, M. (1987b). Knowledge Acquisition Update, Insight Study No. 5, Systems Designers, England.

Wells, T.T. (1989). Hypertext as a means for knowledge acquisition, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp.136-138.

Wetter, T., Schmalhofer, F. (1988). Knowledge Acquisition from Text-Based Think-Aloud Protocols: Situational Specification for a Legal Expert System, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 33.1-15.

Wielinga, B., Akkermans, H., Schreiber, G., Balder, J. (1989). A Knowledge Acquisition Perspective On Knowledge-Level Models, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 36.1-22.

Wilkens, D. C., Clancey, W. J., and Buchanan, G. G. (1987). Knowledge Base Refinement by Editing Abstract Control Knowledge, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 5, *International Journal of Man-Machine Studies*, Vol. 27, No. 4, pp. 281-294.

Williams, R. V., and Thomas, J. R. (1987). An interactive strategic planning expert system, *Proceedings of the first European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, U.K., September, pp. E7.1-19.

Wisniewski, E., Winston, H., Smith, R., Kleyn, M. (1987). Case Generation for Rule Synthesis, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 4, *International Journal of Man-Machine Studies*, Vol. 27, No. 3, pp. 295-313.

Witten, I. H., MacDonald, B. (1988). Using Concept Learning for Knowledge Acquisition, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*,Vol. 29, No. 2, pp. 171-196.

Wolf, W.A. (1989). Knowledge acquisition from multiple experts, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 138-140.

Woods, D. D., Hollnagel, E. (1987). Mapping Cognitive Demands and Activities in Complex Problems Solving Worlds, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 2, *International Journal of Man-Machine Studies*, Vol. 26, No.2, pp. 257-275; also in Gaines, B. R., and Boose, J. H. (eds), *Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems*, New York: Academic Press, 1988, pp. 45-64.

Woodward, B. (1988) Knowledge engineering at the front-end: defining the domain. Boose, J.H. & Gaines, B.R. (Eds) *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*.. pp.37-1-37-25. Banff (November).

Wrobel, S. (1988). Design Goals for Sloppy Modeling Systems, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, Vol. 29, No. 4, pp. 461-477.

Xie, S., Dumaresq, D. F., Winne, P. H. (1988). PRED: An Interface Development Tool for Editing Primitives in Knowledge Representation Languages, *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 38.1-17.

Yamada, S., Tsuji, S. (1989). Acquisition of macro-operators from worked examples i problem solving, *SIGART Newsletter*, Special Issue on Knowledge Acquisition, No. 108, April, pp. 171-172.

Yang, X. P., Okrent, D. (1989). Knowledge Representation Under Uncertainty By Using Bayesian Networks and Stochastic Models, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 91-95.

Yoshii, R. (1989). Portable Rule Acquisition Interface for Maintaining Expert Systems, *IJCAI-89 Workshop on Knowledge Acquisition*, Detroit, August, pp. 115-119.

Young, R. M., Gammack, J. (1987). Role of Psychological Techniques and Intermediate Representations in Knowledge Elicitation, *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, September, pp. D7.1-5.

Zinßmeister, G., Hoppe, H. U. (1989). Acquisition Of Task-Knowledge For An Adaptive User Interface, *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July, 1989, Paris, July, 1989, pp.269-284.

IMPLICATIONS OF THE SYSTEM-MODEL-OPERATOR METAPHOR FOR KNOWLEDGE ACQUISITION

William J. Clancey, Senior Research Scientist

Institute for Research on Learning
2550 Hanover Street
Palo Alto, CA 94304 USA

Expert systems can be viewed as programs that construct a model of some system in the world so that it can be assembled, repaired, controlled, etc. In contrast with most conventional computer programs, these models represent processes and structures by relational networks. Control knowledge for constructing such a model can be described as operators that construct a graph linking processes and structures causally, temporally, spatially, by subtype, etc. From this perspective, we find that the terminology of blackboard expert systems is not specific to a particular set of programs, but is rather a valuable perspective for understanding what every expert system is doing. Knowledge acquisition tools can be classified according to the systems they are designed to model, the modeling purpose or *task*, and the environmental constraints under which the modeling will take place.

1. INTRODUCTION

In the past decade, a broad collection of "generic" expert system shells have been developed, called *task-specific architectures* because they represent knowledge specialized for different problems such as diagnosis or design and often different domains such as electronics or medicine [2, 4, 6, 8, 10].

The technique for building expert systems has thus advanced from tools that provide an "empty knowledge base" (e.g., EMYCIN [15]) to tools that presuppose a particular task. These tools provide a way of organizing knowledge (a representational language) and an inference procedure for applying this knowledge. As we collect these tools and attempt to integrate them, we need to understand the relation between specific problems and general ways in which knowledge can be organized and applied. From the perspective of knowledge acquisition tools, can we relate "problems" and "solutions" in a big-switch knowledge acquisition program that could help an knowledge engineer choose and apply the appropriate task-specific architecture? In essence, how should we formalize the part of the knowledge base that gets reused so its capabilities and limits can be related to new problems?

In HERACLES-DX, a diagnostic shell developed from NEOMYCIN, the reusable knowledge consists of an inference procedure represented

by abstract tasks and metarules, and a language of relations by which domain knowledge is expressed. The metarules are abstract in the sense that they use variables rather than domain terms [3, 5].

Our first study of NEOMYCIN's metarules led to the heuristic classification description of inference structure (the relations among assertions). The analysis here focuses on the inference process itself, addressing the following questions:

- What guidelines can be provided for writing metarules for a new problem domain?
- How can we relate the proliferating set of application shells? What space of problem types and solutions is being covered? Is a dimensional analysis possible?
- Can we describe control knowledge in a way that would enable us to relate the diverse terminology used for representing it: metarules, blackboards, objects/methods?
- Is there a common way of describing control so the knowledge in different programs can be systematically combined into one shell? How is a diagnostic procedure for electro-mechanical problems different from a diagnostic procedure for medicine?

At the heart of this analysis is an evolving perspective of what constitutes a solution in reasoning or, put another way, how we can fruitfully describe and compare what inference procedures are doing. Search was described in MYCIN in terms of AND/OR trees; in terms of subtype and causal networks in CASNET [7] and CADUCEUS; and in terms of construction of a situation-specific model in ABEL, ACCORD, HASP [11], and NEOMYCIN. The view that inference consists of constructing and comparing situation-specific models of processes has been elaborated to describe inference procedures in these programs in terms of *operators* for manipulating *graphs* that represent *processes*. These ideas--operators, graphs, and processes--interrelate and form the basis for a synthesis that firmly anchors knowledge engineering in contributions to both computer science and traditional scientific modeling.

The system-model-operator description is not a new theory of intelligence or new architecture, but rather a means of abstractly describing existing programs. How can the methods be more easily taught and better tools be designed that abstract and combine the approaches previously viewed as disparate or incompatible? Indeed, one major result is an integrated description of object-oriented programming, rule-based programming, logic programming, and blackboard architectures. Although it may appear counterintuitive at first and even obscuring, a reformulation of past

work indicates that we would be right to say that *every expert system has a blackboard* and, from a related perspective, *every expert system is "model-based."* This is true in the same sense that we can say that MYCIN does top-down refinement of a disease classification, even though such a characterization would have been foreign to the original designers (indeed, they might have said that this was an appropriate characterization of the "opposition" approach used by researchers developing medical expert systems based on frame representations).

The redescription of NEOMYCIN's metarules in terms of graph-manipulation operators follows the familiar pattern in science of relating empirical phenomenon (the metarules and control knowledge from diverse expert systems) to a formal language (here, simple constructs from logic, set and graph theory) [1]. Unfortunately, a formalization phobia has made the very mention of mathematical techniques anathema to many researchers. The concrete examples provided here demonstrate that "doing logic on networks" [12] does not entail something lifeless or useless. Indeed, by such an analysis--which abstracts a wide variety of techniques to show a common approach--we find that expert systems researchers have perhaps unwittingly made major contributions to computer science in the development and use of languages for modeling processes.

Equally important, as we redescribe what expert systems are doing we make a major shift from a one-dimensional, programmer's view of rules and frames to descriptions on multiple levels, which are appropriate ways of describing any expert system:

- a *system* in the world being modeled for some purpose (a *task*),
- general and situation-specific *models of processes* in this system (corresponding to the knowledge base and a problem solution),
- *relational networks* that represent these processes (hierarchies and transitional graphs),
- *computational methods* for constructing these models (inference procedures),
- and finally, an implementation of the relational networks and inference procedure in some *programming language* (e.g., frame and rule-based languages).

This general framework is called the *system-model perspective* and forms the basis, along with the set/graph-operator description of inference procedures, for answering the questions posed above. The

title of this paper reflects the idea that control knowledge can be fruitfully described in terms of operators for constructing models of systems, and this is something that all expert systems do. As will become evident, we can use this perspective to define expert systems as a class of computer programs that use relational networks for representing processes so the system being modeled can be assembled, repaired, controlled, etc. Compilers and operating systems are interesting special cases--an important realization if we are to efficiently integrate the relational network approach of AI with computer science as a whole.

Successfully synthesizing the work on task-specific architectures and all its attendant jargon requires changing common ways of talking about AI programming in general. In particular, three fundamental changes are called for:

- view *qualitative reasoning* as including both classification and simulation models of processes (that is, all knowledge bases contain models and all expert systems do qualitative reasoning),

- view AI programming as a methodology for representing processes by relational networks (specifically, rule, frame, and blackboard languages are methods for encoding and manipulating relational networks),

- view a domain expert as an informant about some system in the world (therefore, knowledge acquisition is primarily concerned with modeling some system in the expert's world, in contrast with modeling his mental processes).

These shifts in perspective add up to a view of AI programming as a contribution to scientific and engineering modeling in general. In expert systems, processes are represented by spatial, temporal, causal, and subtype relations among objects and events, in contrast with traditional numeric models, which are based on relations among measures. The shift is from linear, quantitative measures of *substance* (mass, energy, length) to non-linear descriptions of *form* (orders, grammars, space/time relations), which parallels the scientific trend of this century, and indeed stems from the origin of heuristic programming in cybernetics and operations research [1, 16]. The system-model-operator approach of expert systems programming is implicitly a claim that qualitative modeling provides an appropriate rigor and set of dimensions for systems analysis and design, which was lacking in traditional numeric modeling.

Perhaps the most difficult part of our analysis is to realize that there are always different perspectives for describing a representation and never one right interpretation. Because representations can be used

to implicitly encode relations by position and ordering, such as elements in a list or clauses in a rule, we must often go beyond surface terminology to find the relational networks embedded in a set of expert system rules. An important claim of this paper is that the model-construction perspective readily applies to expert systems and that making it explicit in our descriptions of knowledge acquisition tools provides an valuable means for relating our programs--indeed, our knowledge bases--into a shared collection of representational languages and modeling techniques. Identifying our place in a historical perspective, the direction the research has been taking, and its contribution to the whole of science and engineering provides a substantial basis for teaching our methods and making them available to the application communities. This is no idle task, given that the very aim of expert systems research is to provide useful tools for every community from physicians to geneticists and civil engineers. And it is indeed in this connection that the systems-modelling description is so intuitive and useful.

The systems-modeling perspective provides a good basis for developing knowledge acquisition tools. To illustrate this, we will consider examples from XPLAIN and TOPO, a recent expert system developed in the HERACLES shell. We emphasize how expert systems can be generalized and more easily related to each other if we express strategic knowledge in terms of operators for constructing system models. Finally, the argument is summarized and more general conclusions are drawn.

2. XPLAIN: ABSTRACTING DOMAIN PRINCIPLES

XPLAIN [13], like NEOMYCIN, was developed to improve the explanation program of a medical program. The work complements NEOMYCIN by emphasizing how procedures can be written by an automatic-programming system from a domain model and set of prototype methods. Figure 1 gives a well-known example from the program.

The prototype method does not fit the definition of abstract control knowledge used in NEOMYCIN because it mentions the term "drug dose." "Finding" might refer to any domain from car diagnosis to legal reasoning. However, the domain suggested by "drug dose" is much narrower. While the system being treated isn't specified (it could be a sick house plant), practically speaking this prototype method will only be useful for medical diagnosis. This example illustrates that there is no sharp line between "domain specific" and "abstract;" it is just a judgement that we make based on the systems we know about. The final part of this section will consider principles for selecting a useful level of abstraction.

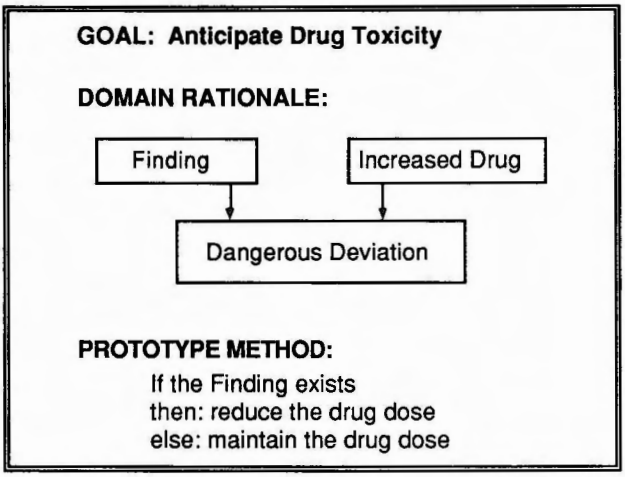


Figure 1. Example of a Domain Principle from XPLAIN

A second observation is that the domain rationale does not express the process-model relations between its terms. Figure 2 illustrates one way of formulating this domain principle in a NEOMYCIN-like metarule. Reinstantiating this rule in terms of our example, drug dosage is a quantitatively-variable action that causes an undesirable effect (a dangerous deviation) in the system being treated, proportional to the amount of the drug. In addition, another cause ($FINDING) of this effect is currently present. Therefore, to avoid increasing this undesirable effect further, the amount of the drug should be reduced.

```
TASK: ANTICIPATE-PLAN-SIDE-EFFECT
FOCUS: $PLAN-ACTION

IF:     (AND    (QUANTITATIVELY-VARIABLE $PLAN-ACTION)
                (CAUSES $PLAN-ACTION $EFFECT)
                (PROPORTIONAL-CAUSE $PLAN-ACTION $EFFECT)
                (NOT (DESIRABLE $EFFECT))
                (CAUSES $FINDING $EFFECT)
                (SAME CNTXT $FINDING))

THEN :          (TASK REDUCE-AMOUNT $PLAN-ACTION)
```

Figure 2. Example of a Domain Principle from XPLAIN

Obviously, there are many other ways of writing this rule. The example is intended to illustrate how abstraction can bring out process-modeling relations (e.g., proportional-cause) that are implicit

in the domain rationale and prototype method. Although it is not worked out here in detail, notice also how the metarule is written in terms of modifying a SSM of a system-modification plan, namely reducing the amount of the quantitatively-variable action (called "therapy" in medicine). Moving from the domain principle of Figure 1 to the abstract metarule of Figure 2 exemplifies the abstraction process of dropping domain-specific terms and expressing the inference procedure in terms of a process-model and some action that depends upon it. A knowledge engineer with this vocabulary will have a great advantage in analyzing other problems. With a metarule like this in the toolkit (contrasted with the domain principle in Figure 1), the next program should be much easier to build. In particular, the idea of a quantitatively-variable systems-control action is very basic to systems-modeling and should be taught as part of knowledge engineering methodology.

3. TOPO: BLACKBOARDS AND OPERATORS FOR COMPUTER NETWORK LAYOUT

TOPO is an expert system for logical topology design of computer networks; it is built in the HERACLES shell. TOPO serves as a front end for the programs developed by Digital Equipment Corporation for computer system configuration. TOPO provides a sales person with a language for modeling a potential client's business and information-processing requirements. TOPO proposes a layout of generic components and connections, suitable for input to sizing and configuration programs, such as XCON [9].

The development of TOPO illustrates the process of using the HERACLES task/metarule language to define operators for a configuration task. Recall that HERACLES was originally designed with tasks and metarules for diagnosis in NEOMYCIN. Here we show how the level of abstraction results in domain models, blackboards, and operators that can be reused in a wide variety of related applications. We also discover that the idea of a "physical-organizational model" is a common starting point for many expert systems.

The basic method of TOPO's reasoning is as follows:

1. Construct a model of the **physical and organizational structure** of the client's business. Describe floor locations of workgroups at each site.
2. Determine the **information-processing requirements**, leading to a preliminary sizing (e.g., the number of printers required).
3. Design the **network topology**:
   a. derive the backbone from the physical layout.
   b. represent the components and connections.

Following the lessons learned in our heuristic classification analysis, we first conjectured that TOPO would contain three kinds of models corresponding to these three steps. However, in the final design, we found that it was more convenient to view information-processing requirements as properties of workgroups, rather than as separate objects. In effect, there are relations between elements of the physical and organizational structure (e.g., sites and buildings). And there are relationships between elements of the network topology (e.g., segments and backbones). But we chose not to represent relations between information-processing requirements, and therefore do not require a separate model for them.

Conceptually, we view the reasoning of the program in terms of constructing a situation-specific model (SSM) for each connected system or process (e.g., Business-Site-1, Network-at-Site-2), illustrated by Figure 3. In effect, the program first constructs a model of the client's business and then produces a corresponding network design.

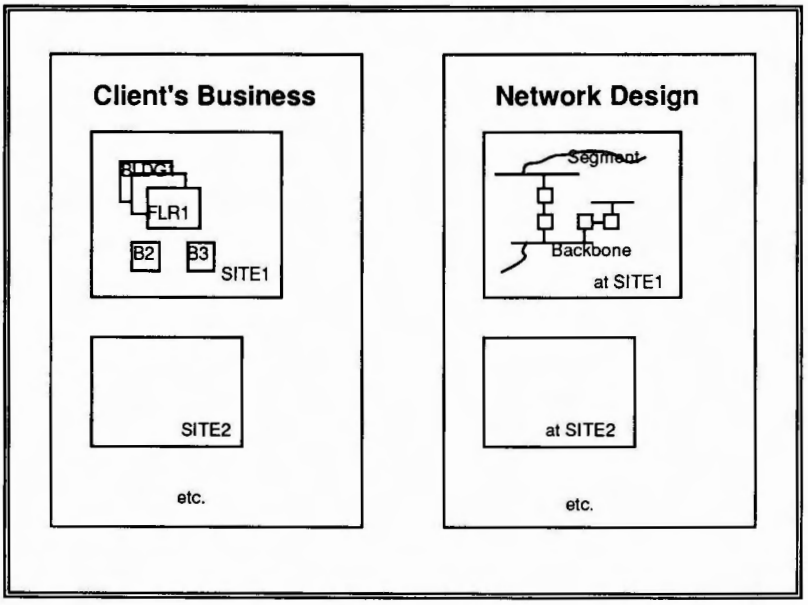


Figure 3. Two blackboards in TOPO (client business and network design) with a panel for each site; levels of each panel correspond to hierarchical connections (e.g., segments on a backbone).

The following is a generalization of how to define inference operators, based on our experience in developing TOPO:

1. Draw separate pictures for each system in the world being modeled (e.g., business and computer service networks).
2. Specify class structure of nodes in each SSM (e.g., buildings, segments) and draw separate SSMs for unconnected systems (e.g., different sites).
3. Draw links showing information mapping between SSMs of different types (e.g., from workgroup to equipment layout).
4. Describe operators for placing nodes in each SSM, linking them, and specifying their spatial and process attributes.

In practice, our descriptions of TOPO's operators (Figure 4) are abstracted from the tasks/metarules that carry out the necessary computations. A given operator typically corresponds to a single HERACLES task (and hence several metarules). Higher-level tasks control when the operators are performed (just as PROCESS-HYPOTHESIS in NEOMYCIN determines that the operator TEST-HYPOTHESIS is done before REFINE-HYPOTHESIS).

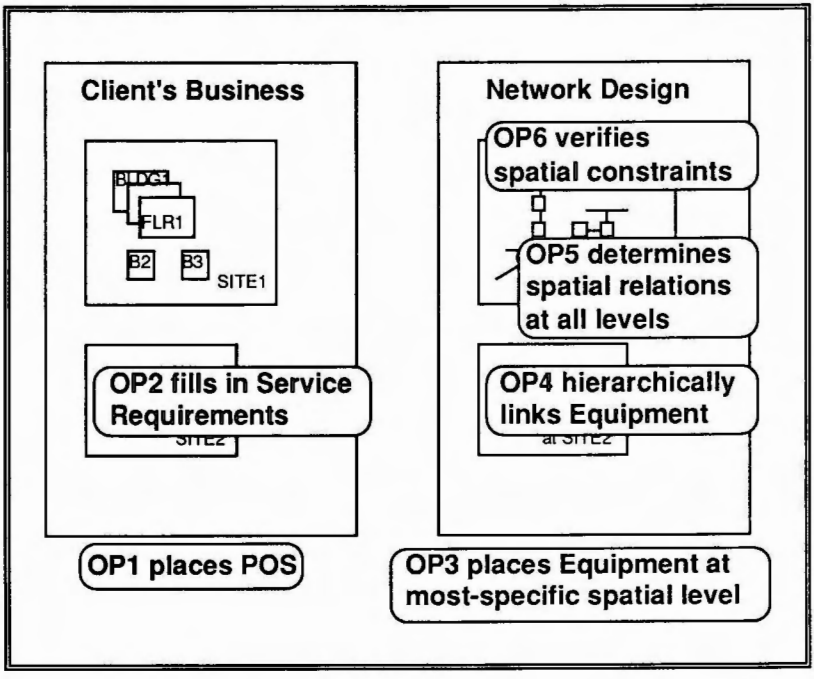


Figure 4. The six operators of TOPO's inference procedure place different nodes and links in the situation-specific models. The sizing table (quantifying the "amount of equipment") used by OP3 is computed by another program.

We can describe TOPO's operators in more detail in terms of the nodes and relations they manipulate on the blackboards (Table 1). In effect, the relations of the SSM are specializations of the general domain model. For example, the general domain model indicates that there can be a **USES/CONSUMES** relation between types of organizational-structures and types of services. A client-business SSM indicates that a particular organizational structure uses or consumes a particular service (e.g., "WORKGROUP-3 **USES/CONSUMES** information-storage").

Operators are implemented as HERACLES tasks; therefore the relations of Table 1 appear in metarule premises. The SSM "nodes" or terms of the relations appear as variables in the metarules.

OPERATOR	(RELATION Node1 Node2)
1) "Determine Position"	(**LOCATED-IN/AT** POS <place>)
2) "Determine Service Requirements"	(**USES/CONSUMES** Organizational-Structure Service)
3) "Process Sizing Data"	(**USED-BY** Organizational-Structure Equipment) (**PROVIDES** Equipment Service) (**AMOUNT-SUPPLY** Service Sizing-data-table) (**SIZING.DATA.TABLE** Equipment #)
4) "Derive Logical Topology from POS and Equipment "	(**PROVIDE-SERVICE-FOR** Physical-Structure Logical-Service-Components)
5) "Transfer properties from POS to Service"	(**PROVIDE-SERVICE-FOR** Physical-Structure Logical-Service-Components) (**LINEAR-SPAN** Physical-Structure #)
6) "Verify Skeletal Design"	<Various relations represented in constraintrules>

Table 1. Domain relations used by TOPO's operators.
(# = a number; POS = Physical-Organizational-Structure such as buildings or workgroups; Sizing = capacity of a service or supply)

To recapitulate, building TOPO in HERACLES was an incremental process in which we shifted attention back and forth between the details of writing metarules and the high-level definition of

blackboards and operators. Just as we were able to build CASTER very quickly using NEOMYCIN's relations and operators, we now conjecture that expert systems similar to TOPO can be constructed more quickly that TOPO itself. Towards this end, we want to describe TOPO more generally, so we can talk about its reasoning and representations at a level more general than computer network layout.

Generalizing, we can describe TOPO's blackboards and operators as being suitable for "social services network configuration." This characterizes a wide variety of potential expert systems that map between a model of social structures (e.g., a university) to a network of services or suppliers (Figure 5). In each of these programs, we will follow similar steps: determine the Physical-Organizational-Structure; determine service and local resource sizing requirements, and finally place and connect servers and suppliers. Our claim is that the relations and operators of Table 1 provide a useful level of abstraction for building such expert systems. Put another way, we don't simply give the knowledge engineer the tasks and metarules used in TOPO; we give the design of the blackboards in terms of the domain relations of Table 1 and a description of what the operators do.

Notice also that we are building up a language for describing kinds of *systems*. We distinguish between a POS and a service network. We observe that the service network involves a distribution of resources, not a single roving supplier. For example, TOPO's blackboards and operators are probably inadequate for solving the traveling salesman problem. We observe that other expert systems that provide "service protection" also start by constructing a POS model. For example, an expert system to configure an earthquake insurance policy for a univeristy would develop a model of the university's physical and organizational structure (e.g., a description of buildings and distribution of workgroups). Notice that such a POS model will contain new distinctions (relations) that are useful for configuring service protection, but not useful for configuring a service network. This suggests that an ultimate library of system models and operators will be roughly hierarchical, with sharing and specialization of descriptions for different tasks.

Figure 5 shows how a classification of system types can be used to describe how specific expert systems are related, and hence the knowledge engineering effort required (in boldface) to generalize and specialize existing knowledge bases. As always, we use the term "system" very generally here; an earthquake insurance policy is a kind of system. We conjecture that it takes much more time to

shift between systems that are of different abstract types (e.g., shifting from service networks to service protection). But it should be much easier to reuse blackboards and operators for systems that are of a similar type (e.g., converting operators developed for bus route configuration to rail configuration). The point of this diagram is to illustrate the research task of developing future knowledge acquisition tools. It shows just a small part of the space of potential expert systems that we might develop, but illustrates the kinds of conversions and adaptations that we expect to find throughout our enterprise. The blackboard/operator metaphor suggests that classifying our tools in terms of models of systems (e.g., telecommunication) will facilitate sharing and reuse of representations and inference procedures.

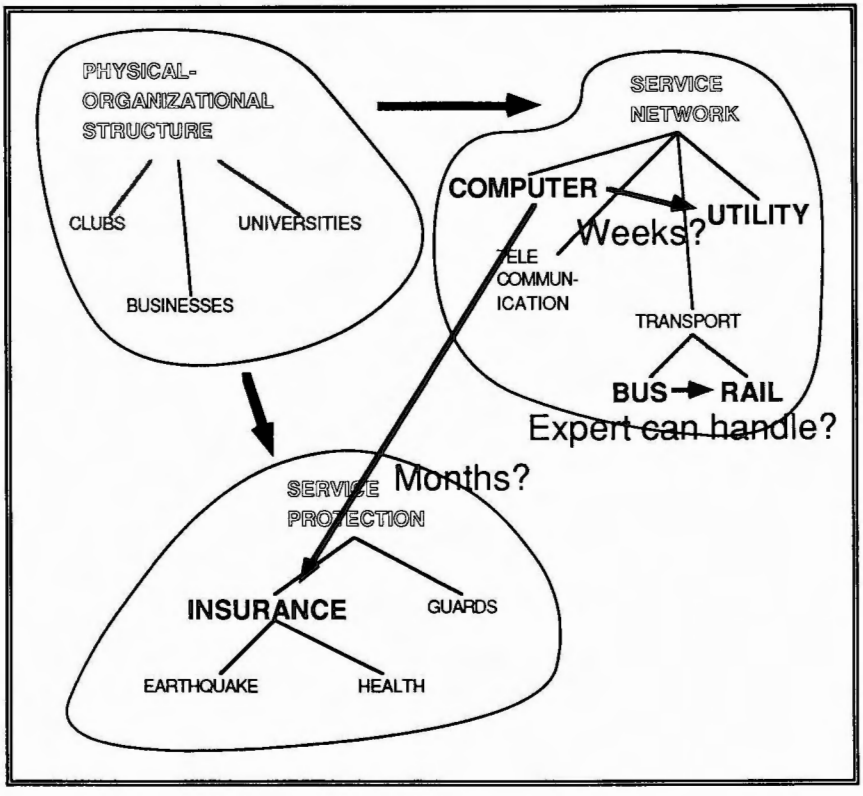


Figure 5. Generalization of TOPO's inference structure: a physical-organizational-structure (POS, e.g., a business description) is related to a service network (e.g., telecommunication network). Existing POS models could be reused in expert systems that design a different kind of system or process (e.g., insurance policy configuration).

Other researchers have advocated the analysis and decomposition of knowledge bases in terms of general or "generic" components. How do these analyses compare? We can summarize the relation between the inference operators described here, Chandrasekaran's generic tasks, and McDermott's role-limiting methods as follows:

- A *(role-limiting) problem-solving method* is a procedure composed of several *inference operators* that--through their controlled interaction--form a situation-specific model that satisfies task constraints. For example, NEOMYCIN's subtasks together implement a variant of MOLE's COVER-AND-DIFFERENTIATE method.

- Furthermore, a subtree in NEOMYCIN's subtask hierarchy (e.g., EXPLORE-AND-REFINE), which appears in multiple problem-solving methods, is packaged and distributed separately in what Chandrasekaran calls a *Generic Task.*

In conclusion, the model-construction operator perspective reveals that different researchers have pursued different levels of generality in formalizing control knowledge: single SSM operators correspond to a NEOMYCIN task; subprocedures of one or more operators correspond to a Generic Task; a complete inference procedure corresponds to a role-limiting method.

4. CONCLUSIONS

A central claim of this paper is that it is productive to view AI programming in terms of a modeling methodology that represents causal, temporal, and spatial relations in systems. From this perspective, control knowledge consists of the procedures for constructing situation-specific models. Different representations, problem-solving architectures, knowledge acquisition tools, specific expert systems, and even different areas of AI research can then be systematically related by this model-construction perspective, in terms of types of relational networks, process models, inference operators, system-domains, and modeling purposes (tasks).

The HERACLES representation reveals that each time we write a new procedure for interpreting a representation, we define new relations that classify its constructs. Thus, we find that classifications and procedures are defined in terms of each other. The representation of strategic knowledge that results from stating control knowledge abstractly, using variables in place of primitive terms, is not domain-independent, but *domain-general*, in the sense that the language or relations can be made more general than any one system being

modeled by using spatial, temporal, causal, and subtype distinctions (a perspective by which all systems can be described).

A major step is realizing that model construction recurs at three nested levels: the domain, inference procedure, and communication with the world. Specifically, the idea of a blackboard or SSM recurs at each level, which is to say that each level involves some general model of processes and an inference procedure for constructing a situation-specific representation (e.g., a model of a patient's disease, a diagnostic plan, a discourse/explanation process for communicating with the user). To see this, we need to extend the idea of a model to include classification descriptions and not just simulations. We then realize that a key contribution of AI programming is in using relational networks to represent processes, specifically developing types of classifications and state-transition networks that are linked and composed in taxonomies, causal-associational networks, and structure-function representations.

The resulting picture is very general, and its power comes not from telling us specifically how to develop a good model for some purpose, but how to relate the diverse research that has attacked specific problems. We can relate blackboards to metarules, NEOMYCIN to ABEL, heuristic classification to qualitative process simulation, XPLAIN to KNACK, and Generic Tasks to Role-limiting methods. Put another way, we can now relate representational constructs, expert systems in a given domain, inference methods, knowledge acquisition programs, and reasoning strategies, all from a model-construction perspective.

Given any expert system, we can ask, "What are the systems being modeled? What are the structure and process characteristics of this system? What kind of relational network is used to represent these structures and processes? What is the inference procedure for constructing a situation-specific model? How is this model employed by later reasoning phases, evaluated, or conveyed to the user?" Rather than simply asking about a new problem domain, "Is there real-world knowledge that allows classification?" we might ask, "Must the system be modeled as open in its interactions with its environment? Is there a known etiological hierarchy? Are there stages or developmental descriptions involving trends and frequency of behaviors? What experience have people had with this system in rebuilding, modifying, assembling it in different situations?" Thus, knowledge engineering is a form of systems analysis.

From these observations, we can conjecture where expert systems and knowledge acquisition research are headed. First, much more

attention could be placed on the recurrent macrostructures in domain models. Hierarchies and transition networks need to be viewed as objects in their own right and classified. Such classifications can then be mapped to a taxonomy of system types, according to whether the system has dynamic properties, whether a classification of interactive-historical patterns is known, and whether a behavioral (causal-associational net) or functional simulation is possible and useful.

Paralleling the approach in software engineering, indeed merging with it [14], object libraries need to be collected and shared. These objects will include and be organized by the relational language, types of process model, system type, inference operators, form of the SSM, constraints on interactions with the world, and modeling purpose. In this respect, HERACLES-DX, BB1-ACCORD, KNACK, Generic Tasks, etc. are precisely the systems that must now be studied and integrated. Automatic programming could play a role here, specifically if the SSM graph-operator representation is adopted for relating the diverse approaches.

In conclusion, a historical perspective suggests that we take care to separate the modeling contributions of AI research from specific theories of intelligence. In many respects, the techniques of knowledge engineering now have a life of their own, as they are adopted and developed by scientists and engineers who have no particular interest in robotics or learning research. Qualitative modeling should be studied, formalized, and presented as a contribution to computer science and more broadly, general systems theory. On this basis, links to operations research and software engineering will be more quickly realized and taught. Indeed, in many respects this generation of AI research may come to be viewed not so much for the particular capabilities of the programs that were developed, but for the generality of the methods, which as computational formalisms are arguably as novel and wide-sweeping in their impact as Newton's calculus.

ACKNOWLEDGEMENTS

TOPO was designed and implemented with Monique Barbanson. This paper is excerpted from "Model Construction Operators," which describes HERACLES in more detail. Partial support for this research was provided by gifts from the Digital Equipment Corporation and the Xerox Corporation.

REFERENCES

[1] Bateson, G., Steps to an Ecology of Mind, Ballentine Books, New York 1972.

[2] Chandrasekaran, B., Expert systems: Matching techniques to tasks, AI Applications for Business, Reitman ed., Ablex Publishing Corp, Norwood, 1984.

[3] Clancey, W. J., The advantages of abstract control knowledge in expert system design, American Association for Artificial Intelligence, 1983, 74-78.

[4] Clancey, W. J., Heuristic classification, Artificial Intelligence 27: 289-350 (1985).

[5] Clancey, W. J. and Bock, C., Representing control knowledge as abstract tasks and metarules, Computer Expert Systems, L. Bolc and M. Coombs ed., Springer-Verlag, Heidelberg, 1988.

[6] Hayes-Roth, B., Hewett, M., Vaughan Johnson, M., and Garvey, A., ACCORD: A framework for a class of design tasks, KSL Technical Report, Stanford University, 1988.

[7] Kulikowski, C. A., Artificial intelligence methods and systems for medical consultation, Readings in Medical Artificial Intelligence, W.J. Clancey and E. H. Shortliffe ed., Addison-Wesley, Reading, 1984.

[8] Marcus, S., Automating Knowledge Acquisition for Expert Systems, Kluwer Academic Publishers, Boston, 1988.

[9] McDermott, J., Preliminary steps toward a taxonomy of problem-solving methods, Automating Knowledge Acquisition for Expert Systems, S. Marcus ed., Kluwer Academic Publishers, Boston, 1988.

[10] McDermott, J., R1: A rule-based configurer of computer systems, Artificial Intelligence, 19(1): 39-88 (1982).

[11] Musen, M., Automated support for building and extending expert models, Machine Learning, 4:(3/4) 347-377 (1989).

[12] Nii, H. P., Blackboard systems, Artificial Intelligence Magazine, 7 (Summer): 40-107 (1986).

[13] Sowa, J., Conceptual structures, Addison-Wesley, Reading, 1984.

[14] Swartout, W. R. Explaining and justifying in expert consulting programs, Proceedings of the Seventh International Joint Conference on Artificial Intelligence, 815-823 (1981).

[15] van Lamsweerde, A., Delcourt, B., Delor, E., Schayes, C., and Champagne, R., Generic lifecycle support in the ALMA environment, IEEE Transactions on Software Engineering, 14(6): 720-741 (1988).

[16] van Melle, W., A domain-independent system that aids in constructing knowledge-based consultation programs, Ph.D Dissertation, Computer Science Department, Stanford University, 1980.

[17] von Bertalanffy, L., General system theory: Foundations, Development Applications, George Braziller Inc., New York, 1968.

An Architecture of Knowledge Acquisition by Interview based on Dynamic Analysis

Atsuo Kawaguchi [1], Hiroshi Motoda [2]
Advanced Research Laboratory, Hitachi, Ltd.
Hatoyama, Saitama 350-03, Japan.

and

Riichiro Mizoguchi [3]
The Institute of Scientific and Industrial Research, Osaka University
Ibaragi, Osaka 567, Japan.

Abstract

Interview is indispensable for eliciting new knowledge from domain experts. This paper describes *I²S-DB*, an interviewing system based on dynamic analysis that supports knowledge acquisition for logical design of databases. The characteristics of dynamic analysis include its capability to 1) identify which knowledge is lacking or erroneous by actually solving a problem, and 2) refine the knowledge base as it goes through the problem solving process. It is very effective in making ambiguous knowledge explicit by forcing the user to think along the way s/he solves the problem, and is able to handle requirements which the user did not think of at the time of the static analysis. This paper also proposes, based on our experience with *I²S-DB*, a general architecture for knowledge acquisition by interviews based on dynamic analysis, consisting of *a problem solver, watcher, debugger, interview module, learning module* and *knowledge base*. The efficiency of the knowledge acquisition can be improved by incorporating machine learning techniques. The ability to learn from failures can be included, and it may even be possible to acquire problem solving methods by analyzing the failure types.

1 Introduction

Interview is an intellectual activity that is very important for eliciting new knowledge from domain experts. An interviewer has two kinds of knowledge. One is knowledge about the domain under consideration and the other is knowledge about interviewing itself. The latter makes the interviewer an interviewing expert, and seems to be domain-independent and appears to consist of several interview knowledge primitives.

SIS, a shell for interview systems, is based on this idea[1]. It has seven question strategy primitives. Task specific interview systems can be automatically generated using interview knowledge represented using these primitives.

SIS first requests initial information to start an interview and constructs an incomplete domain model, which takes a form of a network. The sentences given by an expert are parsed and a *set of attention* is generated based on the combination of primitives. This set of attention is then analyzed and converted to questions. The domain model is then refined through *answer-attention-question* cycles.

The generality and effectiveness of *SIS* have been demonstrated through the implementation of two interview systems, *MORE*[2], a well known knowledge acquisition system for diagnostic expert systems and *I²S-LD*[3], an interview system for the logical design of databases.

The *SIS*-made interview systems only support knowledge acquisition based on static analysis. More efficient knowledge acquisition can be achieved by actually solving a problem using the acquired knowledge and refining it during the course of problem solving. *MOLE* is the knowledge

[1] e-mail: atsuo@harl.hitachi.co.jp
[2] e-mail: motoda@harl.hitachi.co.jp
[3] e-mail: mis@ei.sanken.osaka-u.ac.jp

acquisition system for diagnostic tasks that first made explicit this idea of knowledge acquisition by dynamic analysis[4].

We plan to extend this idea to a more general framework for interview-based knowledge acquisition by extending the scope of *SIS* to include dynamic analysis. As a first step, the *I²S-LD* design system was enhanced to include this feature[5]. This paper details this new system, called *I²S-DB*, and discusses, on the basis of our own experience, a general framework of knowledge acquisition by interview based on dynamic analysis.

2 *I²S-DB* Interview System

I²S-DB is a dynamic-analysis based interview system for logical database design that includes *I²S-LD* (a static-analysis based system) as one of its components. *I²S-DB* acquires domain knowledge that maps the real world into the structure of the database which the domain expert (user) wants to build. This knowledge is acquired through interview of the user by the database designer. *I²S-DB* builds a database and refines its knowledge by having the user use it.

Static analysis such as that used in *MORE* and *I²S-LD* is best suited for cases where the meta structure of the domain knowledge is well understood. What makes the static analysis difficult for the logical design of a database is

(1) It is difficult to prepare sufficient question strategies in advance.

(2) It often happens that a domain expert is unable to answer on the spot.

(3) There are problems of a type which can be noticed only in actual use.

Point (3) was a particular problem in the database design domain. It often happened that model deficiency was detected while using the implemented database.

2.1 General Procedure for Database Modeling

The general procedure for building a database is the following (Fig. 1). First, the requirements are analyzed and the domain model which can handle the information required is constructed, based on which the logical structure is designed. Here the domain model must be described by using the language for the selected DBMS (Data Base Management System). These three steps are called logical design.

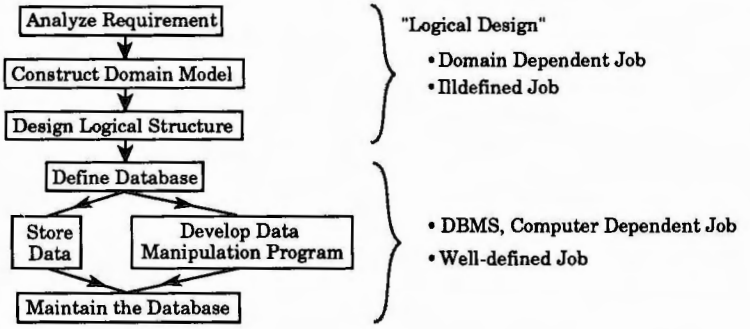


Figure 1: General Procedure for Database Modeling

Next, a framework for the database is built on a computer. This is an empty database, realized as a file. Actual data are then stored in this database. It is necessary to prepare a program to facilitate such data storage. Actual operation becomes possible by developing a data manipulation program. The final step is database maintenance where the detection of deficiency and its correction, data backups and recovery from system crashes, etc. are the major tasks.

The main problems of database construction from the point of view of knowledge acquisition from the real world include the following:

1. Analysis of required information
   - It is not clear how and what data to put into the database.
2. Database modeling
   - It is hard to extract the logical structure embedded in the data.
   - It is hard to build a model that completely satisfies the requirements.

Step 1 is the first step of database construction. Its basis is the analysis of how the data has been used, but what is difficult is to predict the new usage which becomes possible by putting the data into computer. Since the user expects some benefits from making a database, this brings about dissatisfaction.

Step 2 is an even more difficult task for an ordinary user because modeling the real world is difficult even for a database expert. It is not easy to ask such experts, however, since the number of experts is limited.

2.2 Logical Design and Dynamic Analysis

As stated earlier, there is always the problem that a domain expert may be unable to correctly answer a question however excellent the interview may be. Dynamic analysis identifies which knowledge is lacking by actually evaluating the performance of the system and then updates the knowledge base. Default values are used for what is unknown at the time of knowledge acquisition. The results are compared with the expert responses and knowledge refinement is attempted on the basis of the differences. In case of *I²S-DB* the logical structure of the prototype database corresponds to the initially acquired knowledge and its refinement is made through usage.

It should be noted that the knowledge acquired by dynamic analysis is of the same kind as the knowledge acquired by static analysis. The main difference lies in the mental status of the interviewee. In the former it is active in that s/he is trying to solve a problem whereas in the latter it is passive in that s/he is simply answering questions.

This refinement process is iterative going through *construction-problem identification-reconstruction* cycles (Fig. 2).

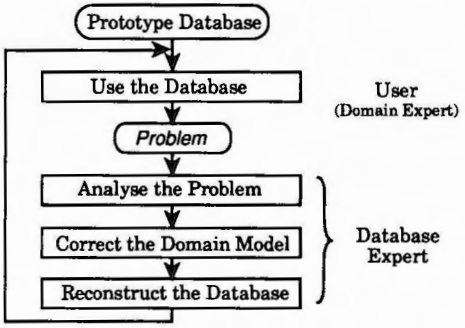


Figure 2: Procedure for Database Improvement

The prototype database is evaluated to identify problems. The problems are then analyzed and corrections are made to the domain model. The database is reconstructed based on this corrected model, and the evaluation is repeated.

From the view point of interviewing this process involves adding to and modifying the initial incomplete knowledge through its use (*i.e* via dynamic analysis) (Fig. 3). The initial insufficient knowledge is acquired first from the user who is a domain expert(Fig. 3(a)). The initial database

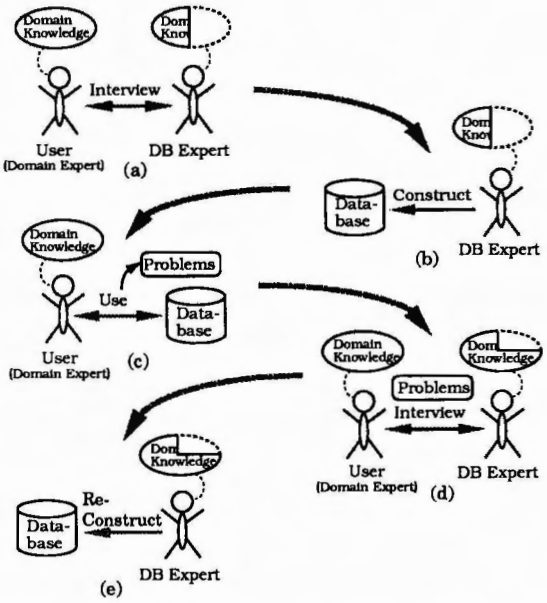


Figure 3: Database Construction and Dynamic Analysis

is then constructed and presented to the user (Fig. 3(b)). Deficiencies are detected through usage (Fig. 3(c)). The database expert modifies the initial knowledge by analyzing the behavior (Fig. 3(d)), and reconstructs the database(Fig. 3(e)).

The application of the knowledge and the analysis of the results are very important in dynamic analysis. For example, in case of the dynamic analysis in the knowledge acquisition for a diagnostic expert system, the application of the knowledge means carrying out diagnosis and the analysis of the results means comparing the predicted results with the decision made by the expert.

There are of course differences between the knowledge acquisition for databases and expert systems. In the former the knowledge to be acquired is a model of the real world which the user has subconsciously. Therefore, it is not necessarily objective and there is no general way to verify it. The only feasible way is to compare the model acquired and the model the user intended to build.

The queries for the database assumes the latest model which the user has in mind. The deficiencies in the knowledge appear in the disagreement between the model which the query assumes and the model implemented. Questions can be raised based on this disagreement and the model is accordingly modified. This simulates the behavior of a database design expert. If a natural language user interface is provided, it is possible to explore the cause of the disagreement without imposing unnecessary burden on the user.

2.3 *I²S-DB* Working Mechanism

Configuration of *I²S-DB* *I²S-DB* consists of *I²S-LD* and other modules such as a DBMS interface, a knowledge base for database construction (KB) and a Supervisor (Fig. 4). The behavior of each module is monitored by the Supervisor, and is closely related to *I²S-LD*. The initial data load program and the database modification program originally generated by *I²S-DB* are also controlled by the Supervisor.

Natural language query inputs are translated by *I²S-DB* into queries written in DML (Data Manipulation Language). The DBMS interface module then transfers the DML query to the DBMS and receives the search results. The module also includes the program library to be used by the initial data load program and the database modification program, both of which are automatically generated by *I²S-DB*.

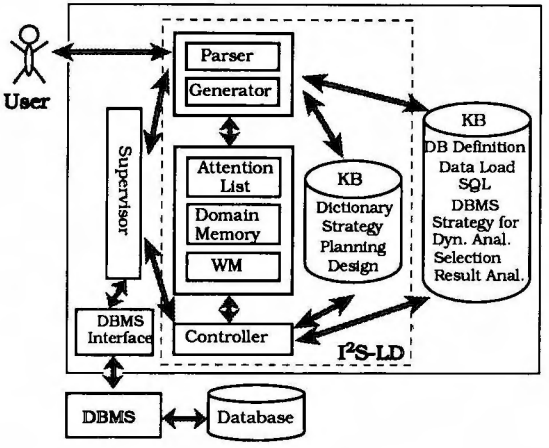


Figure 4: Configuration of I^2S-*DB*

It is necessary to have a DBMS in order for I^2S-*DB* to actually work. The present study employs a relational DBMS written in Prolog which can process DML of SQL (Structured Query Language) type.

Process flow of I^2S-*DB* The system works as shown in Fig. 5. First an interview is performed using I^2S-*LD* and the initial domain knowledge is extracted. This knowledge is stored in the form of plan structure[3], which is explained later.

After the initial domain model is constructed, the logical structure of the database is determined from this plan structure, and the database definition file, the initial data load program and the initial data file specification are created for the target DBMS. The framework of the empty database is then created by transferring the database definition file to the DBMS. At this stage the user can prepare the data following the specification, which are then stored to the database by the aforementioned program. This forms the initial prototype database. The process up to this point can be called database construction support function.

Once the prototype database is implemented, it is then offered to the user. At this stage the system functions as a natural language user interface to the database. Input queries are translated to plan structure representation. Hereafter, this plan structure representation is called "plan representation" and the plan structure representation which is a base of the current database is simply called "plan structure". The system compares the plan representation and the plan structure which is the semantics of the database and checks if the user query fits the current database structure.

If there is any discrepancy between these two, this discrepancy is regarded as a new or a latent request of the user to the database and a modification is made to the old database in the following way . Since the discrepancy is detected as attention, explained later in detail, this attention is fed to I^2S-*LD* together with the plan representation of the query and an interview can be initiated. Based on this interview, the database definition file, the data load program and its specification are created for the modified data. The user then creates a data file in accordance with the new specification as in the database construction support phase. This file is then read by the data load program and the modified database is created. This process is called database modification support function.

If there is no discrepancy between the two, the plan representation is converted to a DML representation and the query is executed. The results are displayed and the next input is prompted. If the answer is *empty*, the user can either stop using the system and add new data or try another query.

Interview during the database construction support phase corresponds to static analysis and the interview during the database modification support phase corresponds to dynamic analysis.

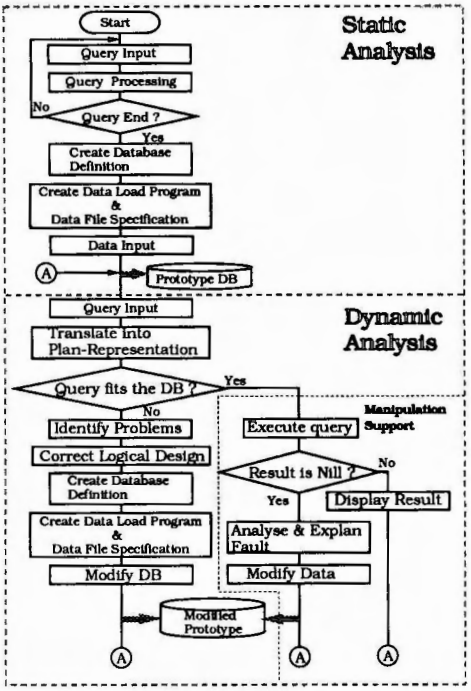


Figure 5: Process flow of *I²S-DB*

The questions raised by the system during static analysis (Query Processing in Fig. 5.) are based on the reasoning performed using four kinds of question strategies[3]. The questions during dynamic analysis are based on the failed query.

As is clear in the above description, an interview is initiated with the query in question as a clue, based on which modifications are made to the logical design. Once the clue is obtained, there is no substantial distinction between static analysis and dynamic analysis. The strategies and the mechanism are the same. The same database would have been obtained if the clue obtained dynamically were known from the beginning.

Central to the knowledge acquisition by interview is *attention generation* and *its process*. Each question and answer is carried out by generating corresponding attention and processing it accordingly. An attention is generated by the controller when a particular structure is found in the database's plan structure. Each attention has its own knowledge to raise a query to the user.

Knowledge added to *I²S-DB* The following knowledge is stored in the newly added *I²S-DB* knowledge base.

- Definition: Knowledge that defines the database using DDL (Data Definition Language) on the basis of the result of logical design
- Data Load: Knowledge used to generate 1) the specification for the initial data format and 2) the initial data load program
- Query Proc.: Knowledge to generate the attentions by comparing the plan structures which the system has and those estimated from the queries
- Strategy for Dynamic Analysis: Knowledge to resolve each attention
- SQL: Knowledge to generate the query by the DML from the plan representation
- Selection Result Analysis: Knowledge to generate the response to the user from the search results

- Modification: Knowledge to generate on the basis of the modification of the domain model 1) the specification of the format for the additional data and 2) the database modification program

Plan Structure Each activity and entity in the domain is represented by a verb frame and a noun frame, respectively. A sequence of activities forms a plan. In each noun frame its attributes and super concept are described. A domain model is constructed by linking these frames by pointers. This model is summarized in a "content frame" which lists the verbs and nouns used in the domain.

An example of such a plan structure is shown in Fig. 6. Rectangles represent frames. From the content frame we can see that the name of the database is "makerDB" and that there are two verbs and two nouns. The activity wherein a part is constructed from other parts supplied by a company is represented by two frames. Each verb frame has three slots: goal, action and precondition. Each slot indicates the state attained by the verb, the action taken by the verb and the states which must have been attained for the verb to execute its action, respectively. There are ten kinds of semantic primitives available as elements for these slots.

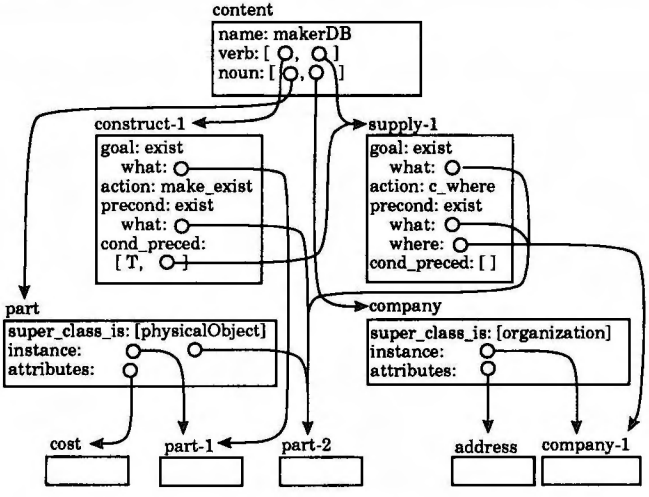


Figure 6: An Example of Plan Structure

Examples An example dialogue between the system and the user is shown in Fig. 7. In this figure (1) and (2) show the interview during the logical design, (3) shows the creation of the prototype database program, and (4) shows the initial use. Further, (5) ~ (7), (8) ~ (10) show the dialogue in case there is discrepancies. The examples used in the following explanations are all cited from this dialogue and the number in the sentences indicates the corresponding part in the dialogue.

Modification of Plan Structure Figure 8(a) shows how the database's plan structure was modified during the course of interviewing in the example shown in Fig. 7. The solid rectangle shows the plan structure created by the dialogue step (1) and the dashed rectangle is the one which has been added at step (5). Figure 8(b) shows the plan representation created from the query of ⑪.

First, a database consisting of the relations: supply, stock and part is created in steps (1) to (3) (①②③). The query (⑥) at step (4) could be answered by the domain model in the database. The noun "company" (⑦⑧) which was not in the plan structure appeared in the query at step (5), and the corresponding new frames were then created. The supply preconditions p:exist: where: were also added at this stage.

The plan representation for the query of ⑪ is to explain how attentions are generated and the plan structure is modified. By comparing the plan representation and the plan structure which had been formed up to this point, it was noticed that there were no corresponding "construct" and

```
|?- interview.
Please enter the name of your database.
> makerDB
Let's start designing conceptual schema of maker_db.
Please enter an information requirement.
> Find the name and quantity of parts which are stocked. ..... ①
Is 'part' an instance of an entity ?
> No.
Please enter an information requirement.
> Find the name of parts which are supplied. ..... ②
Please enter an information requirement.
> None.
Let's discuss the following noun:
N: part.
I think it's one of the physicalObject.
Is it right ?
> Yes.
Please choose the attributes of the part from:
1. size  2. color  3. price  4. (others)  (.<nl> for none)
> 2, 3.
Ok, I start designing conceptual schema.
Wait a moment, please.
Please choose the key attributes of the part from:
1. name  2. color  3. price
Which ?
> 1.
```

(1)

```
Designed conceptual schema is the following: ..... ③
  type(part,price,int)
  type(part,color,char)
  type(part,name,char)
  rel(stock,[part],[quantity])
  key(stock,[part])
  hie(stock,part,[],1:1)
  rel(supply,[part],[])
  key(supply,[part])
  hie(supply,part,[],1:1)
  selEnt(part,[],[name,color,price])
  key(part,[name])
yes
|?- do_format.
Ok !! I start to make the load program. ..... ④

yes
|?- do_load.
Ok ! Load start !!
Completed to make the first data base ! ..... ⑤

yes
|?- do_use.
Please enter a query sentence.<none. for end.>
> Find the name and color of parts which are stocked. ..... ⑥
select name, color from  part, stock where
  stock.part=part.name

-----------------
name      color
-----------------
nail      black
wood      blown
body      white
furniture yellow
```

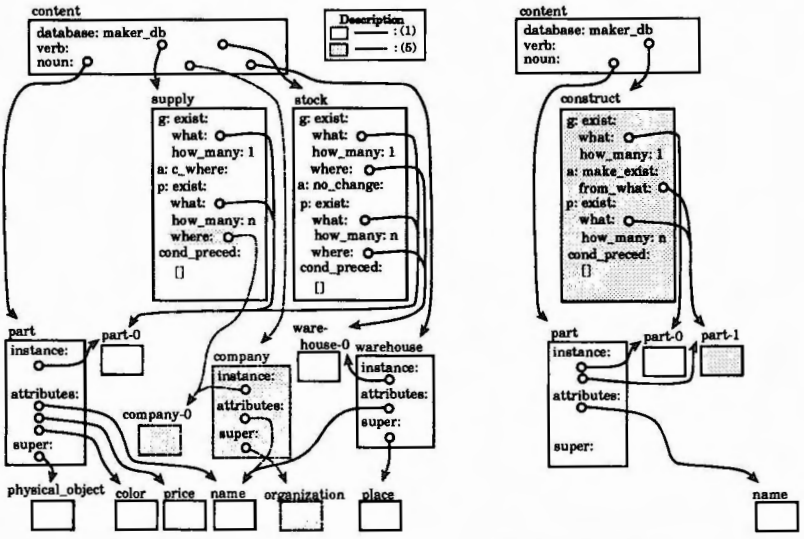
(2) (3) (4)

Figure 7: Dialogue with *I²S-DB* (part 1)

```
Please enter a query sentence.<none. for end.>
> Find the name and quantity of parts which are
    supplied from marui. ..... ⑦
Is 'marui' an instance of an entity ?
> Yes.
Please enter the entity name of 'marui'.
> Company. ..... ⑧
Let's discuss the following noun:
N: company.
I think it's one of the organization.
Is it right ?
> Yes.
Please choose the attributes of the company from:
1. address  2. (others)  (.<nl> for none)
> .
I think you should reform the following table:
1: reform 'rel(supply,[part],[])' to
      'rel(supply,[company,part],[quantity])'
Is it right ?
> Yes.
Please choose the hierarchical relation of company and
part of 'supply' from:
1. 1:1  2. 1:n  3. n:1  4. m:n
Which ?
> 3.
Ok !! I start to make the load program. ..... ⑨

yes
|?- do_modify.
Ok !  Load start !!
Completed to modify the data base ! ..... ⑩

yes
|?- do_use.
Please enter a query sentence.<none. for end.>
> Find the name and quantity of parts which are
  supplied from toyota.
Is 'toyota' an instance of a company ?
> Yes.
select quantity, part from supply where company=toyota

----------------
quantity part
----------------
500      body
300      tire

Please enter a query sentence.<none. for end.>
> Find the name of parts which are constructed
  from tire. ..... ⑪
I think you should reform the following table:
1: add 'rel(construct,[part(0),part(1)],[])'
Is it right ?
> Yes.
Please choose the hierarchical relation of part(0) and
 part(1) of 'construct' from:
1. 1:1  2. 1:n  3. n:1  4. m:n
Which ?
> 4.
Ok !! I start to make the load program. ..... ⑫

yes
|?- do_modify
Ok ! Load start !!
Completed to make first data base ! ..... ⑬

yes
|?- do_use.
Please enter a query sentence.<none. for end.>
> Find the name of parts which are constructed
  from tire.
select part_0 from construct where part_1=tire

--------
part_0
--------
car

Please enter a query sentence.<none. for end.>
> None.
```

(5) (6) (7) (8) (9) (10)

Figure 7: Dialogue with *I²S-DB* (part 2)



(a) plan structure up to step (7)

(b) plan representation for the query of (11) in step (8)

Figure 8: Modification of Plan Structure

"part-1" frames in the plan structures. This caused *I²S-DB* to generate two kinds of attention: *frame_not_exist* and *class_not_exist* and the new frames were created (step (8)) according to the procedures described below.

Attention and its processing Analysis with *I²S-DB* suggested that there are five (two of which appeared in the above explanation) kinds of attention which are generated during the comparison of plan representation and plan structure. How these are processed is described below.

(1) frame_not_exist

This attention is generated when a verb frame in the query's plan representation does not exist in the database's plan structure. This corresponds to the situation where a query about an activity not known to the domain model is entered (Fig. 7(1) is an example of such a query).

In this case there are two possibilities. One is that there is no frame for a verb with that meaning, and the other is that a synonym was used. *I²S-DB*, therefore, searches for the synonyms of the verb in the plan structure and processes according to the result.

Activities which verbs imply are classified in the database plan structure according to their action's semantic primitives, and it is further subclassified by their goals and preconditions. Therefore, the search is made in the following order.

1. compare by action.
2. compare by goal.
3. compare by precondition.

If the search is a success, the verb found is shown to the user for confirmation. If it is a valid synonym, the frame is renamed. Otherwise, the verb is added to the plan structure.

(2) slot_not_exist

This attention is generated when a slot in a verb frame in the query's plan representation does not exist in the corresponding verb frame in the database's plan structure. It is also generated when a link between the verb and some other verb in the query's plan representation does

not exist between the corresponding verbs in the database's plan structure. This implies that a new attribute is needed in the domain model.

I²S-DB asks the user if this attribute is to be added or not. If the answer is *yes*, it is added to the verb.

(3) slot_not_same

This attention is generated when the value of a slot in a verb frame differs in the query's plan representation and the database's plan structure (Fig. 7⑦ is an example of such a query).

There are two cases. One is that a new relation or a modification of an existing relation is needed. The other is that either 1) the entity name which corresponds to the value used in the query is a synonym of some existing entity in the plan structure, or 2) the entity name subsumes (or is subsumed by) some entity in the plan structure (for example, *factory* is subsumed by *company*).

I²S-DB checks if there is a synonym or a subsumption relationship and, if there is, replaces the slot value of the query's plan representation so that the search becomes possible. If not, an interview is initiated and a new verb frame is created if needed.

(4) class_not_exist

This attention is generated when a noun frame in the query's plan representation does not exists in the database's plan structure (company in Fig. 7⑧ is an example of such a noun). There are two cases. One is that an entity unknown to the domain is entered and the other is that a synonym of the known noun is entered. *I²S-DB* interviews the user and adds a new noun frame if needed.

(5) attr_not_same

This attention is generated when an attribute of a noun frame in the query's plan representation is not an attribute in the corresponding noun frame in the database's plan structure. This implies that a new attribute is needed. *I²S-DB* asks the user if the attribute should be included with the noun and adds the attribute to the noun if so.

2.4 Knowledge Acquired in *I²S-DB*

The following knowledge is obtained in *I²S-DB*.

- Entity(noun) and its attribute(s)
- Relation(verb) and its attribute(s)
- Link between entity and relation
- Synonyms among entities and part_of relation
- Link between two relations
- Synonyms for relation

The first part of knowledge acquisition is carried out without having a prototype database. The user must answer the interview without using the database. The second part uses the database. It is this part which distinguishes *I²S-DB*, *i.e.* knowledge acquisition by dynamic analysis. Actually using the database is a good stimulus for the user. There are requirements which can be clarified or emerged only after the usage. *I²S-DB* helps the user detect these requirements and gradually improves itself.

3 Knowledge Acquisition and Problem Solving

3.1 Relation to Other Works

In this section the brief discussion is made comparing *I²S-DB* to previous related works including

- *TEIRESIAS* [6]
- *MORE* [2]
- *MOLE* [4].

TEIRESIAS is one of the earliest knowledge acquisition support system. It was developed as a debugger for *MYCIN*. It is invoked when *MYCIN* has made a misjudgment and supports the input of new rules and the modification of the existing rules using two kinds of meta knowledge called a rule model and a data structure scheme.

MORE is a knowledge acquisition support system which resulted from the development of *MUD*, a diagnostic expert system in the drilling fluids domain. It maintains the acquired knowledge as a network with three kinds of nodes: symptom, hypothesis and condition, and interviews are performed based on eight question strategies.

MOLE is the successor of *MORE*. *MOLE* allows uncertainty in the domain model and increases the interviewing capability by adopting dynamic analysis. It first constructs the domain model using question strategies similar to *MORE*, and then builds a diagnostic expert system and refines the domain model based on diagnostic failure.

What is common to all of them is that the problem solving function is utilized in the interview in some way a(Fig. 9). *TEIRESIAS* uses *MYCIN*. *MORE* has the eight question strategies. *MOLE* constructs a problem solving system and use it. *I²S-DB* also does interview using the constructed database.

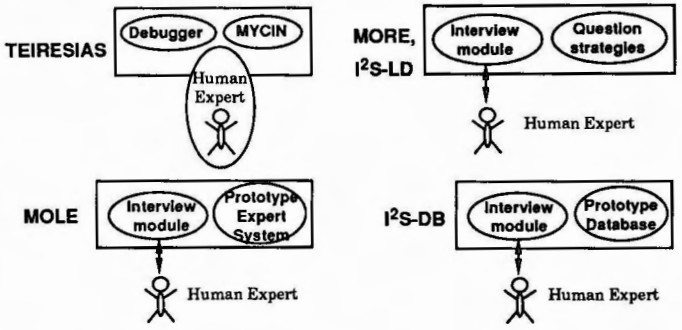


Figure 9: Interview System and Problem Solving Function

3.2 Problem Solving and Interview

Needless to say, the purpose of an interview is to acquire knowledge which is useful in problem solving. The fundamental challenge is thus to decide what kind of knowledge to ask for in what situations. It affects whether the interviewee is able to answer or not when the knowledge asked for is complicated or ambiguous. The best way to decide which knowledge is required is to attempt to solve the problem. Once the necessary knowledge is analyzed and acquired, no difficulty arises when similar problems are encountered in the future. The performance of the system increases gradually.

Good questions pull the interviewee into the problem solving process. The interviewee becomes conscious about the knowledge s/he is using. It is thus necessary to present the problem solving process and related suggestions .

The aforementioned four systems do interview in this way. The question strategies of *MORE* confirm whether the knowledge acquired up to a certain point is enough to solve the problem. *MOLE* and *I²S-DB* solve problems and based on the difficulties that arise initiate interviews.

In the case of *TEIRESIAS* it can be looked as a debugger of *MYCIN*. Detection of the failure of problem solving, identification of the wrong knowledge and its modification are all left to the expert. *TEIRESIAS* only gives suggestions to prevent possible errors in making rule modification. It also suggests conditions and attributes which seem necessary. The method used here is statistical and is not guaranteed to work in all situations.

In case of *I²S-LD* four kinds of question strategies are used to predict possible troubles which may take place when a database is constructed by the current domain model. *I²S-DB* constructs a database and sees what happens.

3.3 Learning and Interview

The question strategies used in *MORE* are based on the insights made in advance, which makes actual problem solving unnecessary. For example, the strategy *Differentiation* is used to differentiate two hypotheses for which the same symptom appears. Indeed, if this assumption is used in diagnosis, two hypotheses will be derived for a single symptom. This kind of problem can easily be predicted in advance.

The eight strategies in *MORE* have made it possible to ask effective questions by relatively simple pre-analysis. In *MOLE* and *I²S-DB* the interview during the problem solving phase can be minimized by performing the static analysis first.

In general it is not efficient to do interview while solving a problem. If the interviewer were lost in thought, it would be a very lengthy interview.

The efficiency can be improved by chunking the problem solving process in the past and using it for the problem solving at a later interview. This essentially involves adding a learning mechanism to an interview system. It is a function that makes it possible to use in future interviews knowledge acquired in the past.

This learning mechanism is strongly related to EBL[7]. Domain theory in EBL corresponds to the initial knowledge given to the interview system. Explanation in EBL corresponds to the problem solving and interview processes. What is obtained is the updated domain theory. Generalizing the interview process can also improve efficiency.

4 General Architecture of Interview System

4.1 Proposed Architecture

Based on the discussion in 3 the general architecture of interview systems can be described as follows (Fig. 10).

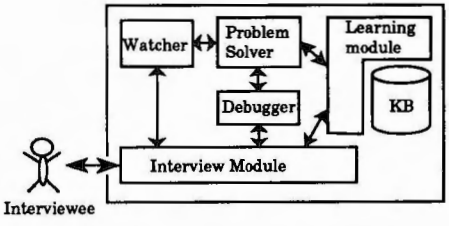


Figure 10: General Architecture of Interview System

The roles of the modules in this architecture include:

Problem Solver This module solves a problem during the course of interview. This must be suited to the nature of the task.

Watcher This module monitors the behavior of the problem solver. When it becomes impossible to continue, it passes the following information to the interview module and prompts for questions:

- problem solving history
- requirement for lacking knowledge.

Debugger This module controls the problem solver and makes it possible to solve a partial problem. This is used by the interview module to control the problem solving and test the acquired knowledge.

Interview Module This module controls the overall behavior of the whole system. It monitors the problem solver through the watcher and performs the interview with the interviewee. It uses the debugger, tries to solve the problem, and asks questions to add to acquired knowledge.

Learning Module and Knowledge Base The learning module monitors how the knowledge is used and chunks the problem solving and interview process for efficiency by making links among the pieces of knowledge. The knowledge base stores the knowledge necessary for interview as well as the acquired knowledge.

4.2 Analysis of the Failure in Problem Solving

As stated above, a knowledge acquisition support system should have both problem solving and interviewing functions. Knowledge acquisition system based on dynamic analysis explicitly uses the problem solving function in acquiring knowledge.

Given this way of thinking, the following important items should be discussed.

- failure detection in problem solving
- failure type classification and resolution

Failure detection It is fairly easy to detect the failure for problems which *MOLE* is intended to treat, *i.e.* heuristic classification problems. The situation is the same for logical design of database. There is a well defined criterion, *i.e.* the search is executable.

In general the detection of failure is easy for domains where the criteria to evaluate the results are well defined. Conversely it is possible to use the ambiguity of failure detection to clarify the criteria. For example, if the design results are hard to evaluate, this may imply that the design specification is ambiguous and the specification can be a target for acquisition.

Failure classification and resolution If the failure is easily classified and the resolution is definitely determined, knowledge acquisition by dynamic analysis is relatively easy. *MOLE* and *I*^{2}*S-DB* fall in this category.

It may be very difficult in general, however, to analyze the types of failure in advance and determine what actions to take. Nevertheless, domain independent resolutions for failures are conceivable based on the characteristics of the problem solver. Taking a production system as an example, the ways one can fail to solving problems can be classified in a domain independent manner as

- There is no rule to fire.
- The rule should have fired, but actually it did not.
- The rule should not have fired, but actually it did.

This is also true in Prolog. The algorithmic debugging system is based on this idea[8].

Therefore, the problem is whether these classifications and the corresponding resolutions are properly defined in a form which is dependent on problem solvers but not on domains. Such knowledge can also be regarded as general knowledge for interviewing.

Resolutions can also be used in static analysis. The question strategies in *MORE*, for example, correspond to the resolutions for failure.

What is said here is related to the generic task methodology[9], but no explicit method is given in this methodology to derive strategies from problem solvers.

4.3 Link between Interview and Problem Solving Modules

The following functions are required between the interview module and the problem solver to perform dynamic analysis:

- Delivery of the problem solver's results

- Delivery of the problem solver's behavior
- Resumption of the problem solving process.

The results can be delivered to the interview module either continuously or discretely, *i.e.* only when the problem solving gets stuck. In the former case, the dialogue may become complicated because each piece of incoming information is a clue to knowledge acquisition. On the other hand in the latter case, the problem solving history may become long and its analysis may become complicated. Therefore, some guideline is required as to how often the results should be transferred.

It is necessary to administer the dependency of the problem solving process on the knowledge base such that the modification of the knowledge base can efficiently be reflected back to the problem solving process. Unfortunately *I²S-DB* does not have this capability , and each time a modification is requested, everything has to be recreated according to the new specification and load program both of which are automatically generated by *I²S-DB*. This is not efficient and needs improvement.

4.4 Dynamic Analysis and Problem Solving Method Acquisition

Dynamic analysis focuses on the knowledge which causes failure in problem solving and acquires new knowledge. Would it not be possible to acquire a problem solving method pertinent to the domain by looking carefully at the failure mechanism? It seems possible while acquiring the knowledge to store how the failure takes place each time such a thing occurs and generalize the failure pattern to infer the problem solving structure by the following process:

1. prepare the problem solving primitives which the problem solver uses
2. represent the problem solving process by these primitives
3. extract the common failure pattern from the accumulated failure examples.

Of course the frequently used problem solving method can also be identified by extracting general features from successful examples. However, this is not enough. Learning from failure is also important.

4.5 Failure-Driven Growing System

A common problem for current expert systems is the rapid performance degradation that appears in regions they were not designed to work (*i.e.* lack of robustness). *I²S-DB*, for example, is totally incompetence with queries which have not been taken into account at the time of design. However, continuous growth of the database becomes feasible in principle by dynamic analysis provided a more efficient update mechanism is employed.

Smooth resumption of the problem solving process becomes important to allow for continuous growth. In an extreme case the system can be used in actual practice when its performance reaches the minimum required level and can be left alone to grow as it is used. The requirements for such an approach include:

1. to secure the minimum required initial performance
2. to secure a domain expert
3. to offer knowledge acquisition support appropriate for the domain expert.

To make dynamic analysis possible is equivalent to allowing the system to recognize when it cannot solve the problem. Since existing expert systems do not have this capability, they sometimes behave as if they have solved problems which they actually cannot.

5 Conclusion

An architecture of knowledge acquisition by interview which adopts dynamic analysis is proposed based on our own experience with *I²S-DB*, a system that supports knowledge acquisition for logical design of a database.

The proposed architecture consists of the following six parts: *problem solver, watcher, debugger, interview module, learning module* and *knowledge base.*

Dynamic analysis identifies which knowledge is lacking or erroneous by evaluating the performance of the system while actually solving a problem and refines the knowledge as it goes through problem solving process. It is very effective in making ambiguous knowledge explicit by putting the user in a problem solving environment. It can handle a requirement which the user can not even think of in advance at the time of the static analysis.

Problems for future work include:

1. The efficiency of knowledge acquisition can be improved by chunking previously experienced problem solving process and using it at a later process, thereby speeding up problem solving. The number of questions can be reduced by chunking and generalizing interview processes.
2. It is important to analyze failure in problem solving, and classify the failure type and specify an appropriate resolution in a form which is dependent on problem solvers but not on domains.
3. Some guideline is required as to how often the results should be transferred from the problem solver to the interview module. This should be determined based on the balance between the complexity of dialogue and difficulty in analyzing problem solving history.
4. It is necessary to administer the dependency of problem solving process on the knowledge base such that its modification can efficiently be reflected to the problem solving process.
5. It may be possible to acquire a problem solving method pertinent to the domain by looking carefully at the failure mechanism if the problem solving process can be represented by a set of known primitives.
6. The continuous growth of the performance is feasible in principle by dynamic analysis provided an efficient updating mechanism and smooth resumption of problem solving process are secured. In an extreme case the system can be used in actual practice when its performance reaches the minimum required and can be left to grow as it is used on the basis of a failure-driven growing mechanism.

Acknowledgment

Authors are grateful to Dr. Mark Klein at ARL, Hitachi, Ltd. for the comments to the early draft of this paper.

References

[1] **Kawaguchi, A., Mizoguchi, R., Yamaguchi, T. and Kakusho, O.** , "*SIS*: A Shell for Interview Systems", *Proc. of IJCAI'87*, 1, pp. 359-361, 1987.

[2] **Kahn, G., Nowlan, S. and McDermott, J.**, "Strategies for Knowledge Acquisition", *IEEE PAMI*, PAMI-7(5), pp. 511–522, 9, 1985.

[3] **Kawaguchi, A., Taoka, N., Mizoguchi, R., Yamaguchi, T. and Kakusho, O.**, "An Intelligent Interview System for Conceptual Design of Database", *Proc. of ECAI'86* , 2, pp. 39-47, 1986.

[4] **Eshelman, L. and McDermott, J.**, "MOLE: A knowledge acquisition tool that uses its head", *Proc. of AAAI'86*, pp. 950-955, 1986.

[5] **Numata, K., Kawaguchi, A., Mizoguchi, R., Yamaguchi, T. and Kakusho, O.**, "Development of a Database Construction/ Manipulation Support System based on an Intelligent Interview System *SIS* (in Japanese)", *Proc. of IPSJ*, 4L-11, 1986.

[6] **Davis, R. and Lenat, D. B.**, *Knowledge-Based Systems in Artificial Intelligence*, MacGrow-Hill, 1982.

[7] **Mitchell, T. M., Keller, R. M. and Kedar-Cabelli, S. T.**, "Explanation-based generalization: A unifying view", *Machine Learning*, pp. 47–80, Springer-Verlag, 1986.

[8] **Shapiro, E. Y.**, *Algorithmic Program Debugging*, The MIT Press, 1982.

[9] **Chandrasekaran, B.**, "Generic Tasks in Knowledge- based Reasoning: High-Level Building Blocks for Expert System Design ", *IEEE Expert*, pp.23-30, Fall, 1986.

INDUCTIVE KNOWLEDGE ACQUISITION FROM STRUCTURED DATA

J. R. Quinlan

Basser Department of Computer Science
University of Sydney
Sydney NSW Australia 2006

Inductive techniques for extracting rules from example objects, as commonly applied, face difficulties when the objects are themselves structures. This paper illustrates the use of simple techniques and shows one method of extending them to cover structured data.

1. INTRODUCTION

The knowledge bases that power expert systems today are significantly larger than those of the early 1980s. A few years ago, an organisation's first foray into ES technology might have resulted in a demonstration system with a hundred or so rules, but the figure nowadays is more likely to be closer to the thousand mark. This growth is alarming because the problems associated with codifying rules by interviewing experts, identified ten years ago by Feigenbaum, have not diminished proportionately.

Several approaches to resolving this dilemma have been suggested. One idea is to systematise the knowledge engineering process in an attempt to increase the productivity of the knowledge engineer and participating experts. The recent work of Debenham [5] in this area is receiving considerable attention. Another technique is to develop a meta-theory of expertise in a circumscribed class of application domains (such as the diagnosis of equipment malfunction) and to provide a knowledge representation formalism and framework that is tailored to such applications. Marcus [9] contains case studies illustrating this idea.

One of the oldest and most widely-used knowledge acquisition aids uses inductive generalisation processes to derive expert-level knowledge from sets of examples. If the task is to classify a mushroom as poisonous or edible, for instance, it is often possible to develop effective rules from analysis of descriptions of mushrooms whose toxicity is known. Quinlan [20] gives an account of this method of drafting knowledge bases. Michie [12] presents some recent advances which have the effect of improving the knowledge-like nature of rules derived in this way.

While these inductive techniques are undoubtedly effective, they often depend on the application being cast in a simple framework in which objects are described by fixed properties and the task is one of classifying unseen objects in terms of these properties. This works well when a suitable set of properties or attributes appropriate for such classification is known. It is much less straightforward when the objects themselves are complex, structured entities such as assemblies of subcomponents.

The next section reviews briefly the two principal approaches to deriving classification rules from attribute-value descriptions of objects. One approach is illustrated by a case study involving design parameters of bridges. An application is then introduced that does not fit into the attribute-value formalism and the paper discusses ways in which the simple learning methods can be adapted to handle it. This leads to an outline of a new system, FOIL, that analyses structured objects described by relations and generates rules in Horn Clause logic. Although this approach is experimental and is still being refined, it has proved adequate for many prototypical learning tasks described in the literature; results on some of these are presented.

2. ATTRIBUTE-BASED INDUCTION

In an attribute-value formalism, *objects* are described in terms of a fixed collection of *attributes*. An attribute might be *discrete*, with a specified set of nominal values, or *continuous*, with a numeric value. Each object belongs to one of a small number of mutually exclusive and exhaustive *classes*. The learning task can be stated as: given a training set of objects whose classes are known, find a rule for predicting the class of an unseen object as a function of its attribute values.

2.1 Divide-and-conquer methods

Several learning systems express this classification rule in the form of a *decision tree*, examples being CLS [7], its descendant ID3 [17, 18], CART [2] and ASSISTANT [3]. At a somewhat oversimplified level, these systems use a common method to construct a decision tree from a training set of objects:

- If all training objects belong to a single class, the tree is a leaf labelled with that class.
- Otherwise,
  - select a test based on one attribute,
  - divide the training set into subsets, each corresponding to one of the possible (mutually exclusive) outcomes of the test, and
  - apply the same procedure to each subset

The effectiveness of this divide-and-conquer strategy depends critically on the choice of a test to partition the training set. A good choice should assist the development of single-class subsets which do not require further division. It has proved possible to design simple information-based or encoding-length heuristics that provide effective

guidance for division, to the extent that greedy hill-climbing techniques are usually adequate. Examples of such heuristics appear in [18, 20].

2.2 Covering methods

Another group of algorithms, as exemplified by the AQ family [10, 11], construct a classification rule in the form a disjunctive Boolean expression for each class. The core of this method can be summarised as:

- Find a conjunction of conditions that is satisfied by some objects in the target class, but no objects from another class;
- Append this conjunction as one disjunct of the rule;
- Remove all objects that satisfy this conjunction and, if there are remaining objects of the target class, repeat the procedure.

Members of the AQ family find a suitable conjunction by generalizing a single seed object from the target class. Specifically, they carry out a beam search on the space of subsets of the primitive descriptors of the seed object, looking for simple conjunctions that "cover" many objects in the target class. A later algorithm, CN2 [4], shares this core but uses a divide-and-conquer method very similar to that employed by the decision tree systems to construct a suitable conjunction.

2.3 An example

Inductive learning has an excellent track record in drafting knowledge bases for engineering applications of ES technology. Leech [8] describes an expert systems for controlling a nuclear fuel process whose rules were derived from examples. The same technique was used to construct a 2500-rule knowledge base for the design of gas-oil separator vessels as reported in *Expert Systems User*, August 1986 (pp 16-19). Adeli and Balasubramanyam [1] employ inductive methods to formalise the top-level knowledge in a system to design bridge trusses, using training objects obtained from a simulation.

As an illustration of what can be achieved with these simple learning formalisms, a decision-tree method was used to analyse data, assembled by Mary Lou Maher and her co-workers, from about one hundred bridges in the Pittsburgh area. In this application there are eight descriptive attributes relating to bridges:

- the *river* over which the bridge was constructed
- the *location* of the bridge (a continuous value)
- the *purpose* for which the bridge was constructed: for a highway, railroad, walkway or aqueduct
- the *number of lanes* on the bridge, if applicable
- the bridge's *year of erection*

Design Descriptor	Predictive Accuracy	
	Decision Tree	Default Rule
Material	85%	75%
Type	60%	42%
Through or deck	85%	85%
Span	64%	58%
Relative length	68%	56%

Table 1: Results on bridge data

- whether or not *vertical clearance* governs the design of the bridge
- the *length* of the bridge

The design of the bridge has five dimensions:

- the *material* from which the bridge was constructed: wood, iron or steel
- the *type* of the bridge: wooden, suspension, simple-t, arch, cantilever or cont-t
- whether the bridge is *through* or *deck*
- the length of the *main span* of the bridge: short, medium or long
- the *relative length* of the main span to the channel width: S, F or S-F

Each of these latter can be regarded as a class, i.e. a small number of discrete categories into which bridges are divided. The task of predicting the design dimensions of a bridge from its descriptive attributes can thus be transformed into five classification tasks.

The above data was presented to C4.5, an inductive learning program of mine that grows decision trees. Table 1 summarises the accuracy on unseen cases of the five decision trees, one corresponding to each design dimension. (These accuracies were obtained from a cross-validation study in which the available data was divided into ten subsets and, for each subset in turn, a tree was derived from all data except that subset and then evaluated on the subset.) The table also gives the accuracy of the simple default rule in which the value of each design dimension is always predicted to be the most frequent value of that dimension. It is clear that the decision trees perform at a level considerably better than the default rule for three of the five design dimensions and about the same on the others.[1] The trees must therefore contain elements of useful predictive knowledge. If it was intended to construct an expert system for deciding on design specifications for bridges built in Pittsburgh, the decision trees above could

[1]In this particular study, accuracy was limited by the relatively small number of training objects available and the amount of information missing from them. As with all real-world data, some attribute values of many of the hundred-odd bridges are not known. Missing data causes problems both during tree formation and when the decision tree is used to classify unseen objects.

erected after 1874 : *steel*
erected up to 1874 :
| location > 28 : *wood*
| location ≤ 28 :
| | erected up to 1857 : *wood*
| | erected after 1857 : *iron*

Figure 1: Derived decision tree for *material*

provide an initial starting point for the knowledge base. As an example, the decision tree found for predicting the material of a bridge appears in Figure 1.

3. MOVING TO A RELATIONAL FORMALISM

Although these learning methods have been used to effect in a great variety of applications, they are limited by a lack of expressive power in the languages used to describe objects and classification rules. In the first place, the methods require that an object be specified by its values of a fixed number of attributes. This is clearly inappropriate when an object may have a varying number of component parts or when the interesting things about an object are its relationships to a variable number of other objects. In the second place, the methods develop classification rules that are couched in terms of the same fixed properties. There is no way to express a condition of the form, "the object to be classified has relationship R to some other object".

Both difficulties are illustrated by the following task, based on the network shown in Figure 2. Suppose we attempted to describe a node in such a network by its links to other nodes. We could not draft a fixed set of attributes such as "attribute 1 gives the first node to which this node is linked, attribute 2 the second" and so on because there is no upper limit on the connectivity of a node in such networks; no matter how many attributes we had, we might need one more. Secondly, there is no way to express a

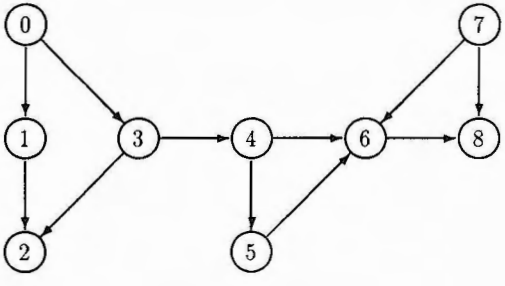


Figure 2: A network

learned rule of the form "node X can reach node Y if X is linked to some node Z and Z can reach Y".

A natural way to represent information contained in structured objects is as a collection of *relations*. A relation is associated with a k-ary predicate $P(X_1, X_2, ..., X_k)$ and consists of the set of k-tuples of constants of the form $\langle c_1, c_2, ..., c_k \rangle$ that satisfy the predicate. In the network example of Figure 2, the linkage of nodes can be captured by the relation

linked-to: $\{\langle 0,1\rangle, \langle 0,3\rangle, \langle 1,2\rangle, \langle 3,2\rangle, \langle 3,4\rangle, ..., \langle 7,8\rangle\}$

Similarly, first-order logic seems an appropriate vehicle for expressing what is learned about a relation. In this paper, the target language for learned concepts will be "pure" Prolog extended to allow negation, i.e., a collection of clauses of the form

$C \leftarrow A_1,\ A_2,\ ...,\ A_n$

which is interpreted as, "if $A_1 \wedge A_2 \wedge ... \wedge A_n$ then C". In such a clause, C is a predicate and each literal A_i is either a predicate or the negation of a predicate. We will follow the usual Prolog convention of starting variables with a capital letter. As above, a rule describing connectedness in our network might be

can-reach$(X, Y) \leftarrow$ linked-to(X, Z), can-reach(Z, Y)

3.1 Simplified description of FOIL

We now introduce a new system called FOIL that extends some of the ideas of attribute-based induction to this richer descriptive formalism. This outline is not intended as a specification of the system, which may be found in [22]; rather, it gives an overview of the way FOIL works and some of its key ideas.

The learning task addressed by FOIL is to find, for one relation at a time, a collection of Prolog clauses which defines that relation in terms of itself and the other relations. Consider a target relation based on a k-ary predicate $P(X_1, X_2, ...X_k)$. Instead of a set of objects we have a *bindings set* containing constant k-tuples, each tuple representing a value assignment to the variables $X_1, X_2, ..., X_k$. By analogy with an object's class, each of these tuples is labelled with a $\oplus$ or $\ominus$ to indicate whether or not it is in the target relation. The $\oplus$ tuples are just the set of tuples known to be in P. The $\ominus$ tuples can either be given explicitly or, more usually, consist of the complement of the $\oplus$ tuples. This corresponds to the closed-world assumption in Prolog (and databases): if a tuple is not explicitly given as a component of the relation, then it is not in the relation.

At the outermost level, FOIL employs a covering strategy similar to that of AQ:

- Initialise the global bindings set to contain all relevant constant tuples, some labelled $\oplus$ and some $\ominus$.
- Until there are no $\oplus$ tuples in the global bindings set:
  - Find a clause that characterises part of the target relation.

- Remove all tuples that satisfy the right-hand side of this clause from the global bindings set.

In its inner loop, FOIL uses a divide-and-conquer approach to grow a Prolog clause of the form

$$P(X_1, X_2, ..., X_k) \leftarrow A_1,\ A_2,\ ...,\ A_n$$

that characterises some part of the relation P. This inner loop also has a structure:

- Initialise the bound variables to $\{X_1, X_2, ..., X_k\}$, the bindings set T to the current global bindings set, and set $i = 1$.
- While this local bindings set T contains $\ominus$ tuples:
  - Find a literal A_i for the right-hand side of the clause.
  - Produce a new bindings set N based on those tuples in T that satisfy A_i.
  - Add any new variables introduced by the literal to the bound variables, set $T = N$, $i = i + 1$, and continue.

The choice of literal at each iteration of this inner loop uses an information-based heuristic similar to that employed in ID3. Each possible literal is evaluated by determining how many of the existing $\oplus$ tuples would be represented in the new binding set, and how much information would be gained about each as measured by the increased concentration of $\oplus$ tuples in this set. A good literal to choose would be satisfied by many of the existing $\oplus$ tuples and few of the $\ominus$ tuples, so that the new binding set would consist largely of $\oplus$ tuples.

When choosing the literal A_i, FOIL considers almost all tests of the form

$$Q(V_1, V_2, ..., V_r)$$

or its negation and selects that literal assessed by the heuristic as most valuable. There are three reasons for the "almost all". In the first case, we require at least one of the V_i's to be a variable introduced by the left-hand side of the clause or by a previous literal. The rationale for this restriction is that a new literal should have some linkage to previous literals in the clause. Secondly, the heuristic used to assess literals allows a form of pruning similar to the well-known alpha-beta [25]; some combinations of variables can be ruled out *a priori* because they can never exceed the value of the best combination so far. The third restriction, intended to avoid problems with infinite recursion, concerns recursive definitions where Q is the same as the left-hand side literal P. Here we insist that the literal must not allow the clause to invoke itself with arguments that could lead to a repetition of earlier arguments. The mechanisms used by FOIL to detect such anomalies, based on partial orderings of the constants, lie beyond the scope of this paper but may be found in [22].

Having selected the literal A_i, FOIL forms the new bindings set N and continues looking for further literals. If N contains only $\oplus$ tuples, the current clause is complete; those tuples explained by this clause are removed from the global bindings set and, if any

$\oplus$ tuples remain, the search for a new clause is commenced with the reduced global bindings set.

3.2 Other aspects and limitations of FOIL

Four other features of the system that complicate this simple description are:

- *Types and keys*: In databases, each argument of a relation has a specified type and the ways in which tuples in the relation can be accessed are spelled out as a set of keys, each key being a subset of the arguments that must be known in order to determine values for the remaining arguments. When constructing a clause, FOIL also uses information on types and keys to limit search for sensible literals.

- *Imperfect definitions*: In real-world problems it is often impossible to formulate rules that precisely characterise the domain. In such circumstances, many studies have found that simple, approximate rules can be more valuable than overspecified rules that "fit the noise" [2, 4, 19]. In this learning context, FOIL embodies *stopping criteria* for deciding

  - that a clause should not be extended further, even though the bindings set is not yet free of $\ominus$ tuples, and
  - that a set of clauses should not be expanded, even though they do not cover all instances of a relation

  These decisions are based on the premiss that the number of bits required to encode a sensible clause should never exceed the number of bits needed to indicate explicitly the $\oplus$ tuples covered by the clause. The rationale is similar to that described in [21].

- *Backtracking*: Sometimes, several possible literals appear to have approximately equal promise. FOIL contains a simple backup mechanism so that, when this occurs, a *checkpoint* is established. If the system paints itself into a corner while developing a clause, it can resume at the most promising checkpoint and try the next best literal. Note that the most promising checkpoint might not be the most recent, so this backtracking is non-chronological.

- *Post-processing of clauses*: Any greedy algorithm is prone to making locally optimal but globally undesirable choices. In this context, a literal chosen for the right-hand side of a clause may subsequently turn out to be unnecessary or even counterproductive. Once a clause has been developed, the literals of the right-hand side are examined to see whether any could be omitted without compromising the accuracy of the clause as a whole. Similarly, when a set of clauses defining the relation has been found, the system looks for redundant clauses that can be discarded without loss.

The present implementation, FOIL.2, is an unpolished 3300-line C program Before it is recoded as a more robust system for real-world problems, several limitations of the design presented here need to be addressed:

- *Literals*: the right-hand side of a clause consists of a string of literals of the form $Q(...)$ or $\neg Q(...)$. Further alternatives should be added:
  - X_i = k, binding some free variable to a constant.
  - $X_i > t$, where X_i is a variable bound to a numeric value and t is a threshold. The same techniques used by attribute-value systems to find appropriate thresholds (e.g. [2]) would seem to be applicable here.

  Of course, the negation of these literals would be considered at the same time.

- *Recursion*: The current algorithm attempts to avoid infinitely recursive definitions of a relation P by prohibiting occurrences of P on the right-hand side of a clause which could cause infinite recursion *on that clause*. A much more careful specification of how to avoid infinite recursion needs to be formulated. This may not turn out to be purely syntactic, but may involve empirical testing of partial or complete clauses.

- *More sophisticated search*: FOIL has no mechanism for planning the overall layout of a clause; it just adds the most attractive-seeming literal at each step. A simplified or abstracted plan of the target clause, such as GOLEM [16] generates, would help to keep search on track and would allow the system to find more complex clauses in a reasonable time.

4. RESULTS

FOIL has been tested on several learning tasks studied by other research workers. To illustrate its performance, three of these evaluations are summarised here.

4.1 Learning recursive relations on lists

Numerous authors have studied the task of learning list manipulation functions, among them Shapiro [24], Sammut & Banerji [23], Muggleton & Buntine [14] and Muggleton & Feng [16]. In many of these studies, carefully selected examples were presented to the particular learning system and it is unclear to what extent the choice of examples affected the systems' reported performance.

Two non-trivial relations learned by FOIL were the definitions of

append(X,Y,Z)	appending X to Y gives Z
reverse(X,Y)	X is the reverse of Y

given the other relations

list(X)	X is a list
null(X)	X is nil
components(X,H,T)	X has head H and tail T

Any effect of choice of training examples was neutralised by defining the relations over all lists of length up to four, each containing non-repeated atoms drawn from the set {1,2,3,4}. This gives a total of $4! + 4 \times 3! + 6 \times 2! + 4 + 1 = 65$ lists. For *reverse*, the

system used the closed-world assumption in which the $\ominus$ tuples consisted of all tuples not in the relation. This was not possible for *append* – there are 69^3 (or about 330,000) 3-tuples over 69 constants – so the $\ominus$ tuples were defined by selecting 10,000 of them at random.

The definition constructed for *append* in 51 seconds consists of the clauses

append(A,B,C) ← *B=C, null(A)*
append(A,B,C) ← *components(C,D,E), components(A,D,F), append(F,B,E)*

This example demonstrates the utility of post-processing clauses. The initial definition contained four clauses and the final clause started with a literal *components(E,G,B)*, effectively limiting the clause to cases where the first argument is a two-element list. When the definition had been completed, the post-processing phase noted that two clauses could be omitted and this literal could be removed from the final clause, resulting in the perfectly general definition above.

From the data for *reverse*, FOIL required 124 seconds to find the definition

reverse(A,B) ← *A=B, null(A)*
reverse(A,B) ← *append(C,D,A), append(D,E,B), components(B,F,E),*
reverse(C,E)

The times required to find the definitions of these relations are much greater than those reported by Shapiro [24]. namely 11 and 6 seconds respectively on a DEC 2060. For these tasks, however, there was no attempt to construct a minimal set of tuples from which suitable definitions can be found. FOIL used 10,261 tuples for *append* and 4,761 for *reverse* compared to 34 and 13 respectively in the case of Shapiro's MIS system. On the other hand, FOIL is probably incapable of finding adequate definitions from such small numbers of tuples. Whereas MIS's search for clauses is guided and constrained by external information (e.g., it can ask questions), FOIL navigates by extracting information-based hints from the data. As clauses become more complex, sensible choice of literals depends on being able to distinguish helpful clues from simple random variations and this is hampered if the number of tuples in the training set is small.

A recent system, GOLEM [16], provides another interesting point of comparison. GOLEM does not assemble clauses incrementally, as FOIL does, but rather extracts clauses from the least general generalisation of two positive training examples. On the same training sets used by FOIL, GOLEM running on an Apollo DN10000 found definitions for *append* and *reverse* in 48 and 14 seconds respectively, the latter being noticeably faster than FOIL. Part of this speedup can be ascribed to limitations on GOLEM's hypothesis space (e.g., it does not permit negated literals), but an important component is due to planning mechanisms that reduce or eliminate search, for which FOIL has no counterpart.

4.2 Chess endgame

Muggleton, Bain, Hayes-Michie, and Michie [16] describe a learning task that nicely demonstrates the utility of being able to generate imperfect but accurate definitions. The domain is the chess endgame White King and Rook versus Black King. The target

Table 2. Results on the chess endgame task

System	100 training objects		1000 training objects	
	Accuracy	Time	Accuracy	Time
CIGOL	77.2%	21.5 hr	N/A	N/A
DUCE	33.7%	2 hr	37.7%	10 hr
FOIL	92.5% sd 3.6%	1.5 sec	99.6% sd 0.1%	11.7 sec

relation is the six-place predicate *illegal(A,B,C,D,E,F)* indicating that the position in which the White King is at (A, B), the Rook at (C, D), and the Black King at (E, F) is not a legal White-to-move position. There are two other relations available:

adj(X,Y)	row or column X is adjacent to Y
less-than(X,Y)	row or column X is less than Y

A training set for this task consists of a random sample of positions, some of which (about 34%) are illegal and some not. Experiments were carried out with training sets containing 100 positions and 1000 positions and the learned definition was then tested on a large random sample of positions selected independently from the training sets. Note that there is no simple correct definition of *illegal* in terms of the given relations. The amount of information provided by training sets of this size is probably insufficient to allow any learning system to discover an exact set of clauses.

Table 2 shows results reported by Muggleton et al. for two clause-learning systems, CIGOL and DUCE. Each experiment was performed five times; the accuracies shown are the averages obtained over 5,000 unseen cases and the approximate times are for a Sun 3/60. The table also gives similar results for FOIL averaged over 20 repetitions; times are for a DECstation 3100 and accuracies were determined over 10,000 unseen positions. FOIL finds more accurate definitions than either CIGOL or DUCE, and requires much less time than can be accounted for by differences in hardware and implementation language.

As an example, the clauses found by FOIL on the first trial with 100 training objects are

illegal(A,B,C,D,E,F) ← *C=E*
illegal(A,B,C,D,E,F) ← *D=F*
illegal(A,B,C,D,E,F) ← *adj(B,F), adj(A,E)*
illegal(A,B,C,D,E,F) ← *A=E, B=F*
illegal(A,B,C,D,E,F) ← *A=C, B=D*

These may be paraphrased as: a position is illegal if the Black King is on the same row or column as the Rook, or the White King's row and column are both next to the Black King's, or the White King is on the same square as the Black King or the Rook. The clauses are certainly not exact – they cover only 30 of the 32 *illegal* tuples in this training set – but they correctly classify better than 95% of the 10,000 unseen cases.

main line	A♡	7♣	6♣	9♠	10♡	7♡	10◇	J♣	A◇	4♡	8◇	7♣	9♠
side lines			K◇			5♠		Q◇		3♠			9♡
			J♡										6♡
main line (ctd)	10♣	K♠	2♣	10♠	J♠								
side lines (ctd)		Q♡											
		A◇											

main line	J♣	4◇	Q♡	3♠	Q◇	9♡	Q♣	7♡	Q◇	9◇	Q♣	3♡	K♡
side lines	K♣	5♠				4♠		10◇					
		7♠											
main line (ctd)	4♣	K◇	6♣	J◇	8◇	J♡	7♣	J◇	7♡	J♡	6♡	K◇	

main line	4♡	5◇	8♣	J♠	2♣	5♠	A♣	5♠	10♡
side lines	7♣	6♠	K♣	A♡		6♣		A♠	
	J♡	7♡	3♡	K◇					
	4♣	2♣		Q♠					
	10♠	7♠							
	8♡	6◇							
	A◇	6♡							
	2◇	4♣							

Figure 3. Three Eleusis layouts (from [6])

4.3 Eleusis

The final task concerns the card game Eleusis in which players attempt to learn rules governing sequences. The dealer invents a secret rule specifying the conditions under which a card can be added to a sequence of cards. The players attempt to add to the current sequence; each card played is either placed to the right of the last card in the sequence if it is a legal successor, or placed under the last card if it is not. The horizontal *main line* thus represents the sequence as developed so far while the vertical *side lines* show incorrect plays.

Figure 3 contains three game layouts taken from Dietterich and Michalski [6], all of which arose in human play. The same paper describes SPARC/E, a system that tries to discover the dealer's rule from given layouts, i.e. the system learns from passive observation rather than experimentation. SPARC/E handled these problems well, producing a few hypotheses that included a reasonable rule in each case, and requiring only 0.5 to 6.5 seconds on a CYBER 175 to do so.

The same problems were presented to FOIL. After some experimentation, the following relations were set up to capture information relevant to these examples:

can-follow(A,B,C,D,E,F)	card A (suit B) can follow a sequence ending with card C (suit D), E consecutive cards of suit D, and F consecutive cards of the same color
next-suit(X,Y)	suit X precedes suit Y
next-card(X,Y)	rank X precedes rank Y
lower(X,Y)	suit/rank X is lower than suit/rank Y
face(X)	rank X is a face card
same-color(X,Y)	suits X and Y are the same color
odd(X)	rank/length X is odd

Each card other than the first in the sequence gives a tuple for the target *can-follow* relation, $\oplus$ if the card appears in the main line and $\ominus$ if it is in a side line.

For the first layout, FOIL found the clauses

can-follow(A,B,C,D,E,F) ← *same-color(B,D)*
can-follow(A,B,C,D,E,F) ← *odd(F), odd(A)*

in 0.1 seconds. The rule, which can be paraphrased "completed color sequences must be odd and must end with an odd card", does not account for one card in the main line. The rule intended by the dealer, "completed color sequences must be odd, and a male card cannot appear next to a female card", uses information on card sex that is not encoded in the above relations. Interestingly, SPARC/E also finds the inexact rule "completed sequences must be odd" which errs the other way in being too general.

The second layout gave

can-follow(A,B,C,D,E,F) ← *face(A), ¬face(C)*
can-follow(A,B,C,D,E,F) ← *face(C), ¬face(A)*

("play alternate face and non-face cards", the rule intended by the dealer) in 0.1 seconds.

For the third layout, FOIL found the clauses

can-follow(A,B,C,D,E,F) ← *next-suit(B,D), lower(C,A)*
can-follow(A,B,C,D,E,F) ← *next-suit(D,B), lower(A,C)*

("play a higher card in the suit preceding that of the last card, or play a lower card in the suit following that of the last card") in 0.1 seconds.

Both SPARC/E and FOIL are quite competent in this domain, but they differ in one important respect. Whereas SPARC/E generates several hypotheses, some of which may be too complex to be justified by the given layout, FOIL plumps for a single hypothesis which may be incomplete if the layout is small.

5. CONCLUSION

Starting from the perceived need to assist knowledge acquisition for expert systems, this paper has discussed one line of attack that generates knowledge through inductive generalisation of examples. Although this approach has proved its worth, it is lim-

ited by the simplicity of attribute-based descriptions of objects and learned concepts. This limitation is particularly relevant to application areas in design and engineering where structured objects are the norm. The attribute-based methods, however, can be extended in a fairly natural way to learn Prolog programs from function-free ground examples of several interconnected relations, such as those commonly represented in relational databases.

Many researchers have looked at the task of formulating Prolog programs from inspection of cases, but FOIL differs from other systems in several respects.

- FOIL finds a definition for each relation in turn, using the other relations as a kind of background knowledge. In this sense, FOIL is much more restricted in the way background knowledge can be expressed than most explanation-based learners [13].
- FOIL employs no idea of proof, being based instead on empirically-guided search over a space of bindings. This sharply differentiates FOIL from the explanation-based learners as well as systems such as MARVIN [23] and CIGOL [14].
- Some constructive induction systems like CIGOL can formulate new predicates. FOIL has no such facilities, being restricted to the definition of a concept in terms of existing relations.
- FOIL can learn multi-clause, recursive definitions of relations.

Finally, several extensions to FOIL are required before it could be considered a mature system capable of handling practical problems. However, it is hoped that many of these will involve only the borrowing of further well-proven techniques from the simpler attribute-value formalism.

Acknowledgements

I gratefully acknowledge support for this research from the Australian Research Council. Comments and suggestions of many colleagues, particularly Norman Foo, Dennis Kibler, Pat Langley and Jack Mostow, are much appreciated. Thanks to Mary Lou Maher and Yoram Reich of Carnegie Mellon University for providing the bridge data.

References

[1] Adeli, H. and Balasubramanyam, K.V., A novel approach to expert systems for design of large structures, *AI Magazine 9*,4:54-64 (Winter 1988)

[2] Breiman, L., Friedman, J.H., Olshen, R.A. and Stone, C.J., *Classification and Regression Trees*, Belmont: Wadsworth, 1984.

[3] Cestnik, B., Kononenko, I. and Bratko, I., ASSISTANT 86: a knowledge elicitation tool for sophisticated users, in Bratko and Lavrač (Eds.), *Progress in Machine Learning*, Wilmslow: Sigma Press, 1987.

[4] Clark, P and Niblett, T., Induction in noisy domains, in Bratko and Lavrač (Eds.), *Progress in Machine Learning*, Wilmslow: Sigma Press, 1987.

[5] Debenham, J.K., *Knowledge Systems Design*, Sydney: Prentice-Hall, 1989.

[6] Dietterich, T.G. and Michalski, R.S., Inductive learning of structural descriptions, *Artificial Intelligence 16*:257-294 (1981).

[7] Hunt, E.B., Marin, J. and Stone, P.J., *Experiments in Induction,* New York: Academic Press, 1966.

[8] Leech, W.J., A rule based process control method with feedback, *Proceedings Instrument Society of America Conference*, Houston, 1986.

[9] Marcus, Sandra (Ed.), *Automating Knowledge Acquisition for Expert Systems*, Boston: Kluwer, 1989.

[10] Michalski, R.S., Pattern recognition as rule-guided inductive inference, *IEEE Transactions Pattern Analysis and Machine Intelligence 2*:349-361(1980)

[11] Michalski, R.S., Mozetič, I., Hong, J. and Lavrač, N., The multipurpose incremental learning system AQ15 and its testing application to three medical domains, *Proceedings Fifth National Conference Artificial Intelligence*, Los Altos: Morgan Kaufmann, 1041-1045, 1986.

[12] Michie, D., Problems of computer-aided concept formation, in Quinlan (Ed.), *Applications of Expert Systems (Vol 2)*, Wokingham: Addison-Wesley, 1989.

[13] Mitchell, T.M., Keller, R.M. and Kedar-Cabelli, S.T., Explanation-based generalization: a unifying view, *Machine Learning 1*:47-80 (1986)

[14] Muggleton, S. and Buntine, W., Machine invention of first-order predicates by inverting resolution, in *Proceedings Fifth International Conference Machine Learning*, Los Altos: Morgan Kaufmann, 1988.

[15] Muggleton, S., Bain, M., Hayes-Michie, J. and Michie, D., An experimental comparison of human and machine learning formalisms, *Proceedings Sixth International Conference Machine Learning*, Los Altos: Morgan Kaufmann, 1989.

[16] Muggleton, S. and Feng, C., Efficient induction of logic programs, *Proceedings First Conference on Algorithmic Learning Theory*, Tokyo, 1990.

[17] Quinlan, J.R., Discovering rules by induction from large collections of examples, in Michie (Ed.), *Expert Systems in the Micro Electronic Age,* Edinburgh University Press, 1979.

[18] Quinlan, J.R., Induction of decision trees, *Machine Learning 1*:81-106 (1986)

[19] Quinlan, J.R., Simplifying decision trees, *International Journal Man-Machine Studies 27*:221-234 (1987)

[20] Quinlan, J.R., Induction, knowledge and expert systems, in Gero and Stanton (Eds.), *Artificial Intelligence Developments and Applications*, Amsterdam: North Holland, 1988.

[21] Quinlan, J.R. and Rivest, R.L., Inferring decision trees using the Minimum Description Length principle, *Information and Computation 80*:227-248 (1989)

[22] Quinlan, J.R., Learning logical definitions from relations, *Machine Learning 5*:239-266 (1990)

[23] Sammut, C.A. and Banerji, R.B., Learning concepts by asking questions, in Michalski, Carbonell and Mitchell (Eds.), *Machine Learning: an Artificial Intelligence Approach (Vol 2)*, Los Altos: Morgan Kaufmann, 1986.

[24] Shapiro, E.Y., *Algorithmic Program Debugging*, Cambridge: MIT Press, 1983.

[25] Winston, P.H., *Artificial Intelligence* (2nd Ed.), Reading: Academic Press, 1984.

Research Activities of Knowledge Acquisition and Learning in Japan

Shigenobu Kobayashi
Tokyo Institute of Technology*

Takao Terano
University of Tsukuba†

Hiroshi Motoda
Hitachi Ltd.‡

Riichiro Mizoguchi
Osaka University§

1 Introduction

The number of projects for research and development of knowledge-based systems in Japan has exceeded 3,000, with applications extending to all areas of industry. Despite this, even liberal estimates put the number of systems which have entered practical use under 500 and many systems still remain in the prototype or field test stages. This is caused by several factors. Among them, the knowledge acquisition bottleneck has been closed up more and more. As reasons of knowledge acquisition bottleneck, the following have been pointed out:

1. human experts rarely recognize or organize their expertise as an explicit model for problem solving,
2. they have not necessarily perfect knowledge which covers their domains, and
3. they use to make effort to acquire new knowledge from cases of success or failure continually, especially in design domains.

Therefore, it is quite important and urgent to establish a methodolgy for supporting or automating knowledge acquisition. In Japan, sice a few years ago, research activities of knowledge acquisition and machine learning have become active remarkably.

The purpose of this paper is to review the present of research activities related to knowledge acquisition in Japan. This paper consists of eight chapters. Chapter 1 is an introduction. Chapter 2 introduces a case study of practical knowledge acquisition activities, which was carried out at JIPDEC(Japan Information Processing Development Center). Chapter 3 is concerned with knowledge acquisition systems which support knowledge acquisition from domain experts. And several research activities about

*Graduate School of Science and Engineering 4259, Nagatsuta, Midori-ku, Yokohama 227, Japan kobayasi@sys.titech.ac.jp
†Graduate School of Systems Management 3-29-1, Otsuka, Bunkyo-ku, Tokyo 112, Japan terano@gssm.otsuka.tsukuba.ac.jp
‡Advanced Research Laboraotry Hatoyama, Saitama 350-03, Japan motoda@harl.hitachi.co.jp
§I.S.I.R., 8-1, Mihogaoka, Ibaraki, Osaka 567, Japan miz%ei.sanken.osaka-u.ac.jp@relay.cs.net

knowledge compilation from deep knowledge into operational knowledge are described in chapter 4. Chapter 5 deals with topics of knowledge acquisition from text. Chapter 6 reviews several activities about machine learning, focusing on similarity-based learning and explanation-based learning. Finally, topics of case-based reasoning are reviewed in chapter 7.

2 A Case Study of Knowledge Acquisition Activities

Knowledge acquisition (KA) tools and techniques have reached the level applicable to expert system (ES) development without knowledge engineers if they are used to solve small size analysis problems. However, present automated tools and techniques can hardly be applied to development of real-scale ESs for practical use because the tasks that they process are too simple and they do not have domain knowledge nor models.

The objectives of the case study which was carried out by a technical committee of JIPDEC, were[1]

1. to identify the causes of the knowledge acquisition bottleneck by analyzing real development processes of expert systems, and
2. to explore the future architectures for KA tools.

Over 30 ESs at the field use stage, developed in Japan, were surveyed, among which, the following five cases were intensively investigated:

1. ACAS-PRO: ES for assessing credit card application developed by Mitsubishi Research Institute,
2. ATREX: Automobile TRouble-shooting EXpert system developed by Toyota Central Research Lab.,
3. ALIS: a furnace operation monitoring system for a steel plant developed by Nippon Steel Corp.,
4. MH system: a scheduling system for operation of multi-chain hydro-power stations developed by Central Research Institute of Electric Power Industry, and
5. CAD system: a system for board design of electronics equipments developed by NTT Corp.

To make the investigation clear, we divide the knowledge acquisition processes into four phases in the case study.

1. knowledge extraction: Domain models and empirical knowledge of human experts for problem solving are elicited so as to be understood by knowledge engineers.
2. knowledge translation: The elicited knowledge is translated into machine usable forms using several knowledge representation techniques such as rules, frames, and so on.

3. knowledge structuring: The fragments of translated knowledge are organized into knowledge bases with inference mechanisms to be used in the problem solving processes.
4. knowledge refinement: The knowledge bases are checked, enhanced and maintained to make the systems more powerful.

The case study indicates that the key issues of KA of large ESs include the difficulty of 1) identifying tasks and problem solving functions, 2) eliciting domain knowledge and domain models, 3) managing complexity of target systems, and 4) supporting the life cycle activities. These are summarized in the following:

1. Task Analysis and Problem Solving Functions: It takes long time to make an initial expert model by identifying the tasks and problem solving functions at the beginning of the development. The process is essentially a diversing cooperative work carried out by both knowledge engineers and top level human experts. In each case above, however, no formal methods are used to support the process. we must make further researches on generic tasks and problem solving components.

2. Domain Knowledge and Domain Models: Domain knowledge and models are essential for ESs. In every case above, such knowledge and models are adequately established in the earlier stages of the development. However, the general knowledge or models of the domains can be acquired independently of the system development. For example, the automobile mechanics information of ATREX, the sensory information of ALIS, the conventional numerical programs of the MH system and the CAD system, etc. constitute basic domain knowledge and models of ESs. Empirical knowledge essential for the problem solving must be, of course, acquired from top level experts, however, the basic knowledge or these models can be obtained from the information of existing systems and middle- or lower-class experts. This suggests that development of generic prototypical library of domain knowledge and models will be of use for KA activities.

3. Complexity of Target Systems: The difference of size and complexity of target systems requires various KA activities and problem solving functions. No uniform formal methods can be used. The machine learning algorithm of ACAS-PRO had to be revised because of the size of test data. Large rule bases of ATREX and ALIS required special purpose knowledge editors. To make automated KA tools more feasible and usable, special care must be paid on the size and complexity of problems.

4. Life Cycle Issues: The role of KA activities usually changes with the progress of the ES development. In the knowledge extraction phase, the cooperative works of human experts and knowledge engineers are important to develop initial expert models of ESs. In the knowledge translation phase, structured interviews and machine learning methods are useful to represent knowledge. In the knowledge structuring phase, intelligent knowledge editors and testing environments are important for building the knowledge bases. In the knowledge refinement phase, we need verification-and-validation tools for automated maintenance management

aids to the ES enhancement and configuration management aids to the change control[2]. These software engineering issues must be resolved in the contexts of knowledge acquisition activities.

The above discussions led to a proposal of a general architecture for a future KA tool. The tool contains 1) interview components (including knowledge editors) for various users throughout the life cycle, 2) cooperative work support component for the earlier stage of the development, 3) machine learning component for middle and later stages of the development, 4) knowledge base maintenance component, 5) configuration management components for the later stages, 6) basic descriptions of generic tasks and corresponding problem solving functions, and 7) prototypical domain models and domain knowledge.

To develop such a tool, it is necessary that KA tools and KA methodologies are integrated, and that various libraries of generic tasks and problem solving functions and large scale common knowledge bases are developed.

In what follows, we have summarized some of the reseach activities, being carried in Japan, which are related to knowledge acquisition and machine learning.

3 Knowledge acquisition support systems

The main role of knowledge acquisition support systems is to support knowledge extraction or knowledge transformation through interactions between domain experts and systems without knowledge engineers. Here, knowledge acquisition support systems are classified into interview-based systems and task-oriented systems. The former lays stress on interview strategies and the latter does stress on use of generic tasks.

3.1 Knowledge acquisition by interview

Interviewing is an intellectual activity and very important for eliciting new knowledge from domain experts.

(1) SIS

An interviewer has two kinds of knowledge. One is about a domain under consideration and the other is knowledge about the interview process itself, which makes the interviewer an interview expert. The latter seems to be independent of the domain and consists of several interview knowledge primitives. A shell for an interview system, called SIS , has been developed on the basis of this idea [3]. It has seven domain-independent question strategy primitives. Task-specific interview systems can be generated automatically by describing the interview knowledge in term of these primitives.

To start an interview, SIS first requests initial information and then constructs an incomplete domain model which takes the form of a network. The sentences given by an expert are parsed and attentions are generated based on the primitives. These attentions are analyzed and converted to questions. The domain model is refined through answer-attention-question cycles.

The generality and effectiveness of SIS have been demonstrated through implementation of the two interview systems, MORE[4], a well-known knowledge acquisition system for diagnostic expert systems and $I^2S - LD$ [5], an interview system for logic design of data bases.

The SIS-made interview system only supports knowledge acquisition based on static analysis. More efficient knowledge acquisition can be done by solving problems using the acquired knowledge and refine it during the course of problem solving. $I^2S - LD$ has been extended to include this dynamic analysis capability [6], based on which a general architecture of knowledge acquisition by interview has been proposed [7].

(2) ISAK

Knowledge acquisition can be coupled with machine learning techniques. Along this line, a new system called ISAK, an interview system for acquiring design knowledge of oil-hydraulic circuits [8][9], is now under development. ISAK is composed of two major modules: UBL and UBI (Understanding-Based Learning and Interview). UBL is a module that compiles design knowledge through understanding some examples of design drawings. UBI is a module activated when UBL fails to understand an example and it acquires design know-how or missing domain theory which contributes to improving the performance of UBL.

ISAK has some interview strategies as well as initial domain knowledge. Given a design example, ISAK tries to explain why the example satisfies the design specification associated with it according to the domain theory and the topological structure of the circuit and identifies which component is most necessary for the specification to be satisfied. Identification is done on the basis of a generate-and-test paradigm, and an explanation is generated from which a piece of compiled design knowledge that maps function to attribute is extracted.

If the proof fails, the explanation process is analyzed and an interview is initiated according to the strategies. If the failure is due to a deficiency of the domain knowledge, the knowledge is refined or added by the expert, and if it is due to something which cannot be deduced theoretically, new heuristic knowledge is also acquired from the expert. Thus this system acquires not only heuristics but also domain knowledge and compiled design knowledge. Having the domain theory in advance helps reduce the number of questions raised by the interview system. The method used here is a combination of interviews and an explanation-based learning method (EBL). EBL-obtained design knowledge can be regarded as a specialization of domain theory rather than a generalization of an example, which is a key idea of ISAK.

(3) EPSILON/One

EPSILON/One is a knowledge acquisition system based on expert model consisting of operations which represent small tasks in human expert's problem solving process[10]. Seven types of operations are identified by analyzing real knowledge bases built for diagnosis problems in production rules. An interview method referred to as pre-post method is also developed. It consists of two interview strategies. It stimulates domain experts to remind pre or post operations by asking what operations are performed

before and after a specific operation. It enables to elicit knowledge about operations efficiently according to operation types. EPSILON/One is implemented in ESP on PSI machine and is composed of interface supporting graphic multi-windows and mouse, knowledge elicitation module, knowledge refinement module and knowledge transfer module. The knowledge acquisition research at ICOT has been conducted extensively and several results are obtained [10] [11][12].

(4) CONSIST

Most of the existing knowledge acquisition systems are based on top-down elicitation, by which we mean an interview for obtaining task-specific knowledge conducted assuming that the task type is known in advance. On the other hand, there exists another type of elicitation, i.e., bottom-up interview. CONSIST which stands for CONstruction Support system of information STructure) is a system of this type [13]. It is a frame-based system aiming at eliciting information structure underlying the domain of interest using KJ method [14]. CONSIST provides us with a useful assistance when the problem solving knowledge is vague even for domain experts.

(5) GRAPE

When more than one domain expert are available, we have to acquire expertise from all of them. GRAPE, GRoup ware for Acquiring, Processing and Evaluating knowledge, is a system for such situations [15]. It exploits an WHYSIWIS interface and elicit expertise for selection problems using AHP (Analytic Hierarchy Process).

3.2 Task-oriented knowledge acquisition

Building an expert system is still an art. Problem solving based on human information processing tasks has brought great insight into expert system architecture. A generic task is an abstract description of a particular task, and a problem solving process is simulated by the proper combination of generic tasks.

(1) MULTIS

Inspired by pioneering works on the generic task method [16][17] and the half weak method [18], new research is being conducted which seeks to establish a generic method for building an expert system [19][20][21].

In MULTIS, each generic task has its own problem solving method, is implemented as a building block or combination of building blocks, and requires task specific knowledge. A problem solver compiler generates a problem solving engine by structuring these building blocks in accordance with the generic tasks identified for the problem to be solved. MULTIS, MULti-Task Interview System, is now being developed based on this idea.

MULTIS acquires the necessary knowledge for each generic task. Therefore, two kinds of knowledge must be acquired, one being problem solving methods(task identification) and the other being the knowledge used in the task. The latter is acquired by an interview method similar to the one described above. The former is based on an

interview of another type, i.e. task analysis interview aimed at identifying the nature of the task. Since each task is considered as being domain-independent, it is possible to identify the task by dividing the given problem into a set of known tasks.

Collection of vocabularies specific to the task is important in this identification. MULTIS has, as its library, generic vocabularies, and case data of representative expert systems expressed by the generic vocabularies. A rough identification is first made under the framework of the classification problem and then the detailed identification is made based on case-based reasoning (CBR).

After the generic tasks have been identified, it is then necessary to synthesize the building blocks for these tasks into a single problem-solving engine. This part has not been completed yet.

(2) Automated generation of task-oriented problem solver

Another interesting work is an automatic generation of task-oriented expert system [22]. This system takes an approach which is slightly different from the above. It is based on the assumption that each information processing task can be regarded as a search problem with two kinds of control strategies: task-specific (domain-independent) and domain-specific. By identifying and specifying these strategies as searches, it is possible to automate generation of a task-specific expert system embedded with domain-specific knowledge. It is not necessary to prepare a set of generic tasks as construct primitives in advance. A task-specific problem-solving program can be constructed by specifying six element functions of a general search program. Search is defined here, based on a concept of abstract data type, as data structures and functions which operate on the data. These data structures consist of initial nodes, target nodes, intermediate nodes, relation-to-children nodes, node constraints, and node state. A total of 21 primitive functions have been predefined, which represent the node state. They operate on the data to alter the state, and are used to specify the six element functions above.

The user can build an expert system by providing search strategies of both kinds and domain-specific knowledge by answering the instructions given by using the data structure specified. The system is implemented by ES/KERNEL. Three different kinds of existing expert systems on planning problems in nuclear power plants, all of which are also implemented by ES/ KERNEL, have been reproduced using this system.

The generated expert systems exhibit exactly the same performance, but the number of programming steps has been reduced to one third of the original for each case. It is hoped that this approach can be applied to a wide class of tasks, although this still remains to be confirmed.

4 Knowledge compilation

Knowledge compilation is a very effective method for acquiring heuristics from a model describing a working mechanism for a system where deep functional knowledge is available. The need for this approach has been recognized since the early phase of expert system technology [23] and significant work has already been carried out on compiling diagnostic knowledge from deep models [24].

Deep knowledge is objective knowledge which basically depends on domain, not on task. Task knowledge is embedded in the compiler. The underlying concept is that many useful heuristics can be justified by deep objective knowledge. It would be desirable if deep knowledge could be prepared in a task-independent manner and if various task-dependent knowledge compilers worked on this knowledge to produce task-specific shallow knowledge.

4.1 Diagnostic knowledge compilation

(1) KC-II

One major research in this field is the development of a knowledge compiler for a diagnostic expert system which uses qualitative reasoning [25]. It has been determined that there are five kinds of knowledge required for diagnosis: knowledge of 1) device world (DW), i.e. structure and function of the system, 2) physical world (PW), i.e. principles governing the domain, 3) control world (CW), i.e. failure frequency and ease of measurement used to control reasoning, 4) interpretation world (IW), i.e. interpretation of internal state to make correspondence with failure, and 5) failure mechanism world (FMW), i.e. heuristics about how failure takes place. DW and PW are domain-dependent but task-independent, and can be called deep knowledge. CW and IW are both domain and task-dependent. These are forms of knowledge that the knowledge compiler makes reference to during compilation rather than being deep knowledge. FMW is heuristics about failure, but some of them are described in a domain-independent manner, e.g., existence of an obstacle is caused either by production or by movement.

Using these five kinds of knowledge, retrospective qualitative reasoning is first made in a backward direction from a symptom towards possible hypotheses. Then prospective qualitative reasoning is performed to find possible symptoms, and diagnostic rules are generated. The method seems very simple but it is very useful and it has been demonstrated that when applied to diagnosis of air-conditioners, it can generate rules which even domain experts could not think of easily.

The idea of knowledge compilation is also applied to a scheduling expert system of electric power network maintenance [26]. They identified several kinds of deep knowledge by investigating how every shallow knowledge in the expertise could be justified. A unified framework of knowledge compilation is designed based on these two results [27].

(2) KC-II/DST

The latest results related to the above system include a domain- and task-specific shell referred to as KCII/DST [28]. It is designed based on the above mentioned knowledge compiler. Its major characteristics are:

1) Diagnostic rules are compiled from deep knowledge, 2) The rules generated are hierarchically organized and infer intermediate hypotheses according to the hierarchical structure of the device of interest, 3) It can deal with devices with feedback loops and failure mode dependent diagnosis, and 4) Sophisticated iconic interface is available for describing DW.

KCII/DST is implemented in ESP (Extended Self-contained Prolog) developed by ICOT on PSI-II(Personal Sequential Inference machine) also developed by ICOT, and in Common ESP (CESP) on SUN3/260. Although it is a prototype system, a novice user can obtain a diagnostic expert system for the device by describing its structure through the iconic interface.

(3) Compilation using EBL

Compilation of diagnostic knowledge from deep models can be performed based on the explanation-based learning method[29]. This study uses adjustment cases of domain experts as training examples, the causal model as domain theory and observability and strategic importance as operationality criteria respectively. This method consists of generation of explanation structure, reduction of explanation structure, and transformation to rules. First, explanation structure is generated by explaining adjustment case using the causal model under machine adjustment goal. Second, this explanation structure is generalized by structural reduction using the operationality criteria. Finally it is transformed into operational rules. A knowledge compilation system, which has been developed based on this framework, acquired 79 diagnostic rules when appied to 50 actual adjustment cases in cigarette making machines. Comparing these rules with the knowledge base in the practical expert system, the effectiveness of the proposed method has been proven in both knowledge acquisition and knowledge refinement.

(4) QR/P

Qualitative simulation can be applied to knowledge compilation. QR/P is a system for diagnostic knowledge compilation using qualitative simulation, in which diagnosis knowledge rules are generated from a structural model built in terms of qualitative parameters and constraints among them[30]. The difference between the normal behavior and abnormal one activates the rule generation procedure. It has been implemented in ESP on a PSI machine and was applied to glaucoma diagnosis.

(5) BERG

Problem solving and knowledge acquisition is intricately connected. A hypothetical reasoning method called BERG[31] is developed to achieve a function of compiling compact rules for diagnosis. This method first performs a diagnosis of a specific problem based on the RESIDUE algorithm developed in the DART project, and the result is generalized (i.e. compiled) using EBL. Application to electric circuits indicates that the generated rules run about 10 times faster.

4.2 Design knowledge compilation

(1) MECHANICOT

MECHANICOT[32][33] is a tool for design expert system's based on knowledge compilation. Design process is formulated as a constraint satisfaction problem in MECHANICOT, where knowledge about the domain such as constraints, facts and theories is

described in a declarative form. Knowledge compiler in MECHANICOT generates existing paths of constraint processing more efficient by translating the input design specifications into design plans assuming that the structure of the design objects has been determined. Its main mechanism is based on the idea that constraints described declaratively can be transformed into procedural representations using the model of design process. MECHANICOT has been implemented in ESP on PSI-II developed by ICOT and been applied to parametric design of gear units.

(2) Compilation using EBL

An EBL-based method for acquiring operational design knowledge in physical systems was proposed[34]. In this study, any object can be interpreted via various design rationalities such as technological, causal and economical ones and it can be modeled as a hierarchy which consists of design goals, subgoals, structures and substructures toward attaining those goals. Through organizing domain-specific knowledge of the EBL system according to the above model, this system yields a systematic explanation of how structures attain design goals. This results in a generalized version of Functional Diagram used in Value Engineering. From this diagram, general design knowledge of various levels can be generated by a single instance of designed objects.

(3) Hierarchical compilation

Another study now in progress is the use of hierarchical qualitative reasoning based on approximations [35][36][37]. This method enables acquisition of heuristics and simultaneous utilization of knowledge expressed on multiple levels with different abstractions. It makes use of equations which express physical principles as deep knowledge, and compiles them into shallow knowledge expressing representative behavior of a particular element or composite element. This shallow knowledge expresses the way a human expert understands a complex composite element. The method serves as a framework for formalizing various heuristics which have so far been difficult to handle without an experiential method. Special care was taken to suppress over-generalization caused by approximation. The method is now being tested for its understanding of electric circuits.

5 Knowledge Acquisition from texts

Knowledge acquisition has to be done from all the knowledge sources available. We have discussed thus far knowledge acquisition through interview and compilation. This chapter deals with some results on knowledge acquisition by text understanding.

(1) hmU

A system called hmU performs automatic model construction from hardware manuals for CAD systems[38]. It analyzes given natural language specification of an LSI chip and builds a knowledge structure specifying the behavior of the chip. The model acquired will be used by an intelligent symbolic simulator to provide designers with

explanation about the chip. hmU is composed of natural language analyzer and model builder. The former parses the documents and resolves the ambiguities using the domain specific knowledge. The model builder is given internal representation of the analysis results and builds an automaton-based hardware model structure using the common sense knowledge about hardware, time, action, events, etc.

(2) Knowledge acquisition of relations between concepts

Although concept learning is a very important issue in machine learning, little research has been done about real world problems. This is partly because a teacher is not available, it is difficult to let the system know what to learn in advance, no sophisticated method is available for dealing with the complicated real world concepts. To cope with these difficulty, a system was developed, which acquires knowledge about relations between concepts from news paper articles[39]. The system refines its knowledge about a concept incrementally. It analyzes its knowledge base to find insufficient parts, determines the importance of the knowledge required and tries to acquire it from a human. Five kinds of knowledge acquisition schema such as generalization, specialization, exclusion, categorization and diplomacy are proposed in the system.

6 Machine learning

Learning is defined as a process to acquire knowledge necessary for problem solving and to refine its performance. The former process is called knowledge acquisition and the latter is called knowledge refinement. Machine learning can be classified into inductive leraning, deductive learning, and others. The inductive learning is often called similarity-based learning, because of its investigation of similarities among examples. The deductive learning is often called explanation-based learning, because of its usage of explanations by domain theory. In the following, research activities of similarity-based learning and explanation-based learning are reviewed.

6.1 Similarity-Based Learning(SBL)

(1) ARIS

ARIS (Adaptive Rule Induction System), an extended version of ID3, is developed [40]. Identification of useful attributes and generalization of the resulting trees play crucial rolls among several issues in decision tree induction. ARIS is designed based on a decision tree induction method but is augmented by introducing an attribute generation mechanism and generalization operators. New attributes are generated in the same way as the original induction method using only samples that are difficult to discriminate. ARIS has an another function which deals with attributes that take numerical values. It is implemented in Common Lisp, Flavors and OPS5 on Symbolics 3620. The latest version of ARIS is extended to coordinate with production rules written by human experts. An integrated environment enables to amalgamate human knowledge and one obtained by machine learning. The performance of ARIS was evaluated through the phoneme recognition in continuous speech. The result shows

the decision tree obtained by ARIS is superior to production rules written by human experts.

(2) LS/1

LS/1 is a domain independent inductive learning system, which repeats Question-Response-Answer interactions with a teacher, and learns the structure of the relation between questions and answers[41]. The knowledge representation system of LS/1 is called a label net which represents relational data base, conceptual hierarchy and certainty factors of rules. The learning algorithm of LS/1 includes generalization and merging of rules, generalization by identification of two predicates, and changing ceratinty factors in rules. The effectiveness of LS/1 has been shown by application to the task of learning English-Japanese translation.

(3) CIGOL

CIGOL[42] is one of the approaches that weaken the bias, and it can invent new terms by using inverse resolution operators while it is learning new concepts. Inductive learning methods are important for intelligent systems to acquire knowledge. Since most inductive learning systems are strongly biased by their representation language of concepts, auxiliary concepts must be provided before learning. For this reason, the bias is generally too strong. However, its teacher must answer difficult questions. In order to alleviate the teacher's responsibility, CIGOL adopts a learning method which improves the interaction between the teacher and the learner. That is, any regular set can be learned with only restricted inverse resolution operators and simple queries.

6.2 Explanation-Based Learning(EBL)

The mainstream of machine learning for the last few years seems to have been EBL and its extension. EBL can learn justified generalizations from only one example when the domain theory is perfect, but it does not work when the domain theory is imperfect. Imperfectness of the domain theory can be classified into four levels, i.e. incomplete, intractable, inconsistent and non-operational theories. Recently, most researchers have been concerned with an attempt to solve these imperfect theory problems.

(1) Augmented EBL

An augmented EBL can be formulated to handle plural examples simultaneously to unify EBL and SBL and a concept of the least EBG is introduced to extract similarities from multiple examples[43].

For the non-operational theory problem, it is shown that the least EBG is more operational than the usual EBGs. Theorems about operationality of the least EBG show the utility of the least EBG. For example, well known programs, (*e.g.*, simple list processing predicates such as *member* or *append*) are already operational because they have no backtrackings. In much of existing EBLs, its performance often becomes wrong because it generates the same number of macros as examples, but a learner using least EBGs can learn the most operational EBG.

(2) EBG version space method

For the inconsistent theory problem, a concept of an EBG version space is introduced, and the MSV (the Most Special Version) is shown to be the least EBG[44]. We assume that training examples are given with judgements whether they are positive or negative, and define a domain theory is inconsistent iff some negative examples are provable as positive examples. It is further assumed that a learner swaps rules which conclude the goal concept with EBGs. A learning task is to find EBGs s.t. prove positive examples but never prove negative examples. In SBL, such generalizations make a range called a version space. Its least upper bound and its greatest lower bound are called MGV (the Most General Version) and MSV, respectively. It is a powerful method for incremental learning. It is shown that a generalization space of EBG has the same structure as a version space.

The generalization hierarchy of common explanation structures, which are generated from multiple primitive explanation structures, can be effectively organized based on a concept of maximal covering.[45] The maximal covering reflects similarities among the common explanation structures. By top-down search over the generalization hierarchy, the best set of macro rules, which includes all positive examples and excludes all negatives, can be found. A learning system based on this method has been implemented in Prolog. Some experiments show that it can easily generate valid macro rules under the inconsistent domain theory. Its incremental version has been developed as an extension[46].

(3) EBL with background theory

The intractable theory problem is distinctive in synthetic problems such as planning and design. A learning task for intractable theory problem is to find strategies to solve any problems without search. The least EBG can also be used. However, it often falls into a difficulty of SBL that a learner requires numerous negative examples before convergence. It means that the computational load shifts from EBL to SBL to resolve the imperfectness of the domain theory. Therefore, to solve the intractable theory problem, it is suggested to introduce a meta level domain theory. In practice, it is shown that the use of meta-level domain theory may reduce computational loads[47].

For the incomplete theory problem in classification, an approach using a background theory in addition to the usual domain theory is proposed[48]. A background theory consists of a commonsense knowledge such as text books or dictionaries. A learning task is to extend a domain theory by transferring knowledge from the background theory based on similarities of examples. The least EBG can also be used. Thus, to solve the incomplete theory problem, it is necessary to consider knowledge transfer from other domains. Analogical reasoning or case-based reasoning seems useful methods realizing knowledge transfer.

(4) PiL

Paradigm-based inference learner, PiL for short, is a learning system for acquiring strategic knowledge of problem solving. It first tries to solve a given problem[49][50][51]. When it fails, it asks a solution from which it acquires some strategic knowledge for

basic operator applications. The strategic knowledge is represented as macro-operators which are subsequences of operator applications. PiL was applied to generation of strategic knowledge for linear equations. The evaluation function used for selection of macro-operators is called the perfect causality, which is a heuristics for selecting valid macro-operators. Macro-operators are generalized according to EBG. PiL2 is a direct successor of PiL and has the augmented perfect causality. It is applied to robot planning and obtained a successful results.

(5) Learning from partial explanation structures

An integrated learning method with an extended generator of macro operators is proposed[52]. In this study, it is shown that a set of rules can be specialized and generalized by rule composition and decomposition. Explanation structures of examples are generated using domain theory and they are generalized by decomposition. The usefulness of this method was demonstrated by application to synthesis of logical circuits and English-Japanese translation.

(6) Frustration-Based Learning (FBL)

A new learning method worth noting is a method acquiring strategic knowledge based on frustration [53][54]. The method, called FBL, can extract underlying information while solving a problem and use it as strategic knowledge for future problems. To do this, FBL must find where, what and how much to learn. FBL uses frustrated states during problem-solving processes as motive forces for learning. It learns a kind of associative knowledge which pairs a frustration element with the information relevant to it.

FBL has two important processes: identification of frustrated states; and limited reasoning. Frustrated states are those states where no further local inference is possible due to the lack of some associative knowledge. Identification of these states contributes to discovering where to learn. Limited reasoning is a process for extracting underlying information which is essential for resolving the frustration. A restricted world is constructed to each frustrated state by extracting the individual pieces of information that allow the local inference to continue. This process can derive inferences not in the original problem but potentially useful for later problems. This reasoning contributes to determining what and how much to learn.

FBL acquires an associative and strategic knowledge. This process is another type of knowledge compilation, but the compiled knowledge is not necessarily guaranteed to work in all situations, and thus is heuristic.

FBL has been applied to two problem domains of different natures: the auxiliary-line problem in elementary geometry; and the metaphor-understanding problem. Both are problems in which correct associations are required; simply having domain theory is not enough. When applied to the former problem, it successfully learned strategies for drawing appropriate auxiliary-lines and when applied to the latter problem, it successfully discerned general metaphors behind processed sentences. In both cases the performance of problem solving capabilities increased as it learned various pieces of strategic knowledge.

These results suggest the generality of the FBL framework. Learning by FBL is equivalent to understanding the whole problem as a composite of the acquired underlying information.

7 Case-Based Reasoning

The knowledge-based reasoning(KBR) requires complete and consistent knowledge base. This brings about the difficult problem of knowledge acquisition and refinement. For supplementing KBR, it is useful to introduce analogical reasoning or case-based reasoning that explicitly uses past cases which are experts' experiences with success or failure results.

(1) Activity of ICOT KSA/KAR WG

The KSA/KAR working group, established in ICOT last year, has surveyed the needs and seeds of CBR technologies from a variety points of views[55]. The main results of this group's activities are summarized as follows:

1. practical needs on planning and design problem domains were extracted and the availability and usefulness of the CBR approach was clarified,
2. effective ideas to overcome issues related to indexing of cases, organization of case base, retrieval of analogical cases, modification and repair of cases, and so on were presented,
3. a theoretical connection between CBR and EBL was investigated,
4. the CBR was located at the center of a framework of integrating problem solving paradigms and learning paradigms.

(2) Analogical Reasoning

A new formalism of analogical reasoning is presented[56]. According to the formalism, a single domain for the reasoning is given to a reasoning system. The domain has some rules, constraints, and negative facts. The negative facts are used to reject in appropriate analogies that derive contradictions. Based on the formalism, a possibility of inferring incomplete information by analogy is discussed. Moreover a procudure to find a sound analogy is presented.

(3) Interactive CBR model

To automate the CBR process, domain knowledge for problem indexing and case modification or repair must be prepared in advance. However, it is difficult in practice. To overcome such a difficulty, an interactive CBR model is proposed[57]. This model makes interactions between the user and the system flexible under imcomplete domain knowledge. This model has been applied to a machine adjustment problem. An adjustment case is defined as an instance that goal was set under condition such that some faults were inspected, possible cases were identified, method was performed, and result and its evaluation were recorded. A prototype CBR system based on this model has

been developed. Application of this CBR system to machine adjustment problems has shown that it can generate appropriate method plans in cases of failure by the existing expert system.

Roles of domain knowledge and user in interactive CBR systems should be investigated from the viewpoints of costs and benefits[58]. The domain knowledge needed to realize CBR consists of conceptual knowledge hierarchy, causal knowledge, heuristic knowledge and meta-knowledge. Heuristic knowledge consists of case modification knowledge and measurement and testing knowledge. Conceptual knowledge hierarchy is often used in building/managing the case base, problem indexing and case modification/repair. Causal knowledge is often used in case modification/repair. As case modification/repair is a central task in CBR, these three kinds of knowledge are expected to contribute to automating the whole CBR process. Control knowledge for reasoning is important especially in automating case modification process. Costs of knowleadge acquisition and benefits of knowledge utilization at each step of CBR can be estimated qualitatively. The results of such a cost-benefit analysis suggest a form of cooperation between domain knowledge and user.

(4) Case-based process diagnosis

A framework for case-based diagnosis is prensented[59]. In this framework, a mechanism to drive the dynamic organization of the memory is introduced. In the memory, knowledge is represented by a frame which is structured to maintain the number of slots small by sharing the common attribute-values among cases. An architecture for case-based diagnosis is presented, using the memory organization where diagnostic cases are indexed both by the symptoms and by the causal relations among symptoms. The program CAOS(CAse Operation System) is implemented for the prosess diagnosis domain.

(5) Case-based information retrieval

Information retrieval requires a lot of time for users to generate appropriate retrieval conditions. CBR is effective in reducing such an unwilling loss of effort. By introducing the framework of Carbonell's derivational analogy and organizing retrieval cases systematically, a case-based reasoning system can be constructed[60]. In this study, a case is defined as a tuple of intial and final retrieval conditions, transformation rules and trial sequence. The case base is organized as a lattice structure on the basis of similarities among cases. For retrieval of analogical cases, conceptual hierarchical knowledge are necessary. A prototype CBR system has been implemented on Prolog. As the framework of this system is domain-independent, it is applicable not only to information retrieval but also to other problems with analogical structure.

8 Concluding remarks

Before finishing this review, one of new projects in progress in Japan will be mentioned. ASTEM (Advanced Software Technology & Mechatronics Research Institute of Kyoto) is going to set up a project on a large-scale engineering knowledge base under

collaboration with academic and industrial organizations. The project aims at building knowledge bases playing as an infrastructure of expert systems which contains a huge amount of knowledge such as domain principles , general knowledge and common sense in engineering. The resulting knowledge base will be used as deep knowledge from which several kinds of shallow knowledge is generated, as domain theory in EBL and as stuff of basic knowledge in interview system, which helps knowledge acquisition to a greate extent. This project will make a useful contribution to the progress of knowledge-based systems research.

Research activities related to expert system building technology are becoming very active not only at universities, but also at industries in Japan. The ones reviewed in this paper cover only a small portion of such an activity, but highlight efforts being made to advance expert system development from an art to an engineering discipline. Research activities of knowledge acquisition and machine learning will make a true contribution to realization of intelligent systems. We have confidence that fruitful results will be expected in the near future.

References

[1] Kobayashi S., and Terano T.,(eds.) *Knowledge Systems Handbook(in Japanese)*, Ohm Co., Tokyo (in Press) .

[2] Terano T., and Kobayashi S., *Problem Analyses, Tool Evaluation, and Verification and Validation Study: Three Steps for Knowledge-Based Systems Development Methodology*, Proc. of IJCAI'89 Workshop on Verification, Validation and Testing of Knowledge-Based Systems, 1989.

[3] Kawaguchi A., Mizoguchi R., Yamaguchi T., and Kakusho O., *SIS: A Shell for Interview Systems*, Proc. of International Joint Conference on Artificial Intelligence, pp.359 -361, 1987

[4] Kahn G., Nowlan S., and McDermott J., *MORE: An Intelligent Knowledge Acquisition Tool*, Proc. of International Joint Conference on Artificial Intelligence, pp. 581-584, 1985

[5] Kawaguchi A., Taoka N., Mizoguchi R., Yamaguchi T., and Kakusho O., *An Intelligent Interview System for Conceptual Design of Database*, Proc. of the 7th European Conference on Artificial Intelligence, pp.39-47, 1986

[6] Kawaguchi A., Matsuyama E., Numata K., Mizoguchi R., Nomura Y., and Kakusho O., *A Database Construction/Manipulation Support System based on Interviewing (in Japanese)*, Journal of Japanese Society of Artificial Intelligence, Vol. 4, No. 7, pp.421-430, 1989

[7] Kawaguchi, A., Motoda, H., and Mizoguchi, R., *An Architecture of Knowledge Acquisition by Interview based on Dynamic Analysis*, Proc. of JKAW90, 1990

[8] Matsuda K., Nomura Y., Baba T., Nakashima Y., Kawaguchi A.,and Mizoguchi R., *An Interview System for Acquiring Design Knowledge based on Understanding of Examples (in Japanese)*, Proc. of Japanese Society of Artificial Intelligence, pp.739-742, 1989

[9] Matsuda, K., and Mizoguchi R., et al., *ISAK: Interview system for acquiring design knowledge*, Proc. of JKAW90, 1990.

[10] Taki H., and Tsubaki K., *Expert model: A knowledge representation for knowledge acquisition(in Japanese)*, Journal of Japanese Society for Artificial Intelligence, Vol.5, No.2, pp.203-212, 1990.

[11] Taki H., *Knowledge acquisition by observation*, Proc. of FGCS, pp.1250-1258, 1988.

[12] Taki H., et al., *Operation presumption: Knowledge acquisition by induction*, Proc. of EKAW89, pp.34-48, 1989.

[13] Shinohara Y. and Terano T., *Support system for construction information structure using frame-based hierarchical relations(in Japanese)*, SIG-AI-86-32, Information processing Society of Japan, pp.31-36, 1986.

[14] Kawakita J., *The method of abductive thinking (in Japanese)*, Chuoukoron Pub. Co. Ltd., 1967.

[15] Ueda H., and Kunifuji S., et al., *An implementation of knowledge acquisition support groupware GRAPE (in Japanese)*, Proc. of the 4th National Conference of Japanese Society for Artificial Intelligence, 17-8, pp.673-676, 1990.

[16] Chandrasekaran B., *Generic tasks in knowledge-based reasoning: high-level building blocks for expert system design*, IEEE Expert, Vol. 1, No. 3, pp.23-30, 1986

[17] Breuker J., and Wielinga B., *Models of Expertise in Knowledge Acquisition*, Topics in Expert System Design, Guida G., and Tasso C. (Editors), Elsevier Science Publishers, pp.265-295, 1989

[18] McDermott J., *Using problem-solving methods to impose structure on knowledge*, Proc. of Intl. Workshop on Artificial Intelligence for Industrial Applications, pp.7-11, 1988

[19] Tijerino Y., Kawaguchi A., Mizoguchi R., and Kakusho O., *A Generic Vocabulary and Generic Task Library for Use in MULTIS*, SIG-KBS-8805-3, Japanese Society of Artificial Intelligence, pp.21-30, 1988

[20] Mizoguchi R., *Towards Establishment of a Methodology for Building Knowledge-based Systems (in Japanese)*, SIG-KBS-8901-4, Japanese Society of Artificial Intelligence, pp.21-30., 1989

[21] Tijerino Y., and Mizoguchi R., et al. *A task analysis interview system that uses a problem solving model*, Proc. of JKAW90, 1990.

[22] Kasahara T., Yamada N., and Kobayashi Y., *Automated Generation Method of Task-oriented Problem Solver for Expert System Architecture (in Japanese)*, SIG-KBS-8905-6, Japanese Society of Artificial Intelligence, pp.41-48, 1990

[23] Chandrasekaran B., and Mittal S., *Deep Versus Compiled Knowledge Approaches to Diagnostic Problem-solving*, Proc. of the National Conference of Artificial Intelligence, pp.349-354, 1982

[24] Sembugamoorthy V., and Chandrasekaran B., *Functional Representation of Devices and Compilation of Diagnostic Problem- Solving Systems, Learning, Experience and Memory*, edited by Kolodner and Riesbeck, Laurence Erlbaum and Associates, pp.47-73., 1986

[25] Yamaguchi T., Mizoguchi R., Taoka N., Kodaka H., Nomura Y., and Kakusho O., *Basic Design of Knowledge Compiler Based on Deep Knowledge (in Japanese)*, Journal of Japanese Society for Artificial Intelligence, Vol. 2, No. 3., pp.333-340, 1987

[26] Adachi K., Yamaguchi T, and Mizoguchi R. et al., *Deep knowledge in scheduling type of expert systems (in Japanese)*, SIG-KEAI 59-18, Information Processing Society of Japan, pp.1988

[27] Yamaguchi, T., and Mizoguchi R. et al., *A unified framework for deep reasoning in expert systems (in Japanese)*, SIG-KBS-8801-3, Japanese Society for Artificial Intelligence, pp.23-31, 1988

[28] Ozawa T., and Mizoguchi R. et al., *A DST based on a knowledge compiler(KCII)*, SIG-KBS-8905-5, Japanese Society for Artificial Intelligence, 1990

[29] Nakamura K., and Kobayashi S., *Knowledge Acquisition from Machine Adjustment Cases by Causal Model and Operational Criteria*, Proc. of JKAW90, 1990

[30] Owada H., Mizoguchi F. and Kitazawa Y., *A method for developing diagnosis systems based on qualitative simulation (in Japanese)*, J. of Japanese Soc. for AI, Vol.3, No.5, pp.617-626, 1988

[31] Koseki Y., *Experience Learning in Model-Based Diagnostic Systems*, Proc. of International Joint Conference on Artificial Intelligence, pp.1356-1362, 1989

[32] Nagai Y., Terasaki S., Yokoyama T. and Taki H., *Expert system architecture for design tasks*, Proc. of FGCS'88, pp.298-317, 1988

[33] Nagai Y. and Terasaki S., *Towards constraint analysis and plan generation of constraint compiler (in Japanese)*, Proc.of the Third National Conference of Japanese Soc. for AI, 11-43, pp.693-696, 1989

[34] Katai O., Kawakami H., Sawaragi, T., Iwai S. and Konishi T. *Acquisition of operational design knowledge from designed objects using explanation-based learning method (in Japanese)*, Trans. of Soc. of Instrument and Control Engineer, Vol.26, No.8, pp.916-923, 1990

[35] Yoshida K., and Motoda H., *An Approach to Hierarchical Qualitative Reasoning - Constructing Shallow Knowledge from Deep Knowledge - (in Japanese)*, Journal of Japanese Society for Artificial Intelligence, Vol. 4, No. 4., pp.447-455, 1989

[36] Yoshida, K., and Motoda, H., *Towards Automatic Generation of Hierarchical Knowledge Bases*, Working Notes of the AGAA-90 Workshop, Boston, pp.98-109, 1990

[37] Yoshida, K., and Motoda, H., *Hierarchical Knowledge Representation based on Approximation*, Proc. of JKAW90, 1990

[38] Nishida T., Kosaka A. and Doshita S., *Towards knowledge acquisition from natural language documents –Automatic model construction from hardware manual–*, Proc. of the 6th IJCAI, Kalsruhe, pp.482-486, 1983

[39] Handa K. and Ishizaki S., *Acquiring knowledge about relation between concepts*, Proc. of EKAW89, Paris, pp.380-390, 1989

[40] Tsujino K., and Mizoguchi R., et al., *Adaptive rule induction system: ARIS (in Japanese)*, Trans. of IEICE of Japan, Vol.J72-D-II, No.1, pp.121 -131, 1989

[41] Akama K. *Learning of translation by the inductive learning system LS/1 (in Japanese)*, J. of Japanese Soc. of Artificial Intelligence, Vol.2, No.3, pp.341-349, 1987

[42] Sakurai S. and Haraguchi M. *Improving man-machine interaction in learning regular sets*, Proc. of PRICAI'90, 1990

[43] Yamamura M., and Kobayashi S., *An Augmentation of EBL on Plural Examples (in Japanese)*, Journal of Japanese Society for Artificial Intelligence, Vol. 4 , No. 4., pp.389-397, 1989

[44] Yamamura M., and Kobayashi S., *Towards Unifying EBL and SBL to Imperfect Domain Theory Problems*, Proc. of PRICAI'90, 1990

[45] Kobayashi S., Shirai Y., and Yamamura M., *Acquiring Valid Macro Rules under Imperfect Domain Theory by Topdown Search on Generalization Hierarchy of Common Explanation Structures (in Japanese)*, Journal of Japanese Society for Artificial Intelligence, Vol. 6, No. 3 (to appear)

[46] Ono, T, Yamamura M., and Kobayashi S., *Incremental Refinement of Domain Theory based on Structural and Parametric Generalizations (in Japanese)*, Proc. of 11-th Knowledge and Intelligent Systems Symposium, pp.81-86,1990

[47] Yamamura M., and Kobayashi S., *An Augmented EBL Learner for Acquiring Problem Solving Macrotable (in Japanese)*, Journal of Japanese Society for Artificial Intelligence, Vol. 6, No. 1 (to appear).

[48] Araki, Y., Yamamura M., and Kobayashi S., *An Incremental Learning of Domain Theory with a Background Theory (in Japanese)*, Proc. of 11-th Knowledge and Intelligent Systems Symposium, pp.87-92,1990

[49] Yamada S., Abe N. and Tsuji S. *PiL: A system to learn strategy knowledge in problem-solving (A case study in an equation & inequality of the first degree) (in Japanese)*, J. of Japanese Soc. for AI, Vol.3, No.2, pp.206-215, 1988

[50] Yamada S., Tsuji S. and Abe N. *DSBG: Direct solvability-based generalization – Defining operational SOLVABILITY concept– (in Japanese)*, J. of Japanese Soc. for AI, Vol.3, No.6, pp.783-791, 1988

[51] Yamada S. and Tsuji S. *Selective learning of macro-operators with perfect causality (in Japanese)*, J. of Japanese Soc. for AI, Vol.4, No.3, pp.321-329, 1989

[52] Numao M. and Shimura M. *A learning method based on partial structures of explanations (in Japanese)*, Trans. of Inst. of Electronics, Information and Communication Engineers, Vol. J72-D-2, No.2, pp.263-270, 1989

[53] Suwa M., and Motoda H., *Acquisition of Associative Knowledge by Frustration-Based Learning Method in an Auxiliary-line Problem*, Knowledge Acquisition, Vol. 1, No. 1., pp.113-137, 1989

[54] Suwa M., and Motoda H., *Understanding Metaphors by Frustration-Based Learning Method*, Proc. of 4th Knowledge Acquisition Workshop for Knowledge Based Systems, Banff, pp.33_1-33_17, 1989

[55] Kobayashi S. et al, *A Survey of Case-Based Reasoning and Its Applications (in Japanese)*, ICOT Technical Mem., Institute for New Generation Computer Technology, 1990

[56] Haraguchi M. *On inferring incomplete information by analogy*, Proc. of Japan-Czechslovak Joint Symposium on Theoretical Foundation of Knowledge Information Processing, pp.31-40, Inorga Pub., 1990

[57] Nakamura K., and Kobayashi S., *An Interactive Case-Based Reasoning Model and Its Application to Supporting Machine Adjustment (in Japanese)*, Journal of Japanese Society for Artificial Intelligence, Vol. 4 , No. 6., pp.704-712, 1989

[58] Nakamura K., and Kobayashi S., *Roles of Domain Knowledge and User in Case-Based Machine Adjustment Systems*, Proc. of PRICAI'90, 1990

[59] Ishida T., and Tokumaru T., *A framework for Case-Based Diagnosis -Dynamic Frame and Reasoning on the Representation- (in Japanes)*, Trans. of Soc. of Instrument and Control Engineer, Vol.26, No.6, pp.706-713, 1990

[60] Ase H. and Kobayashi S., *A Case-Based Reasoning System for Intelligent Information Retrieval*, Proc. of PRICAI'90, 1990

Explorations in How to Make Application Programming Easier

John McDermott
Geoffroy Dallemagne
Georg Klinker
David Marques
David Tung

Artificial Intelligence Research Group
Digital Equipment Corporation
111 Locke Drive, LM02-1/K11
Marlboro, MA 01752

This paper describes an ongoing research effort to make application programming easier. Our work is based on contributions of several researchers in the area of knowledge acquisition. This work is identified in Section 1. Section 2 describes our approach to making application programming easier and Section 3 relates our approach to the prior work listed in Section 1.

1 Background

This is an informal intellectual history of our work. As with any large body of research, each of the insights emerged gradually through the interaction of a number of researchers. We have artificially singled out individual work for specific attribution of these insights to make the points and examples more clean.

1.1 TEIRESIAS

TEIRESIAS is a software tool developed to study issues in the interactive transfer of expertise [Davis, 79]. It can elicit and encode knowledge for MYCIN's inference engine. (MYCIN is an expert system that assists a physician in diagnosing certain infectious diseases [Buchanan, 84].) TEIRESIAS is invoked when a non-programmer who is developing an expert system is unhappy with the performance of the system. TEIRESIAS exploits the structure of MYCIN's inference engine to determine the inadequacies in the expert system's knowledge. It asks questions that allow it to determine what missing knowledge caused the current problem; it then elicits that knowledge from the non-programmer and encodes it in the form required by MYCIN's inference engine. For our work, the important insights that have emerged from TEIRESIAS are the following:

- *An inference engine can be exploited to guide the elicitation and encoding of knowledge.*
- *Expert system developers can be insulated from the details of implementation by establishing discourse at a conceptual level that abstracts from programming issues.*

1.2 MDX

MDX and its companions RADEX and PATREC are expert systems developed to explore a particular view of how domain knowledge can be decomposed into substructures in expert system design -- namely, the view "that there exist different problem-solving types, ie, uses of knowledge, corresponding to each is a separate substructure specializing in that type of problem-solving" [Mittal, 79; Chandra, 83]. MDX, RADEX, and PATREC each have different problem-solving responsibilities and thus need different kinds of knowledge encoded in ways that make it easy to access when it is needed. MDX is a diagnostic system; its role is to arrive at a classification of a given case into a node of a diagnostic hierarchy. PATREC is a database assistant that acquires patient data, organizes them, and answers the queries of MDX concerning the patient data. RADEX is a radiology consultant to MDX, and it suggests or confirms diagnostic possibilities by reasoning based on its knowledge of imaging procedures and relevant anatomy. For our work, the important contribution of this research is its insightful characterization of the implications of mechanisms for knowledge acquisition:

- *A problem-solving method defines a computational process, requires certain kinds of information as input and produces certain kinds of information as output; it suggests knowledge constructs that the computational process will use as it operates on the inputs to produce the outputs.*

1.3 NEOMYCIN

NEOMYCIN is a medical consultation system developed to explore issues of how expert knowledge can be effectively communicated to non-experts. An initial attempt to use MYCIN's knowledge base as the expert knowledge to be communicated resulted in the discovery that the knowledge base fails to make important structural, strategic, and support knowledge explicit. NEOMYCIN's knowledge base is an extension of MYCIN's where making explicit the knowledge that is only implicit in MYCIN makes it easier for people to understand and modify the system [Clancey, 83]. The work on NEOMYCIN, in which control knowledge is represented abstractly, separately from the domain knowledge it operates upon, resulted in the identification of a knowledge level method: heuristic classification. This method uses knowledge in a few clearly defined ways [Clancey, 85]. For our work, the important insights that have emerged from this work are the following:

- *Since a piece of knowledge may have many different uses, it is important to represent it in a way that allows it to see its relevance for whatever subset of those uses it might encounter.*
- *Representing control knowledge abstractly illuminates the uses to which pieces of knowledge might be put.*

1.4 ROGET

ROGET is a software tool developed to study issues of how to determine the conceptual structure (ie, the nature of the inference engine) appropriate for a particular task. "After identifying the primary problem-solving task, ROGET interacts with the expert to identify a set of domain-specific counterparts to these abstract categories. These acquired terms and their inference support relationships form a skeletal design for the new expert system, the conceptual structure. The conceptual structure indicates, on the basis of the problem-solving task type, the advice or results of the system (the important goals and their subgoals that constitute the solution of the problem-solving task), the evidence or data available to the system (the frequently encountered facts, observations,

and constraints that are required to solve the majority of cases), and the possible inference steps that link evidence to advice" [Bennett, 85]. For our work, the important insight that has emerged from ROGET is the following:

- *A problem-solving method can be exploited to guide the elicitation and encoding of knowledge.*
- *It may be possible to identify an appropriate problem-solving method for any task based on what a domain expert who is not a programmer knows about that task.*

1.5 EES

EES is a software tool developed to explore a paradigm for expert system development that will result in expert systems that can produce plausible explanations of their behavior. EES embodies the view that "explanations produced solely from performance expertise will be inadequate because such explanations exclude the rationale upon which the expert system's behavior is based. Three kinds of expertise must be modeled to provide adequate explanations: knowledge of terminology, domain descriptive knowledge, and abstract problem-solving knowledge" [Swartout, 87]. For our work, the important insight that has emerged from EES is the following:

- *An expert system is easier to maintain if each of the various kinds of expertise relevant to its task is represented explicitly.*

1.6 MORE

MORE is a knowledge acquisition tool developed to explore how to determine what knowledge might be missing from a knowledge base and how to formulate questions that will elicit that missing knowledge from experts. MORE's knowledge elicitation procedures were identified, in part, by reviewing the knowledge-engineering techniques used in the development of MUD, a drilling fluids diagnostic and treatment system [Kahn, 84]. MORE's basic strategy is to consider the various constellations of problems and symptoms that it knows about and then to try to elicit knowledge that will allow it to clearly differentiate among those constellations [Kahn, 88]. For our work, the important insight that has emerged from MORE is the following:

- *A problem-solving method imposes roles on the knowledge it uses, and the existence of these roles can be used to guide knowledge acquisition.*

1.7 MOLE

MOLE, like its predecessor MORE, is a knowledge acquisition tool developed to explore how to determine what knowledge might be missing from a knowledge base and how to formulate questions that will elicit that missing knowledge from experts. MOLE, however, in addition to exploiting its understanding of the roles knowledge can play, also exploits feedback about the mistakes an application program it has built makes when it is used [Eshelman, 88]. MOLE has been used to build several diagnostic expert systems. For our work, the important insight that has emerged from MOLE is the following:

- *Knowledge acquisition can be guided very effectively when the mistakes that an application program makes are used, in conjunction with an understanding of the roles knowledge plays in that program, to diagnose what knowledge is missing.*

1.8 SALT

SALT is a knowledge acquisition tool developed to determine whether problem-solving methods could be exploited for knowledge acquisition in constructive tasks. Most of the knowledge acquisition tools developed prior to SALT were tools to acquire knowledge for selection tasks, and the methods these different tools presupposed were all quite similar. SALT was successfully used to develop two significantly different constructive expert systems -- an elevator system configurer and a flow shop scheduler. Part of the knowledge that was necessary to build these systems was control knowledge -- in particular, knowledge of the order in which constraints are to be relaxed in an over-constrained task [Marcus, 88]. For our work, the important insights that have emerged from SALT are the following:

- *All task-specific methods impose roles on the knowledge they use and it appears that, whatever the method, these roles can be exploited for knowledge acquisition.*
- *Some of the knowledge that needs to be acquired for constructive tasks is control knowledge.*

1.9 KNACK

KNACK is a knowledge acquisition tool developed to explore possible interplay between problem-solving methods and domain models. It can be used to build report generators. KNACK has been used to create report generators for six different tasks; (one of the tasks is to produce a report that comments on the adequacy of the design of an electro-mechanical system from the perspective of how hardened it is against nuclear effects; another task is to generate a business plan). Application program developers experienced different degrees of success in using KNACK for the six tasks. The work on KNACK included trying to determine why KNACK provided different amounts of help on the six tasks [Klinker, 88]. For our work, the important insight that has emerged from KNACK is the following:

- *Features of a task can be used to predict the appropriateness of a particular problem-solving method for that task.*
- *Problem-solving methods can be composed to form a more complex method that covers an additional set of tasks.*

1.10 LEAP

LEAP is a knowledge acquisition tool developed to explore the possibility of an expert system acquiring knowledge through apprenticeship. LEAP directly assimilates and generalizes new knowledge by observing and analyzing the problem solving steps contributed by its users through their normal use of the system. It generalizes by analyzing the particular example that a user has stipulated as an appropriate step at the current point in the task. More specifically, it first explains to itself why the step is appropriate given its theory of the task domain and then generalizes from the example by retaining only those features that were mentioned in its explanation [Mitchell, 85]. For our work, the important insight that has emerged from LEAP is the following:

- *Knowledge acquisition can be guided by a domain theory.*

1.11 CYC

CYC is a knowledge base of real world facts, heuristics, and methods for efficiently reasoning over those facts and heuristics; it is being developed to explore two hypotheses about the role of knowledge in problem solving. The base hypothesis is "if a

program is to perform a complex task well, it must know a great deal about the world in which it operates"; an extension to this hypothesis is "there are two additional abilities necessary for intelligent behavior in unexpected situations: falling back on increasingly general knowledge, and analogizing to specific but far-flung knowledge" [Lenat, 87]. For our work, the important insights that we expect to emerge from CYC are the following:

- *Knowledge can be most easily acquired when the acquiring agent already has a great deal of knowledge.*
- *In order to be non-fragile, agents must be able to identify important commonalties between a current unfamiliar situation and some prior situation in which they behaved appropriately.*

1.12 PROTEGE

PROTEGE is a software tool developed to demonstrate that it is possible to create a model-building tool that can help people who are competent in a particular type of task, identify important relationships among concepts which are central to that type of task. PROTEGE also helps those experts fashion a general task model that can be operated on by a particular problem-solving method such that that model can then be used by a second model-extending tool to permit experts to define specific applications. PROTEGE "allows a system builder to create an explicit model of a set of application tasks that can be solved by skeletal-plan refinement. PROTEGE then automatically generates a knowledge-acquisition tool like OPAL that is custom-tailored for the set of application tasks that was modeled" [Musen, 89]. PROTEGE has been used to generate pOPAL which has in turn been used to generate ONCOCIN, a consultation system for cancer chemotherapy. For our work, the important insight that has emerged from PROTEGE is the following:

- *In some cases, knowledge that would be elicited by many knowledge acquisition tools can be elicited once by a more generic knowledge acquisition tool and then passed on, by that tool, to more specialized tools.*

1.13 ASK

ASK is a knowledge acquisition tool developed to explore issues in the acquisition of control knowledge. The knowledge acquisition tools already discussed in this section assume that much of the control knowledge required for a particular type of task can be specified without regard to the particular situations that arise in performing any particular instance of the task type. However, for at least some types of tasks, this assumption clearly does not hold. ASK distinguishes between "strategic knowledge" and other varieties of control knowledge. "Strategic knowledge is used by an agent to decide what action to perform next, where actions have consequences external to the agent" [Gruber, 89]. For our work, the important insight that has emerged from ASK is the following:

- *Strategic knowledge can be acquired incrementally (and thus augment whatever control structure has been presupposed) because the consequences of an agent having or not having various pieces of strategic knowledge is directly observable by other (more strategically knowledgeable) agents.*

1.14 RIME

RIME is a software engineering methodology devised to help make the on-going development of expert systems less difficult. RIME provides a variety of conceptual structures for imposing coherence on an application program. It encourages the

developer to decompose the task (ie, define subtasks) on the basis of the computational mechanisms that will be employed in solving the problem. It encourages the developer to distinguish among three types of control: algorithmic (control that is situation independent), deliberate decision making (situation-dependent control), and situation-recognition (control that determines that the pre-conditions that must be satisfied in order for some action to be enabled do in fact hold). And finally, it encourages the developer to describe the mechanisms that will be employed so there is a standard against which to compare the implementation [Bachant, 88; Bachant, 89]. RIME has been used to reimplement XCON and to develop XNET, a network configurer. For our work, the important insights that have emerged from RIME are the following:

- *Control knowledge comes in several varieties and distinguishing among the varieties substantially enhances the intelligibility of a program.*
- *Tailoring a control structure to a task is like creating a special purpose language for the task.*

1.15 ACCORD

ACCORD is a task-specific knowledge base developed to explore ways of exploiting domain knowledge for reasoning about problem solving within the context of a particular kind of task. "ACCORD provides a framework for building arrangement-assembly systems in particular domains and for empowering those systems to control, explain, and learn about their own actions more effectively than they could otherwise" [Hayes-Roth, 88]. ACCORD has been used in the development of two application programs. For our work, the important insight that has emerged from ACCORD is the following:

- *Even though two tasks are of the same type, the peculiarities of each may mean that the control structures most appropriate for each are quite different.*

1.16 TAQL

TAQL is a language for mapping task domains onto a particular computational model; it was developed to explore the conceptual closeness between various task domains and that computational model. TAQL assumes the following paradigm of expert system development: an expert with extensive knowledge of some task communicates the knowledge required to perform the task to a knowledge engineer. The knowledge engineer then has the task of mapping the task description given in the language of the expert (eg, English) into an "operationalized" task description (ie, a task description in a form that enables an agent to perform the task) [Yost, 89]. TAQL has been used to create a version of R1-Soar [Rosenbloom, 85] from an English description of R1-Soar's task. For our work, the important insight that has emerged from TAQL is the following:

- *Given an appropriate computational model, an application program can be created directly from a task level description.*

2 Our Approach

In order to give insight into the technology that we believe will make programming easier, we are building three tools: Spark, Burn, and FireFighter. Spark interviews a developer and on the basis of what the developer says about the information available for and the results desired from a task, Spark will select, from a library of pre-defined computational mechanisms, one or more mechanisms that can collectively produce the

results desired on the basis of the information available. Spark will then configure those mechanisms into a problem-solving method.

Burn will retrieve, from a library of knowledge acquisition tools, the tool associated with each of the mechanisms selected by Spark. It will use those tools to elicit expertise from the developers and produce an application program by encoding that expertise in a way that will allow it to be used by the problem-solving method previously configured by Spark.

Whenever the application program is used by one of the developers, FireFighter will ask whether the program has performed correctly and if it has not, will invoke either Burn or Spark to add knowledge to the program or to modify the problem-solving method on which the program is based.

Figures 2-1 through 2-4 show the interrelationships among Spark, Burn, and FireFighter. Figure 2-1 shows Spark using information provided by the developer about the task to be automated and the domain in which the task will be performed to select a set of four mechanisms that are adequate for some variants of the task in that domain. Spark then uses additional information about the task and the domain to make sure that it is at least plausible that the collection of mechanisms will work for this variant of the task. Spark passes the names of the selected mechanisms to Burn which configures a set of knowledge acquisition tools (one corresponding to each mechanism) and those tools then use additional information about the task and the domain to build up a knowledge base for each of the mechanisms selected by Spark. The result is an application program. In this case FireFighter determines that the application program is not at all what the developer had in mind.

Figure 2-2 shows FireFighter reinvoking Spark, and Spark selecting an alternative configuration of mechanisms. After Spark determines that this set of mechanisms is plausible for this variant of the task, it passes the names of the mechanisms to Burn. Two of the mechanisms are the same as in the previously tried configuration; the other mechanism was not used in the previous case. Burn configures three knowledge acquisition tools corresponding to the three mechanisms, but in doing so, preserves two of the knowledge bases built up the first time around. Burn invokes the knowledge acquisition tool corresponding to the previously unused mechanisms and builds up a knowledge base for it.

The result is a new application program that tackles the task somewhat differently and provides different functionality. In this case FireFighter determines that the application program is much closer to what the developer had in mind. However one piece of desired functionality is missing.

Figure 2-3 shows FireFighter reinvoking Spark, and Spark extending the configuration of mechanisms with a fourth mechanism from its mechanisms library. Burn is then invoked, and a new application program is produced. In this case FireFighter determines that the program does not yet have an adequate amount of knowledge. Figure 2-4 shows FireFighter reinvoking Burn and knowledge being added to the knowledge bases until an acceptable verison of the application program is produced.

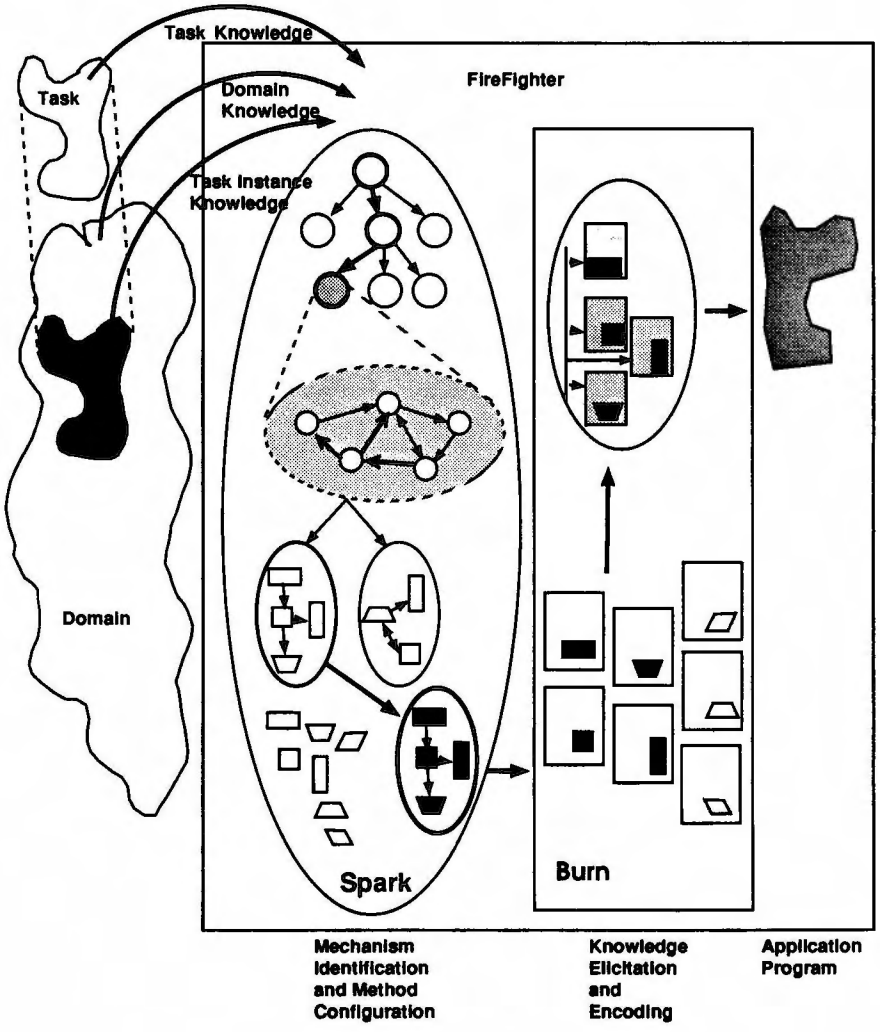


Figure 2-1 Easy Programming Technology

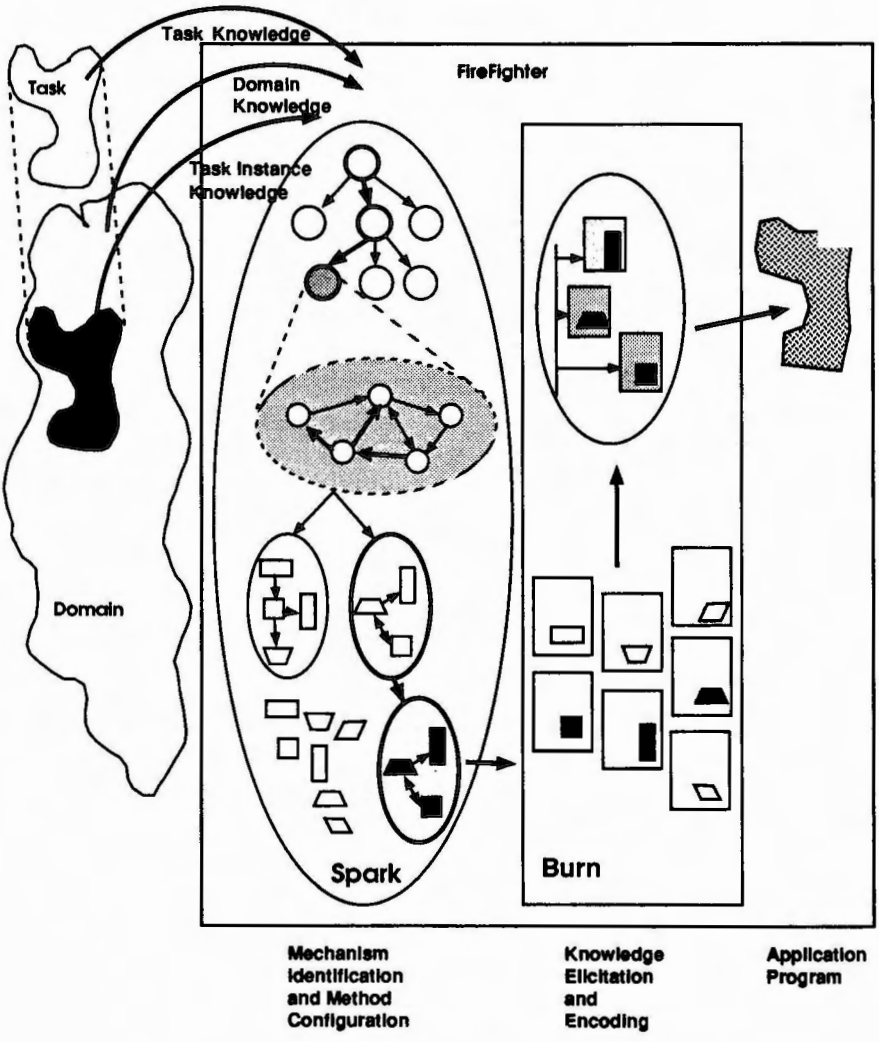


Figure 2-2 Easy Programming Technology

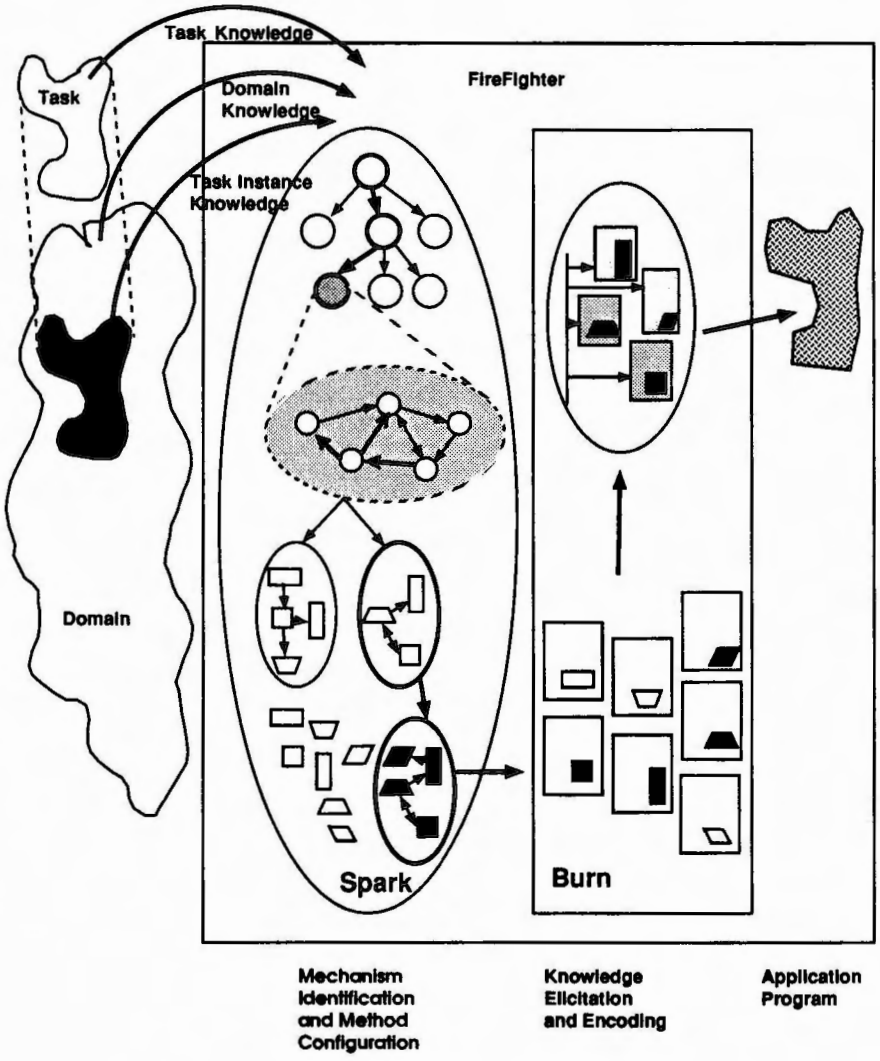


Figure 2-3 Easy Programming Technology

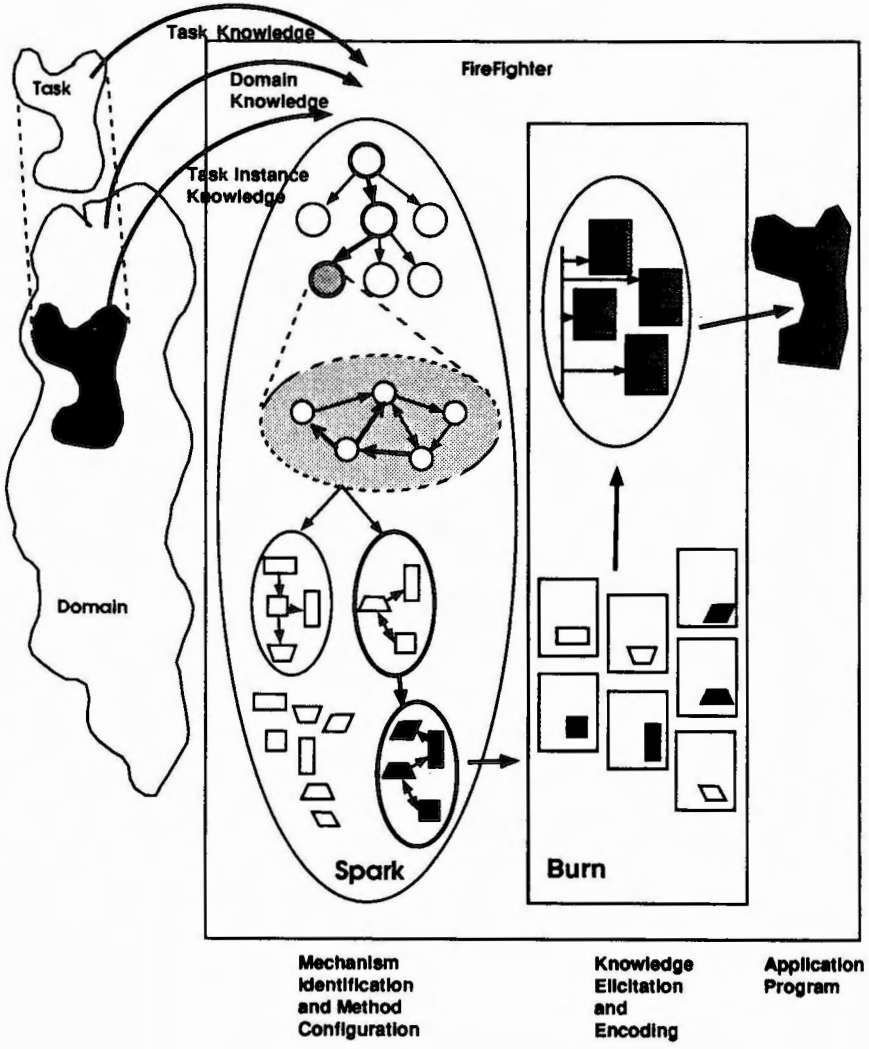


Figure 2-4 Easy Programming Technology

3 Our Work in the Context of the Work We are Building on

3.1 Spark

Spark is the most novel of the three systems we are developing. It takes some inspiration from Chandrasekaran's generic tasks and from Bennett's ROGET. Yost's work on TAQL provides a demonstration of how an application program can be generated from a knowledge level description. Finally the KNACK work provides evidence that task features can be used to predict the appropriateness of a particular problem-solving method for a task. The insights that we believe will emerge from Spark are the following:

- *Based only on characteristics of a task that are obvious to non-programmers, appropriate pre-defined computational mechanisms can be identified and configured to form a method for performing the task.*
- *The number of mechanisms required to automate all tasks is reasonably small or most of the required mechanisms can be created automatically (e.g. by slight modifications to existing mechanisms).*

3.2 BURN

Burn is essentially a manager of knowledge acquisition tools. Knowledge acquisition tools elicit knowledge from human experts and encode that knowledge in an application program. The knowledge acquisition tools Burn manages will not be significantly different from tools that have been developed by a variety of researchers over the past several years. Burn's contribution is that it collects a number of knowledge acquisition tools together so that more complex application programs can be developed. The knowledge acquisition tools in Burn's library are particularly indebted to the insights of the researchers who developed TEIRESIAS, MORE, SALT, ASK, and PROTEGE. In addition, the work on LEAP provides many valuable ideas for how to exploit a domain model in acquiring knowledge. The insight that we believe will emerge from Burn is the following:

- *Given a set of configured mechanisms, pre-defined knowledge acquisition tools associated with those mechanisms can be used to elicit expertise from a non-programmer and to encode that expertise, thereby creating an application program to perform the task.*

3.3 FireFighter

FireFighter's diagnostic task (determining whether an inadequacy in an application program is due to missing expertise or to an inappropriate control structure) is relatively straightforward; the task of prescribing treatment is more challenging. With respect to prescribing treatment, FireFighter builds on the results from MOLE dealing with identifying missing expertise. FireFighter also builds in various ways on the lessons from RIME, ACCORD, NEOMYCIN, EES, and CYC. Each of the first four of these stresses the importance of making the various kinds of knowledge used by the problem-solver explicit. RIME and ACCORD in addition focus on the complexity of the world and on the need to tailor each application program to the peculiarities of the specific task it is automating. CYC stresses the importance of having knowledge in order to acquire knowledge and (complementing RIME and ACCORD) of being able to adapt structures that don't quite fit the task at hand. The insight that we believe will emerge from FireFighter is the following:

- *Given an indication from a non-programmer that an application program is inadequate, it is possible to determine whether the inadequacy is due to missing expertise or to an inappropriate method and then to remedy the inadequacy.*

Acknowledgments

Helpful comments on earlier versions of this plan were made by the other members of AIRG, as well as by Jim Bennett, Bill Clancey, Allen Newell, Mark Stefik and Gregg Yost.

References

[Bachant, 88] Bachant, J. RIME: Preliminary Work Toward a Knowledge Acquisition Tool. In S. Marcus (ed), *Automating Knowledge Acquisition for Expert Systems.* Kluwer, 1988.

[Bachant, 89] Bachant, J., and E. Soloway. The Engineering of XCON. *Communications of the ACM*, **32**, 3, 1989.

[Bennett, 85] Bennett, J. ROGET: A Knowledge-Based System for Acquiring the Conceptual Structure of a Diagnostic Expert System. *Journal of Automated Reasoning*, **1**, 1, 1985.

[Buchanan, 84] Buchanan, B., and E. Shortliffe (eds). *Rule-based Expert Systems.* Addison-Wesley Publishing Company, 1984.

[Chandra, 83] Chandrasekaran, B. Towards a Taxonomy of Problem Solving Types. *AI Magazine*, **4**, 1, 1983.

[Clancey, 83] Clancey, W.J. The Epistemology of a Rule-Based Expert System -- a Framework for Explanation. *Artificial Intelligence*, **20**, 3, 1983.

[Clancey, 85] Clancey, W.J. Heuristic Classification. *Artificial Intelligence*, **27**, 3, 1985.

[Davis, 79] Davis, R. Interactive Transfer of Expertise: Acquisition of New Inference Rules. *Artificial Intelligence*, **12**, 2, 1979.

[Eshelman, 88] Eshelman, L. MOLE: A Knowledge-Acquisition Tool for Cover-and-Differentiate Systems. In S. Marcus (ed), *Automating Knowledge Acquisition for Expert Systems.* Kluwer, 1988.

[Gruber, 89] Gruber, T. Automated Knowledge Acquisition for Strategic Knowledge. *Machine Learning*, **4**, 3/4, 1989.

[Hayes-Roth, 88] Hayes-Roth, B., M. Johnson, A. Garvey, and M. Hewett. Application of the BB1 Blackboard Control Architecture to Arrangement Assembly Tasks. *Artificial Intelligence*, **1**, 2, 1986.

[Kahn, 84] Kahn, G., and J. McDermott. The MUD System. *Proceedings of the First IEEE Conference on Artificial Intelligence Applications*, Denver, Colorado, 1984.

[Kahn, 88] Kahn, G. MORE: From Observing Knowledge Engineers to Automating Knowledge Acquisition. In Sandra Marcus (ed), *Automating Knowledge Acquisition for Expert Systems.* Kluwer, 1988.

[Klinker, 88] Klinker, G. KNACK: Sample-Driven Knowledge-Acquisition Tool for Reporting Systems. In S. Marcus (ed) *Automating Knowledge Acquisition for Expert Systems.* Kluwer, 1988.

[Lenat, 87] Lenat, D., and E. Feigenbaum. On the Thresholds of Knowledge. *Proceedings of the Tenth International Joint Conference on Artifical Intelligence*, Milan, Italy, 1987.

[Marcus, 88] Marcus, S. SALT: A Knowledge-Acquisition Tool for Propose-and-Revise Systems. In S. Marcus (ed), *Automating Knowledge Acquisition for Expert Systems.* Kluwer, 1988

[Mitchell, 85] Mitchell, T., T. Mahadevan and L. Steinberg,. LEAP: A Learning Apprentice for VLSI Design. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, California, 1985.

[Mittal, 79] Mittal, S., B. Chandrasekaran, and J. W. Smith. Overview of MDX -- A System for Medical Diagnosis. *Third Symposium on Computer Applications in Medical Care*. Washington D.C., 1979.

[Musen, 89] Musen, M. Automated Support for Building and Extending Expert Models. *Machine Learning*, 4, 3/4, 1989.

[Rosenbloom, 85] Rosenbloom, P., J. Laird, J. McDermott, A. Newell, E. Orciuch. R1-Soar: An Experiment in Knowledge-Intensive Programming in a Problem-Solving Architecture. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7, 5, 1985.

[Swartout, 87] Swartout, W., and S. Smoliar. On Making Expert Systems More Like Experts. *Expert Systems*, 4, 3, 1987.

[Yost, 89] Yost, G. A Problem Space Approach to Expert System Specification. *Proceedings of the Eleventh International Joint Conference on AI*, Detroit, Michigan, 1989.

Justification-based Knowledge Acquisition*

Thomas Gruber
Knowledge Systems Laboratory
Stanford University
701 Welch Road, Building C
Palo Alto, CA 94304
gruber@sumex-aim.stanford.edu

This paper describes a technique for computer-mediated knowledge acquisition based on the elicitation of justifications from people. I start from the theoretical position that the problem of knowledge acquisition arises from the difference between the form of knowledge in programs — computation models—and the form of descriptions natural to a domain. Then I describe two knowledge acquisition problems, and techniques for providing automated assistance for each problem.

The first problem is the acquisition of strategic knowledge, studied in the context of medical diagnosis and therapy. The knowledge acquisition for this problem is embodied in a program called ASK, which asks experts to critique the strategic actions taken by a running expert system and provide *justifications* for why the system should have chosen different actions in similar situations. ASK generalizes this advice into rules that represent strategic knowledge.

The second problem is the acquisition of design rationale for physical devices—knowledge about the purpose of a component in a designed system. In the approach to this problem, called *rationale by demonstration*, the domain specialist conveys the intended function of a device by configuring a simulation of the structure of interest exhibiting the relevant behavior under certain operating conditions. Later, a different user can discern the original design rationale by replaying the simulation, asking follow-up questions in response to explanations generated by the simulation program.

In the final section I draw an analogy between these seemingly different knowledge acquisition techniques, showing how they share similar abstract characteristics. We call the general approach illustrated by these two examples *justification-based knowledge acquisition*. The key source of power for the approach is how it divides the load in knowledge acquisition between a cooperative user/teacher and the elicitation program.

1. The Problem of Knowledge Acquisition as Representation Mismatch

I view knowledge acquisition *broadly*, as the human activity of creating computational models to automate some performance task or achieve some communicative purpose. In our programs computational models are embodied in knowledge bases and program code; they are machine-intelligible representations that refer to objects, processes, strategies, and other aspects of a domain to be modelled. This view of knowledge acquisition as model construction is prevalent in the literature [Clancey, 1989].

People create computational models for some of the same reasons they create models in other representational media: to record and communicate something worth knowing. In

Copyright © 1990 Thomas R. Gruber. All rights reserved. Support for the author is from NASA grant NCC 2-537 and DARPA. grant NASA NAG 2-581, ARPA order 6822.

traditional knowledge systems the knowledge is use to automate some process that uses the knowledge. So one unique property of computational models is that they can be used for performance tasks, such as automated diagnosis. Other properties of the computational medium are that it is possible to *systematically evaluate* computational models for *consistency*, and sometimes *completeness*. We can build tools that perform these services for knowledge that has been formulated in computational terms. However, these tools do not address the problem of how to formulate the computational models in the first place.

The problem is that building computational models is quite different from writing textbooks or giving lectures, as is obvious to all anyone who has ever built a knowledge system. This is because the language of computational models—being formal, unambiguous, and operational—limits and shapes the domain of discourse. Here lies the essence of what makes knowledge acquisition difficult to automate: the mismatch between the form in which knowledge is described in the world and the form in which is it represented in programs. This has been called *representation mismatch* [Buchanan et al, 1983; Gruber and Cohen, 1987; Gruber, 1989b].

The result is that in practice, knowledge acquisition is the province of people who can understand the language of the domain *and* the language of computation. By "understand the language" I mean be aware of the representational opportunities afforded by the language and constraints imposed by the medium. For example, a knowledge engineer can see a classification task—a computational perspective—in the physician's description of medical diagnosis, and recognize a problem of representing utility—a constraint on making advice operational—when the expert starts talking about "doing things in the best order."

An ideal solution for the knowledge acquisition problem is to make the languages and techniques of computational modeling match the language natural to a domain, allowing the domain specialist to build programs directly. This ambitious goal has motivated important advances throughout the short history of computer science. For example, Fortran was invented for the expressed purpose of allowing scientists and engineers to build computational models in the language they know: mathematical formulae. John McCarthy invented Lisp and launched the paradigm of logic in AI because he wanted to build an advise taker: a program that could take advice in the language of the domain rather than the language of Von Neuman machines.

Since the domains of most knowledge systems are not naturally described in terms of equations or the lambda calculus, the conceptual distance between descriptions available from domain specialists and computational models that constitute knowledge systems is an inherent problem. Therefore, we need to come up with intermediate solutions, bringing the modelling language as close as possible to the domain and task, and bridging the remaining gap with interactive programs that elicit knowledge in the form it can be obtained from the user, and assimilate it into a predesigned computational model. This is the approach taken in the method-, task-, and domain-specific knowledge acquisition tools (e.g., [Boose and Gaines, 1988; Marcus, 1988; Musen, 1989]).

Given this perspective, we turn to the first knowledge acquisition problem, the acquisition of strategic knowledge.

2. The Acquisition of Strategic Knowledge

Consider the classic expert system for medical diagnosis. There are data and disease hypotheses, and some heuristic mapping between the two. If you build a generic backward-chaining

classification system and show it to a practicing physician, what's the first thing they tell you is missing?

One might hear answers like "It sure is dumb; people don't ask questions in that order." Or "The program *assumes* that you have already done all the tests. Deciding what to *do* is the hard part; anyone can read the lab results." What is missing is *strategic knowledge:* knowledge about what action to do next, when one's knowledge of the world is incomplete and uncertain and the effects of actions cannot be predicted with certainty.

In clinical practice, evidence for a diagnosis is gathered over time, before the data are in. The diagnostic and therapeutic actions that produce evidence are chosen strategically, balancing factors such as diagnostic power, therapeutic value, and cost. Strategic knowledge underlies the physician's expertise at choosing the next question to ask, diagnostic test to run, or therapy to try. In different cases the physician performs different sequences of actions, *and order matters.* In this application, domain-specific strategic knowledge is essential to performance. It is control knowledge at the knowledge level [Newell, 1982], distinguished from search-control knowledge, which is about computational performance at the symbol level. If strategic knowledge is part of the expertise in a domain, then we need to acquire it from the experts.

ASK - A Program that Acquires Strategic Knowledge

ASK is an interactive program for acquiring domain-specific strategic knowledge from experts. It is described in [Gruber, 1989a, 1989b]. Here's how it works.

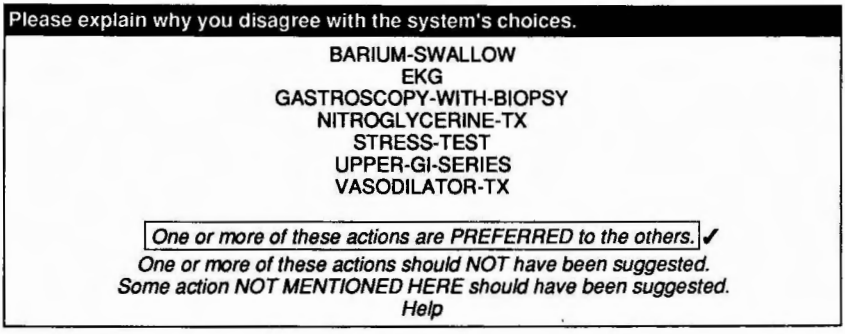
ASK operates in a *context* in which strategic choices are made. In this case the context is a running knowledge system for medical diagnosis and therapy, where the system chooses among diagnostic and therapeutic actions.

The user *demonstrates* to ASK how she would make a strategic choice—what actions she would and would not take in a given situation. In the context of the running knowledge system, the user interrupts the system and critiques a choice of actions.

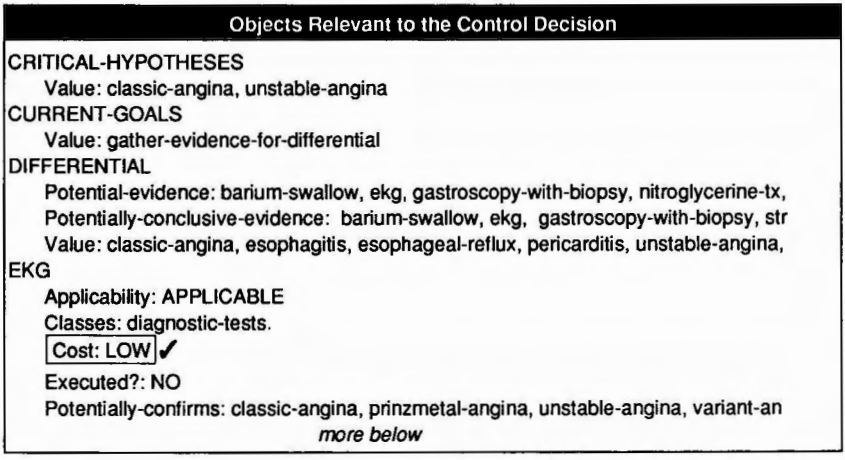
Then ASK elicits, from the user, *justifications* for the strategic decision. The justifications are reasons for choosing one action over another, which the user specifies by selecting *relevant features* of the current situation (state of the application) as presented on the screen. ASK can then use this information to *operationalize* the advice: that is, to incorporate the strategic knowledge exemplified by the justified case into the knowledge base of the system.

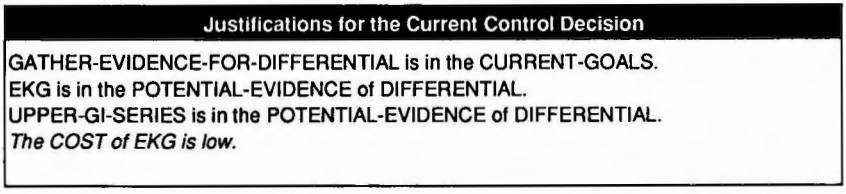
Let's go through an example of a knowledge system for the diagnosis and treatment of chest pain. In this example, the system is part-way through a session of patient complaining of chest pain. The system has gotten to a point where it needs to request some "hard" data: that is, data that isn't free. Given the patient's symptoms and history, there are seven actions suggested. (The system can't decide among them, so it leaves the decision of which of the seven to choose to the application user) The expert, playing the role of the application user, wants the system to be more discriminatory in selecting tests and treatments in the situation. Therefore the expert interrupts the normal execution of the application, and critiques the choices.

ASK then elicits positive and negative examples: actions that should and should not be chosen in this situation. In this case, the expert indicates that instead of recommending all seven actions as equally appropriate, the system should have preferred the EKG over alternative actions, such as Upper-GI-Series.



Then ASK elicits *justifications* for the decisions. Justifications are reasons for choosing the positive exemplar over the alternative. ASK presents a justification interface shown below.





The screen contains two windows of special interest. One is the *relevant objects window*. It is presents a view of objects in the current knowledge base that might be relevant to the strategic decision, and their features. Features are static and dynamic properties, some of which are inferred (e.g., potential-evidence for hypotheses on the differential).

For example, ASK assumes that the features of the positive example EKG are plausibly relevant to choosing that action, so ASK presents the current values of EKG in the relevant objects

window. The user can add other knowledge base objects to the relevant objects window if ASK failed to guess that they might be relevant.

The other window is the *justifications window*. It contains an explanation of why one action is preferred in this case. ASK seeds the justifications window with *the system's reasons* for choosing actions. For example, one of the reasons for selecting these seven actions in the current situation is that they are all elements of the potential-evidence(differential) feature. The explanations are always ground instances (no variables), so we see in the justification window the statement that EKG is in the potential-evidence set.

The user creates additional justifications by pointing out *relevant features* of the current situation, as presented in the relevant objects window. For example, in this example the user has indicated that the cost of EKG was a factor in preferring it over the alternatives. That selection is reflected in the sentence "The cost of EKG is low" added to the justifications window by ASK. The user may also delete justifications.

The user continues modifying the list of justifications until she is satisfied that it explains why the one action should be chosen, and ASK is satisfied that the given justifications can actually distinguish the two actions in the current knowledge system. In this case, the "cost" feature is sufficient to distinguish EKG from the negative example, Upper-GI-series.

Then ASK operationalizes and generalizes the justifications, forming a strategy rule that becomes part of the knowledge base. In similar situations in the future, the knowledge system will use the new rule to choose among actions.

Summary of ASK's Elicitation Technique

I have skipped over a lot of the details and capabilities of ASK, and concentrated on the elicitation of justifications. At this point, let me me summarize the key properties of ASK's technique for acquiring strategic knowledge by eliciting justifications.

1. For every question the machine asks the human, the **operational semantics** of the answers are known to the system. (Features are well-defined attributes, functions, and relations in the KB.)

2. What to elicit depends on the **role** of the knowledge (choosing among actions) and the **knowledge representation** (strategy rules).

3. Justifications are elicited in **context of use** (strategic decisions in a running system).

4. User selects what is **relevant** among available features (she doesn't write a procedure).

5. Machine constructs operational **explanation** from elicited input (justification statements are generated from selected features).

6. User must be able to **extend the language** of relevant features (ASK has an interface for defining new features).

Now let's turn to the second knowledge acquisition problem, the acquisition of knowledge about how things work. This is work in progress at the Stanford Knowledge System Lab.

3. Acquisition of Design Rationale via Machine-Generated Explanations of Simulations

The problem of *design knowledge capture* is how to acquire, in a reusable form, the human knowledge underlying a design, including the structure of the artifact, the rationale for choosing particular components or implementation approaches, the assumptions about the context in which the artifact is to be used, and the institutional experience with designing and manufacturing similar artifacts (e.g., design case histories). One approach to the problem is to elicit structured text about the design rationale, based on argument-style discussions among designers [Conklin and Begeman, 1987]. Another is to treat designs as sets of decisions, and to elicit reasons pro and con for parts of the design in a decision-theoretic or similar framework [Shema, Bradshaw, Covington, and Boose 1989]. A third approach is to record the history of design choices and provide intelligent indexing into a shared design memory [Mark and Schlossberg, 1990]. A fourth approach is to acquire design knowledge in the form of machine-intelligible models [Baudin, Sivard, and Zweben, 1989]. The technique I will described here takes this last course: to acquire device models that can be used to generate textual and graphical explanations about how and why the device works. This approach frames design knowledge capture as a knowledge acquisition problem, where the challenge is to get operational knowledge from domain specialists.

Explainable Simulation as a Communication Medium

In particular, we are building a medium in which people can describe design rationale by demonstrating the intended behavior of a device using an explainable simulation. Details of the approach are available in [Gruber and Russell, 1990].

In engineering practice, simulation is used to predict the behavior of engineered artifacts. A simulation requires behavior models, typically in the form of equations, that describe the behavior of individual components and general processes. The simulation predicts the system behavior that arises from the interactions of components and the operating environment. The formulation of behavior models from structural descriptions is not automated; simulations are set up by engineers to answer some class of questions about artifact behavior.

Using simulation to answer questions about the behavior of an artifact reveals knowledge about the designer's intent. The specific models that are created and the input conditions that are specified describe a behavior of interest and intended context of use. The simulation serves as a demonstration of intended device behavior.

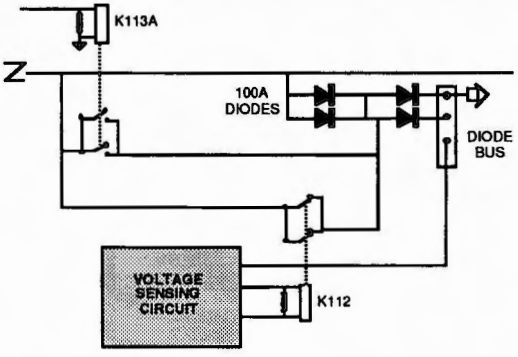
One can employ a computer simulation as a communication medium for design rationale. The basic approach is to arrange for a knowledgeable user to set up simulation scenarios, configuring and guiding an simulation tool so that the predicted results account for the phenomenon to be communicated. In particular, in the technique called *rationale by demonstration*, the user documents the intended function of a bit of artifact structure under given conditions. [Gruber, 1990]

If a person can demonstrate a behavior by constructing an operational simulation scenario, then the machine has enough information to replay the scenario to another person. Instead of writing text describing a device, one engages in a dialog with a knowledge-based modeling and simulation environment, telling the machine what *it* needs to know to generate a demonstration of some behavior of interest. The user identifies the *relevant* structure and *intended* behavior by

constraining the *possibly* relevant structure and behavior within the closed world defined by the modelling primitives. The machine generates explanations based on the outcomes of simulation. The user essentially manipulates the simulation scenario until what the machine says corresponds to the intended message. The result is a machine-intelligible model of design rationale. Because the simulation is generated from underlying models, it can be used to answer questions not typically anticipated by static documentation, and can be effectively indexed with other knowledge about the device.

An Example: Explaining the Purpose of a Voltage Limiter

Consider, for example, the following text fragment from a NASA document explaining the purpose of a piece of a circuit in the electrical power system of the Hubble Space Telescope:



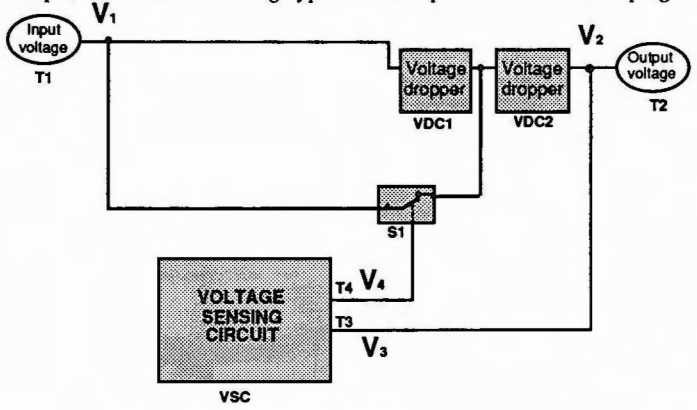

```
5.1.5 Voltage Limiter

A voltage limiting circuit is provided in each diode bus feeder to
limit the voltage at the using equipment to 32.0 V.  With the SSM
design change to six 23-cell, type 44 batteries the user input
could exceed the 32 V user specification.  To prevent this, an
additional diode pair (parallel 100A diodes) was added in series
with the load and bypassed by a relay that is operated by a
voltage sensing circuit.  When the diode bus voltage exceeds 34.15
± 0.2 V the bypass relay opens and inserts the diode in the
circuit as a voltage dropping element.  When the voltage falls to
31.41 ± 0.2 V the relay will reclose, bypassing the diodes.  The
voltage sense circuit board used in this circuit is essentially
the same as the circuit used in the charge current controller
except for the set point voltage.  Figure 5-1 shows one of the
voltage limiter circuits and the associated controls.
```

Note that the human explanation is *Imprecise* and *inconsistent* ("limit the voltage to 32.0 V" ≠ "keep it between 34.12 and 31.41 V"); (2) *incomplete* ("how does inserting diodes accomplish the voltage-limiting function?") ("what are the intended operating conditions?"); and (3) *static* ("why 2 diodes?") ("what are the other diodes doing there?")

No system or document would be able answer all the questions one could think of concerning this circuit. However, an explanation generated from an underlying model of the device would have available considerably more detail. A pre-enumerated set of queries could be supported, organized by types of components, connections, processes, etc. At knowledge

acquisition time, this knowledge about possible query types for a given model can support completeness analysis.

For example, consider the following hypothetical output from a simulation program:[1]



The voltage limiting circuit VLC

```
A purpose of the voltage limiting circuit VLC is to prevent the
output voltage V2 at diode bus terminal T2 from exceeding the
maximum allowed using equipment voltage MAEV, which is 32.0 V.
The voltage limiter VLC works as follows:
When  output voltage V2 > threshold voltage TV1,
      and the bypass relay S1 is closed,
there is current flow from input terminal T1 through relay S1
      through voltage dropping circuit VDC2 to terminal T2,
and the input voltage V1 = voltage V2 + voltage drop VD2 across
      voltage dropping circuit VDC2.
Since terminal T2 is electrically connected to the terminal
      T3 of the voltage sensing circuit VSC,
the voltage V3 at terminal T3 = voltage V2.
Since voltage V3 > threshold TV1,
the voltage sensing circuit VSC outputs voltage V4 > 0V at
      terminal T4,
which causes relay S1 to open.
There is now current flow from terminal T1 through voltage dropper
      VDC1 through voltage dropper VDC2 to terminal T2.
Assuming voltage V1 is constant or decreasing,
then  voltage V2 will decrease by at least voltage drop VD1 across
      voltage dropper VDC1.
```

By mousing on various parts of the text, the user could ask follow up questions, such as "What is the value of TV1?" and "What caused the current flow from T1 through VDC1 through VDC2 to T2?"

To summarize, this is a technique for eliciting a modeling of the purpose of design by *demonstrating* a scenario in which the device goes through the intended behaviors. The elicitation

[1]Although the technology for explainable simulation has been demonstrated (Forbus and Falkenhainer, 1990), the example shown below is currently not implemented.

tool needs to get from the user the relevant structure of interest, the behaviors of interest, and the conditions specifying the situation in which the structure gives rise to the intended behavior. Part of the elicitation task is asking for implicit assumptions and conditions which need to be stated to do the simulation, but which may have been overlooked in a textual description. A design justification is constructed by getting the machine to explain the pertinent aspects of the simulation.

Now let's examine the technique in terms of the key properties identified for ASK's justification method. In this task, justifications are explanations mapping specific features of structure and behavior to purpose.

Elicitation of Design Rationale as Justifications

1. For every question the machine asks the human, the **operational semantics** of the answers are known to the system. (The user specifies what is relevant by constraining an explicit model used for simulation).

2. What to elicit depends on the **role** of the knowledge (specifying structure and behavior for simulation) and the **knowledge representation** (structure and process models).

3. Justifications are elicited in **context of use:** a *simulation scenario* demonstrating how a part of the design enables (prevents) the intended (avoided) behavior.

4. User selects what is **relevant** among available features (the relevant aspects of the structure and behavior to be modeled).

5. Machine constructs operational **explanation** from elicited input (the machine generates an explanation from the simulation that relates relevant design features to relevant intended behaviors)

6. User must be able to **extend the language** of relevant features (the user must be able to build and modify device models).

4. Discussion

Thus, we see surprising parallels between ASK's method of eliciting justifications for strategic choices, and the demonstration approach to elicit justifications for a particular design. We call the generalization *justification-based knowledge acquisition*. The key source of power for the approach is how it divides the load in knowledge acquisition between a cooperative user/teacher and the elicitation program. The machine provides the computational medium, including the knowledge representation and the context of use, such that everything that is acquired from the user can be assimilated into the computational model. The human provides access to the real world (outside the inherently closed world of the computational model), by selecting what is *relevant* among the possibilities afforded by the computational model, and extending the language to include terms not in the original vocabulary defined in the machine. The knowledge acquired using the justification technique is guaranteed to be *operational* because the user always conveys something to the machine by getting the machine to say it.

6. Some of the Related Work

Research on the acquisition of strategic knowledge is thoroughly reviewed in [Gruber, 1989]. The idea of using simulation to communicate knowledge of how things work is well established in the simulation-based training field [Brown, Burton, and deKleer, 1982; Hollan, Hutchins, and Weitzman, 1984; Towne and Munro, 1989]. [Forbus and Falkenhainer , 1990] have shown a simulation architecture designed for the expressed purpose of explanation. The use of machine-generated explanations as an *input medium* for acquiring operational knowledge is inspired by work in automated knowledge acquisition [Bareiss, 1989; Gruber, 1989] and human-computer interaction [Yen, Neches, and DeBellis, 1988; Williams, 1984]. The general idea of design knowledge capture by explanation was independently proposed by [Mark, 1988] (see also [Kellog, et al., 1989]).

Acknowledgements

The ASK work was done at the University of Massachusetts with the guidance and influence of Paul Cohen. The design rationale work is a collaboration with Dan Russell at Xerox PARC and the How Things Work project at the KSL with contributions from Yumi Iwasaki, Bob Engelmore, Chee Meng Low, Pandu Nayak, Daneel Pang, James Rice, and Tilda Brown. Brian Falkenhainer, Brian Williams, and Ken Forbus have been very helpful and influential.

References

Baudin, C. Sivard, C. and Zweben, M. (1989). A model-based approach to design rationale conservation. *IJCAI-89 Workshop on Model-based Reasoning.*

Bareiss, E. R. (1989). *Exemplar-based Knowledge Acquisition: A Unified Approach to Concept Representation, Classification, and Learning.* Boston: Academic Press.

Boose, J. H. and Gaines, B. R. (Eds.). (1988). *Knowledge Acquisition Tools for Expert Systems.* Academic Press.

Buchanan, B. G., Barstow, D. K. , Bechtel, R., Bennett, J., Clancey, W., Kulikowski, C., Mitchell, T., & Waterman, D. A. (1983). Constructing an expert system. In F. Hayes-Roth, D. A. Waterman, & D. B. Lenat, (Eds.), *Building Expert Systems.* Reading, MA: Addison-Wesley.

Brown, J. S., Burton, R., and deKleer, J. (1982). Pedagogical, natural language and knowledge engineering techniques in SOPHIE I, II, and III. in *Intelligent Tutoring Systems.* Sleeman and Brown (Eds.), Academic Press.

Clancey, W. J. (1989). Viewing knowledge bases as qualitative models. IEEE Expert 2(4): 9-23. Previously published as Technical Report KSL-86-27, Stanford University.

Conklin, J. and Begeman, M. (1987). gIBIS: A hypertext tool for team design deliberation. *Hypertext-87.*

Forbus, D. K. and Falkenhainer, B. (1990). Self-explanatory simulations: An integration of qualitative and quantitative knowledge. *AAAI-90.*

Gruber, T. R. (1989a). *The Acquisition of Strategic Knowledge.* Academic Press.

Gruber, T. R. (1989b). Automated knowledge acquisition for strategic knowledge. *Machine Learning*, 4, 293–336.

Gruber, T. R., & Cohen, P. R. (1987). Design for acquisition: Principles of knowledge system design to facilitate knowledge acquisition. *International Journal of Man-Machine Studies, 26*(2), 143–159.

Gruber, T. R. & Russell, D. M.(1990). Design knowledge and design rationale: A framework for representation, capture, and use. Stanford Knowledge Systems Laboratory, Technical report KSL 90–45.

Hollan, J. D., Hutchins, E. L., and Weitzman, L. (1984). STEAMER: An interactive inspectable simulation-based training system. *AI Magazine,* 2.

Kellog, C., Gargan, R., Mark, W., McGuire, J., Pontecorvo, M., Schlossberg, J., Sullivan, J. Genesereth, M., and Singh, N. (1989). The acquisition, verification, and explanation of design knowledge. SIGART Newsletter, 108, April 1989, 163-165.

Marcus, S. (1988). (Ed.), *Automating Knowledge Acquisition for Expert Systems,* Boston: Kluwer Academic Publishers.

Mark, W. (1988). Explanation and interactive knowledge acquisition. *AAAI -88 Workshop on Explanation.*

Mark, W. and Schlossberg, J. (1990). Design Memory. Technical report, Lockheed AI Center, 3251 Hanover Street 96-20 B/259, Palo Alto, CA 94304.

Musen, M. A. (1989). *Automated Generation of of Model-based Knowledge-Acquisition Tools,* London: Pitman. Based on doctoral dissertation, Computer Science Department, Stanford University.

Newell, A. (1982). The knowledge level. *Artificial Intelligence, 18,* 87–127.

Shema, D., Bradshaw, J., Covington, S. and Boose, J. Design knowledge capture and alternative generation using possibility tables in CANARD. *Fourth Knowledge Acquisition fore Knowledge-Based Systems Workshop,* Banff, Canada, October 1-6, 1989.

Towne, D. M. and Munro, A. (1989). Tools for Simulation-Based Training. Technical report 113, Behavioral Technology Laboratories, USC.

Williams, M. (1984). What makes RABBIT run? *International Journal of Man-machine Studies,* 21:333-352.

Yen, J., Neches, R., and DeBellis, M. Specification by reformulation: A paradigm for building integrated user support environments. *AAAI-88.*

AKARS-1 : An *A*utomatic *K*nowledge *A*cquisition and *R*efinement *S*ystem

K. S. Leung, B.Sc.(Eng), Ph.D., SMIEEE, MBCS, MIEE, C.Eng

M. L. Wong, B.Sc.

Department of Computer Science
The Chinese University of Hong Kong
Shatin, N.T.
Hong Kong

Expert Systems are definitely one of the major success of Artificial Intelligence. Unfortunately, the knowledge acquisition bottleneck hampers the development of Expert Systems in practical domains. Machine Learning is a promising solution to this problem and this paper discusses an *A*utomatic *K*nowledge *A*cquisition and *R*efinement *S*ystem (*AKARS-1*) for knowledge in rule form. Two major components of *AKARS-1*, the learning component (*AKA-1*) and the *He*uristic *R*efinement *S*ystem (*HERES*) are detailed. A prototype of *AKA-1* has been implemented and three case studies are used to validate this prototype. A comparison between *ID3* and *AKA-1* is presented. *HERES* employs heuristics and statistical methods to guide its search for refinements. These methods are also discussed in the paper.

1. Introduction

The knowledge acquisition problem of expert systems is a popular topic in Artificial Intelligence recently. Various knowledge acquisition systems have been developed. ***KLAUS*** [12], ***KRITON*** [7], ***MOLE*** [8], ***TIERESIAS*** [5] [6] and ***EMYCIN*** [3] are knowledge acquisition tools that assist in developing expert systems. Since these tools only solve the problem partially, better knowledge acquisition facilities are being studied.

This paper explores the possibility of developing an *A*utomatic *K*nowledge *A*cquisition and *R*efinement *S*ystem (***AKARS***). An ideal ***AKARS*** should have knowledge engineering ability to create knowledge bases in various representation schema. In order to perform most of the knowledge engineering work, it should also have learning ability, so that it can create knowledge bases in different situations. Refinement of existing knowledge bases for improvement and handling of domain evolution are important for a good knowledge acquisition system. Finally, it should be able to acquire knowledge from different knowledge sources (experts, training examples, raw data etc.).

However, an *A*utomatic *K*nowledge *A*cquisition and *R*efinement *S*ystem, AKARS-1, which concentrates on rule-based knowledge is presented in this paper. Section two gives the overall architecture of ***AKARS-1***. A prototype of the learning component of ***AKARS-1***, ***AKA-1***, is described in the third section. Section four details a generalization technique that simplifies rules induced by ***AKA-1***. The case studies performed and a comparison with ***ID3*** are presented in section five. The ***he***uristic ***re***finement ***s***ystem (***HERES***) of ***AKARS-1*** is discussed in section six followed by a conclusion.

2. The architecture of AKARS-1

AKARS-1 acquires knowledge by inducing knowledge in rule form from training examples. Initial knowledge bases acquired by ***AKARS-1*** are progressively refined to improve their performances through executing these knowledge bases. Figures 1 and 2 show the architecture and data flow of ***AKARS-1*** respectively.

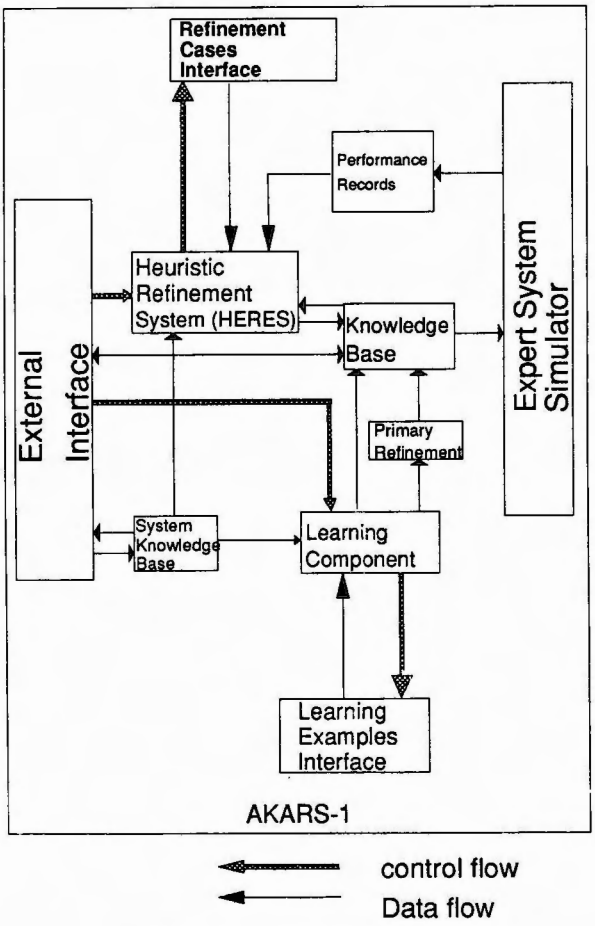


Figure 1: The Architecture of AKARS-1

AKARS-1 is an expert system itself on knowledge acquisition. The logical components of ***AKARS-1*** are as follows:

a) Learning examples interface : The learning examples interface retrieves examples stored in a database.

b) Refinement cases interface : The refinement cases interface accesses cases stored in a database.

c) Expert system simulator : This simulator executes rules in the knowledge base to gather performance statistics for the refinement of the knowledge base.

d) Knowledge base : The knowledge base stores rules induces by the learning component. Rules in the knowledge base will be progressively refined under the control of the refinement component

e) Learning component (*AKA-1*) : This component generates an initial knowledge base from a set of training examples (See section three).

f) Heuristic refinement system (*HERES*) : The initial knowledge base produced by the learning component component is usually imperfect. This component refines the knowledge base to increase its efficacy. Discussions on *HERES* appear in section six.

g) System knowledge base : *AKARS-1* is an expert system on knowledge acquisition. The system knowledge base stores the domain meta-knowledge, generalization rules, specialization heuristics and refinement techniques for *AKARS-1*.

h) External interface : It provides a user-friendly interface between *AKARS-1* and users (experts, knowledge engineer and *AKARS-1* developer). This interface provides two modes of operations: a normal mode and a supervisor mode. In the normal mode, experts and knowledge engineers interact with *AKARS-1* to 1) provide primary domain knowledge, 2) provide helps if *AKARS-1* meets problems that it cannot handle and 3) edit and/or retrieve rules in the knowledge base directly. In the supervisor mode, the *AKARS-1* developer can add and modify knowledge in the system knowledge base.

i) Performance records : They are the performance records of the current knowledge base. These records provide information to the refinement component about how to refine rules in the current knowledge base.

j) Primary refinement component : It employs *Dropping conditions rule* to simplify rules generated by the learning component. It is detailed in section four.

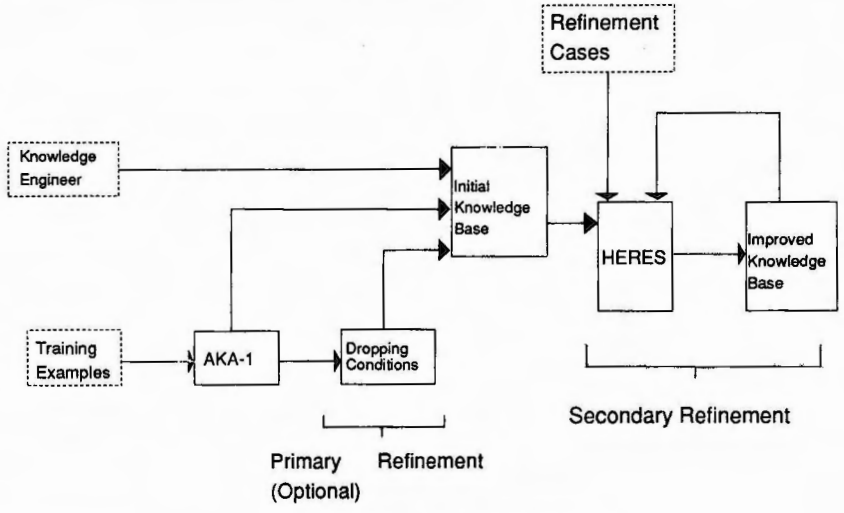


Figure 2: The data flow of AKARS-1

3. AKA-1

AKA-1 is a prototype of the learning component of ***AKARS-1*** which employs inductive learning strategy to induce production rules from training examples. The training examples are stored in a tabular format and production rules are represented in the following format:

IF
Degree-of-technology = old
THEN
Profits = low

The training examples are usually particularized by a number of attributes and their corresponding values. An attribute and its value raise a relation called an *attribute/value pair*. We represent an attribute/value pair by [*attribute*$_i$ *rel-op values*] where *attribute*$_i$ is the attribute being described by the pair, *rel-op* is a relational operator and *values* are the corresponding values of *attribute*$_i$ e.g. [attribute$_1$ = 1] and [attribute$_2$ > 3] are *attribute/value pairs*. A conjunction of *attribute/value pairs* generates a *description* that specifies characteristics of training examples that are being particularized.

3.1. The best attribute/value pair in AKA-1

Since examples are represented by attributes and their corresponding attribute values, it is necessary to evaluate the ability to discriminate examples for each attribute/value pair. In order to find out critical attribute/value pairs for constructing classification rules, every inductive learning system has its own criteria [25]. ***ID3*** [20] [21] is a famous inductive learning algorithm that employs *entropy* as bias of a learning system. However, entropy has the disadvantage of calculating only the overall discriminatory power of each attribute. Thus, entropy fails if an attribute has very good performances for some values but has poor results for others. Another disadvantage of ***ID3*** is its lack of generalization in the procedure of building decision trees for classification [13] [23] [24].

Generalization is an important characteristic of any inductive learning system. A good inductive system must be able to search through its description space to find out effective generalizations of the training examples. *Climbing generalization tree*, *exception excluding*, and *closing interval* are instances of generalization heuristics [15]. These generalization methods have been implemented in ***AKA-1***. In other words, ***AKA-1*** is able to produce less restricted descriptions to characterize training examples.

The criteria employed by ***AKA-1*** is conditional probability $Pr(Class_i|C_j)$. ***AKA-1*** considers all attribute/value pairs (include their generalizations) and estimates their discriminatory abilities by finding their conditional probabilities. The attribute/value pair that has the largest conditional probability is selected by ***AKA-1***. Cendrowska [4] has presented the advantages of using conditional probability as bias.

There are nominal, linear and structural attributes [15] in ***AKA-1*** as follows:

(a) ***Nominal attribute*** : The value set consists of independent symbols and no structure is assumed to relate the values in the domain.

(b) ***Linear attribute*** : The value set is an ordered set. Attributes measured on ordinal, interval, ratio and absolute scales are special cases of linear attribute.

(c) ***Structured attribute*** : The value set has a tree structure that reflects the generalization relation between the values.

Different methods are employed to evaluate the best linear pair, the best nominal pair and the best structural pair. Then, the best attribute/value pair is the one with the largest conditional probability.

3.2. Rule inducing algorithm of AKA-1

The algorithm of **AKA-1** is outlined as follows:

```
Let Examples : /* the set of training examples */
    E1, E2  : /* temporary variables store examples */
    ruleset : /* the set of production rules generated by AKA-1 */
    rule    : /* temporary variable that store rule */
/* There are m concept classes in the training set. These classes are Class1, Class2, Class3, ..... ,
Classm */
For Class-val := Class1 to Classm do
 begin
 ruleset := {};
 E1 := Examples;
 while there are elements in E1 that are instances of the class Class-val do
  begin
  E2 := E1;
  Rule := {};
  while E2 is not homogeneous do
   begin
   find the best attribute/value pair;
   C' := the best attribute/value pair;
   E2 := the instances in E2 that are covered by C';
   rule := rule & C';
   end;
  ruleset := ruleset + rule;
  E1 := E1 - { the set of instance covered by rule };
  end;
 end;
```

4. Generalizing production rules

Production rules induced by ***AKA-1*** can be further generalized in most situations. Generalization by Dropping conditions rule selects some conditions in a rule which are insignificant, then simplifies this rule by eliminating the condition(s). Because the generalized rule may be satisfied by some negative examples (i.e. the rule is inexact), the rules generated by this generalization operation may be inconsistent [15]. However, the certainty factor associates with the generalized rule estimates the accuracy of the rule.

4.1. Testing significance of condition

The key idea of *Dropping conditions rule* is to decide which conditions in a rule are insignificant in the classifying examples. Consider the rule:

IF
C_1 & C_2 & C_3 & ... & C_n
THEN
$CLASS_i$

To determine whether a condition, say C_i is significant, we first collect a population of examples that satisfy conditions C_1, C_2, C_3 ,... C_{i-1}, C_{i+1},... and C_n. This population can be divided into two parts, positive examples and negative examples. Each part can be further split into two: examples that satisfy C_i and examples that do not satisfy C_i. After this arrangement of the population, a 2x2 contingency table [1] is produced as follows:

	class variable		
condition variable	$CLASS_i$	not $CLASS_i$	*marginal*
satisfies C_i	n_{11}	n_{12}	n_{1+}
does not satisfy C_i	n_{21}	n_{22}	n_{2+}
marginal	n_{+1}	n_{+2}	n

n_{11} : the number of examples that satisfy C_i and belong to $Class_i$
n_{12} : the number of examples that satisfy C_i and do not belong to $Class_i$
n_{21} : the number of examples that do not satisfy C_i and belong to $Class_i$
n_{22} : the number of examples that do not satisfy C_i and do no belong to $Class_i$
n : the number of examples

Table 1: A contingency table

Checking significance of C_i is equivalent to the problem of testing the hypothesis that the two variables (condition and class variables) of the 2x2 contingency table are independent. Two well-known statistical methods, *Chi-square test* [1] or *Fisher-Irwin's exact test* [14] [16], can be applied according to the size of the number of examples in the hypothesis test using the corresponding contingency table as data.

4.2. Evaluating certainty factory

The certainty factor of a rule is estimated by using the formula:

$$\text{Certainty-Factor} = \frac{(\text{Pos-Examples} - 0.5)}{\text{Total-Examples}}$$

where **Pos-Examples** is the number of positive examples that satisfy the LHS and RHS of the rule. **Total-Examples** is the number of examples satisfying the conditions of the rule. The number 0.5 is the continuity correction for the binomial distribution [1].

4.3. The algorithm of dropping conditions

The generalization algorithm is

(1) Let **RULES** be the set of production rules generated by ***AKA-1***.

(2) Select a rule, **R**, from **RULES.**

(3) Assume **R** is

IF
C_1 & C_2 & C_3 & ... & C_n
THEN
$CLASS_i$

(4) Select a condition, C_j, and consider those examples that satisfy all other conditions in **R.** Establish a 2x2 contingency table.

(5) Determine whether condition C_j and $CLASS_i$ are independent in the examples. If C_j and $CLASS_i$ are independent, include C_j in **INDEP-SET**.

(6) Repeat (4) for other conditions in **R.**

(7) If **INDEP-SET** is not empty, select from **INDEP-SET** a condition C_k which has the smallest influence on the correctness of **R** if C_k is dropped from **R.** (i.e. probability of mis-classification is minimized.) Generate **R'** which is the same as **R** with the condition C_k missing. Then replaces **R** in **RULES** by **R'**.

(8) If **INDEP-SET** is empty, add **R** to **OUT-SET** in which all the processed rules are kept.

(9) Delete **R** from **RULES.**

(10) Repeat (1) to (9) until **RULES** is empty.

(11) Calculate certainty factor for each rule in **OUT-SET.**

5. Case studies

In order to demonstrate the efficacy of the prototype, three case studies have been explored.

5.1 The first case study

The first case concerns about risk classification of commercial bank loans [22]. This case classifies the risks into three categories that are represented by I, IA and II. The training examples and attributes are listed in tables 2 and 3 respectively. The structures of attributes (*Past account evaluation* and *Account type*) are depicted in figures 3 and 4 respectively.

	Class-I			Class-IA			Class-II		
Customer no. / attribute	1	2	3	4	5	6	7	8	9
F1	High	High	High	Average	High	Average	Average	Marginal	Average
F2	High	High	Average	Average	Average	Average	Marginal	Average	Average
Current assets	57	39	43	42	38	52	45	37	46
Net worth	57	55	49	37	46	40	38	29	36
Total debts	23	17	20	19	28	25	36	27	35
Funds	9	8	7	8	9	6	-9	7	5
Cash	4	3	5	6	4	5	6	6	5
Current liability	39	28	47	55	39	45	57	53	57
Inventory	21	15	8	12	14	1	7	13	14
Average inventory	9	14	11	6	6	5	3	5	6
Average profits	12	15	13	8	9	9	9	9	-1
Past account evaluation	one year	two years	three years	two years	one year	one year	three years	two years	absent
Customer status	current	current	new	current	current	new	new	current	current
Account type	commiss.	employ.	deposit	deposit	trust.	employ.	employ.	trust.	trust.

Table 2: The training examples of case one

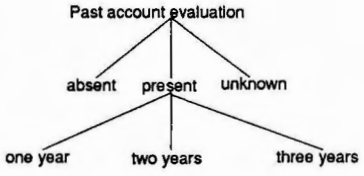


Figure 3: Structure of the attribute (Past account evaluation)

Attribute	Attribute type	Domain
F1	Nominal	{high, average, marginal, reject}
F2	Nominal	{high, average, marginal, reject}
Current assets	Linear	
Net worth	Linear	
Total debts	Linear	
Funds	Linear	
Cash	Linear	
Current liability	Linear	
Inventory	Linear	
Average inventory	Linear	
Average profits	Linear	
Past account evaluation	Structured	
Customer status	nominal	{current, new}
Account type	Structured	

Table 3: Types of attributes

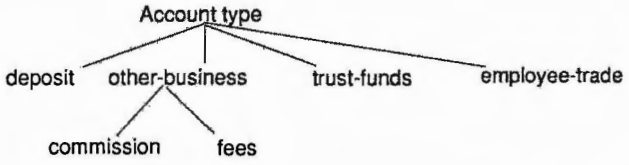


Figure 4: Structure of the attribute (Account type)

For this case study, **AKA-1** first induces rules for Class-I. **AKA-1** finds that the best nominal, linear and structural attribute/value pairs are [F2 = high], [Net-worth = 49..57] and [Account-type = commission] respectively because their conditional probabilities, Pr(Class-I|[F2 = high]), Pr(Class-I|[Net-worth = 49..57]) and Pr(Class-I|[Account-type = commission]), are all equal to 1. Since there are ties in conditional probabilities, **AKA-1** selects the pair that is satisfied by the largest number of positive training instances. The pair [Net-worth = 49..57] is chosen because it is satisfied by three positive instances.

Since [Net-worth = 49..57] is satisfied by all positive training examples and does not satisfied by any negative examples, the rule

If Net-worth = 49 to 57
Then Class is CLASS-I
With CF = 1.0

is induced for the concept CLASS-I. **AKA-1** iterates the above procedures to generated rules for CLASS-IA and CLASS-II. The initial rules induced by **AKA-1** are further generalized by using the generalization technique, *Dropping conditions of rule*. The initial and generalized rules are expressed in table 4. There are no gnerations in this case.

The initial rules	The generalized rules
If NET-WORTH = 49 to 57 Then Class is CLASSI With CF = 0.833	If NET-WORTH = 49 to 57 Then Class is CLASSI With CF = 0.833
If CURRENT-ASSETS = 38 Then Class is CLASSIA With CF = 0.500	If CURRENT-ASSETS = 38 Then Class is CLASSIA With CF = 0.500
If CURRENT-INVENTORY = 11 to 12 Then Class is CLASSIA With CF = 0.750	If CURRENT-INVENTORY = 11 to 12 Then Class is CLASSIA With CF = 0.750
If CUR-LIABILITY = 57 Then Class is CLASSII With CF = 0.750	If CUR-LIABILITY = 57 Then Class is CLASSII With CF = 0.750
If CUR-LIABILITY = 57 Then Class is CLASSII With CF = 0.750	If CUR-LIABILITY = 57 Then Class is CLASSII With CF = 0.750

Table 4: The initial and generalized rules of the first case study

5.2 The second case study

The second case study is taken from Ferrara's paper [9]. This case involves the problem of constructing some rules that assist people to invest on a new company in the computer field. The training examples are detailed records of some firms working in the field. These records report the profits of the firms for the previous year, together with other information regarding their features. Based on these records, *AKA-1* induces rules to characterize two concept classes, which are *low profits class* and *high profits class*. With these rules, an investor can anticipate profits of the company that he/she wants to invest and thus he/she can decide whether this is a good investment. The eleven training examples are listed in table 5 and the induced rules are further generalized. The rules are given in appendix A.

	Attribute			category
Company	Activity type	Competition	Degree-of-technology	Profits
$Comp_1$	software	no	old	low
$Comp_2$	software	yes	average	low
$Comp_3$	software	no	new	high
$Comp_4$	software	no	average	high
$Comp_5$	software	yes	new	high
$Comp_6$	software	yes	old	low
$Comp_7$	software	no	old	low
$Comp_8$	hardware	no	average	high
$Comp_9$	hardware	no	old	low
$Comp_{10}$	hardware	no	new	high
$Comp_{11}$	hardware	yes	average	low

Table 5: Training Examples of the second case study

5.3 The third case study

The third case study concerns the classification of patients with *duodendal ulcer* treated by *highly selective vagotomy (HSV)*. The problem is to learn rules to decide visick gradings of patients base on features of patients. The seventy seven training examples can be found in Pawlak's paper [17] and ***AKA-1*** produces 34 rules for these training examples. These rules are depicted in appendix B.

5.4. Comparison with ID3

A simple inductive learning program using ***ID3*** algorithm [20] [21] is developed for comparison. The second and the third case studies have been involved in the comparison. The advantages of ***AKA-1*** over ***ID3*** are listed as follows:

(A) ***AKA-1*** can handle nominal, linear and structural attributes while ***ID3*** can only process nominal attributes.

(B) ***AKA-1*** performs many generalizations, thus rules generated by it are usually simpler and more concise. On the other hand, since ***ID3*** does not perform sophisticated generalizations, rules produced are typically more complicated.

(C) ***AKA-1*** evaluates the discriminatory ability of every possible attribute/value pair to determine which pairs are used to form rules, thus it generates more precise and concise rules. On the other hand, ***ID3*** calculates the overall discriminatory power of each attribute to discover the crucial attributes. However, this evaluation method may overlook a good attribute/value pair if the overall discriminatory ability of the attribute is poor.

(D) The number of rules generated by ***AKA-1*** is usually smaller than that produced by ***ID3***. Consider the third case study, ***AKA-1*** induces 34 rules (satisfied by 77 training examples) while ***ID3*** generates 36 rules (satisfied by only 72 examples).

(E) ***AKA-1*** can tolerate the problem arised from contradictive training examples (e.g. examples 13 and 55 of the third case study). On the other hand, ***ID3*** will terminate abnormally if there are contradictive examples.

(G) ***AKA-1*** is at least as accurate as ***ID3***. The rules generated by ***AKA-1*** are complete and consistent for the first and the second case studies. If we ignore the two contradictive training examples of the third case (examples 13 and 55), the rules produced by *AKA-1* are also complete and consistent.

6. *He*uristic *re*finement *s*ystem (*HERES*)

A good knowledge base should diagnose refinement examples with small number of mistakes, i.e. misdiagnosed examples. Thus the number of mistakes is an important indicator of the quality of a knowledge base. Therefore, a knowledge base refinement system should search for good modifications (addition, deletion or alteration) of part of the existing rules, to decrease the number of mistakes and improve the overall behaviour of the knowledge base.

To prevent the combinatorial explosion of this refinement problem, heuristics is employed to guide the search of good modifications and this is why a heuristic approach is used in ***AKARS-1***. ***HERES*** progressively refines an initial knowledge base into a high performance one. Ginsbery, Weiss and Politakis have explored the feasibility of a heuristic approach for knowledge base refinement [10] [11] [18].

HERES integrates statistical performance information of a knowledge base and some domain independent knowledge engineering concepts (heuristics) to recommend good refinements for a knowledge base. ***HERES*** requires an initial knowledge base and some refinement cases for its operations. The expert system simulator of ***AKARS-1*** solves refinement cases by using rules in an initial knowledge base and ***HERES*** then compares the conclusions reached by the expert system simulator with the conclusions indicated by domain experts (these conclusions are stored in the refinement cases) and creates statistics on the performance of the current knowledge base. ***HERES*** performs a refinement by using heuristics. These heuristics determine whether a generalization or a specialization should be invoked.

HERES suggests a generalization of a rule if its expected performance gain is higher than that for specialization. Different heuristics exist to recommend plausible generalizations. This generalization techniques used are 1) *Dropping conditions*, 2) *Extending range of linear attribute*, 3) *Climbing generalization tree* and 4) *Generating complementary rules*.

HERES suggests a specialization of a rule if the expected performance gain for specialization is higher than that for generalization. Some heuristics for specialization are as follows:

(A) If a linear attribute is the best candidate for specialization, a more restricted range should be calculated for the linear attribute.

(B) If a structural attribute is the best candidate for specialization, then a less general value should be found for this attribute.

(C) If a rule should be specialized, then add a new condition to the antecedent part of this rule.

7. Conclusion

The architecture of a novel ***A***utomatic ***K***nowledge ***A***cquisition and ***R***efinement ***S***ystem (***AKARS-1***) has been presented. The learning component (***AKA-1***) has been implemented on a ***IBM/PC*** in ***Golden Common Lisp***. Comparing with ***ID3*** algorithm, the novel ***AKA-1*** is more powerful because it can handle nominal, linear and structural attributes. Furthermore, ***AKA-1*** has many generalization heuristics to create abstract rules for classification. Some empirical case studies have been performed, and the results have demonstrated the power of ***AKA-1*** which produces less but more accurate rules than ***ID3***.

Two statistical methods (Chi-square test and Fisher-Irwin exact test) are employed in simplifying the rules induced by ***AKA-1***, and these mechanisms can be viewed as the primary refinement stage of ***AKARS-1***.

The operation of ***He***uristics ***Re***finement ***S***ystem (***HERES***) is also outlined in this paper. ***HERES*** collects and evaluates performance statistics on a knowledge base. Based on these statistics and various refinement heuristics, ***HERES*** can perform plausible knowledge base refinements. This heuristic approach on knowledge base refinement is effective and flexible, because the system can be extended and/or modified easily by mutating or adding refinement heuristics.

Reference

[1] Bhattacharyya, G. K. and Johnson, R. A. (1977). *Statistical Concepts and Methods.* (Wiley).

[2] Buchanan, B. G. and Mitchell, T. M. (1978). Model-Directed Learning of Production Rules, in Waterman, D. A. and Hayes-Roth, Federick. (eds), *attern-Direct Inference Systems*, pp297-312.

[3] Buchanan, B. G. and Shortliffe, E. H. (eds), (1984). *Rule-Based Expert Systems The MYCIN Experiments of the Stanford Heuristic Programming Project* (Addison-Wesley).

[4] Cendrowska, Jadzia (1987). PRISM: An algorithm for inducing modular rules, in *Int. J. Man-Machine Studies* **27**, pp349-370.

[5] Davis, Randall. (1978). Knowledge Acquisition in Rule-Based Systems-Knowledge about Representation as a Basis for System Construction and Maintenance, in Waterman, D. A. and Hayes-Roth, Federick. (eds), *Pattern-Direct Inference Systems*, pp99-134.

[6] Davis, Randall. (1982). Teiresias: Applications of Meta-level Knowledge, in Heiberg, Diane D. and Damstra, David A. (eds), *Knowledge-Based Systems in Artificial Intelligence*, pp227-490.

[7] Diederich, J., Ruhmann, I. and May, M. (1987). KRITON: a knowledge-acquisition tool for expert systems, in *Int. J. Man-Machine Studies* **26**, pp29-40.

[8] Eshelman, L., Ehret D., McDermott, J. and Tan, M. (1987). MOLE: a tenacious knowledge-acquisition tool in *Int. J. Man-Machine Studies* **26**, pp41-54.

[9] Ferrara, Fabrizio Massimo (1988). SOCRATES A System to Extract Rules and Knowledge from Existing Databases, in *SIGART Newsletter*, April 1988, Number 104, pp21-25.

[10] Ginsberg, A. (1986). A Meta-linguistic Approach to the Construction of Knowledge Base Refinement Systems, in *Proceedings AAAI-86*, Philadelphia, PA, pp436-441.

[11] Ginsberg, A., Weiss, Sholom M. and Politakis P. (1988). Automatic Knowledge Base Refinement for Classification Systems, in *Artificial Intelligence* **35** pp227-241.

[12] Hass, N., Hendrix, G. G. (1983). Learning by being told: Acquiring knowledge for information management, in Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (eds), *MACHINE LEARNING: An Artificial Intelligence Approach I*, pp405-427.

[13] Jackson, A. H. (1988). Machine Learining, in *Expert Systems*, May 1988. Vol. **5**, No. 2. pp132-150

[14] Lindgren, B. W. (1976). *Statistical Theory.* Third Edition (Macmillan).

[15] Michalski R. S. (1983). A Theory and Methodology of inductive Learning, in Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (eds), *MACHINE LEARNING: An Artificial Intelligence Approach I*, pp83-134.

[16] Ostle, B. and Mensing, R. W. (1975). *Statistics in Research.* Third Edition (The Iowa State University Press).

[17] Pawlak, Z., Slowinski, K. and Slowinski, R. (1986). Rough classification of patients after highly selective vagotomy for duodendal ulcer, *Int. J. Man-Machine Studies*, **24**, pp.413-433.

[18] Politakis, Peter and Weiss, S. M. (1984). Using Empirical Analysis to Refine Expert System Knowledge Bases, in *ARTIFICIAL INTELLIGENCE* **22**, pp23-48.

[19] Politakis, Peter (1985). *Empirical Analysis for Expert Systems* (Pitman).

[20] Quinlan, J. Ross (1983). Learning Efficient Classification Procedures and their Application to Chess End Games, in Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (eds), *MACHINE LEARNING: An Artificial Intelligence Approach I*, pp463-482.

[21] Quinlan, J. Ross (1986). The Effect of Noise on Concept Learning, in Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (eds), *MACHINE LEARNING: An Artificial Intelligence Approach II*, pp149-166.

[22] Shaw, Michael J. (1987). Applying Inductive Learning to Enhance Knowledge-based Expert Systems, in *Decision Support Systems* **3** pp319-332.

[23] Thompson, Bill and Thompson, Bev (1986). Finding Rules in Data, in *BYTE*, NOVERBER 1986, pp149-158.

[24] Tschudi, Finn (1988). MATRIX REPRESENTATION OF EXPERT SYSTEMS, in *AI EXPERT*, OCTOBER 1988, pp44-53.

[25] Utgoff, P. E. (1986). Shift of Bias for Inductive Concept Learning, in in Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (eds), *MACHINE LEARNING: An Artificial Intelligence Approach II*, pp107-148.

Appendix A : Rules induced in the second case study

Rules induced by **AKA-1**	Rules generalized
If DEGREE-OF-TECHNOLOGY = OLD Then Profits is LOW With CF = 0.875	If DEGREE-OF-TECHNOLOGY = OLD Then Profits is LOW With CF = 0.875
If COMPETITION = YES and DEGREE-OF-TECHNOLOGY = AVERAGE Then Profits is LOW With CF = 0.75	If COMPETITION = YES Then Profits is LOW With CF = 0.625
If COMPETITION = NO and DEGREE-OF-TECHNOLOGY = AVERAGE Then Profits is HIGH With CF = 0.75	If COMPETITION = NO Then Class = HIGH With CF = 0.50
If DEGREE-OF-TECHNOLOGY = NEW Then Profits is HIGH With CF = 0.83	If DEGREE-OF-TECHNOLOGY = NEW Then Class = HIGH With CF = 0.83

Appendix B : Rules induced in the third case study

Rules induced by **AKA-1**	Rules generalized
If BASIC-ACID-OUTPUT = 1 Then Class is 1 With CF = 87.5 %	If BASIC-ACID-OUTPUT = 1 Then Class is 1 With CF = 87.5 %
If COMPLICATIONS-OF-ULCER = 1 AGE = 0 Then Class is 1 With CF = 94.4444 %	If COMPLICATIONS-OF-ULCER = 1 AGE = 0 Then Class is 1 With CF = 94.4444 %
If MAXIMAL-ACID-OUTPUT = 2 COMPLICATIONS-OF-ULCER = 0 Then Class is 1 With CF = 93.75 %	If MAXIMAL-ACID-OUTPUT = 2 COMPLICATIONS-OF-ULCER = 0 Then Class is 1 With CF = 93.75 %
If MAXIMAL-ACID-OUTPUT = 3 AGE = 1 Then Class is 1 With CF = 92.8571 %	If MAXIMAL-ACID-OUTPUT = 3 AGE = 1 Then Class is 1 With CF = 92.8571 %
If HCI-CONCENTRATION-1 = 0 DURATION-OF-THE-DISEASE = 2 Then Class is 1 With CF = 87.5 %	If HCI-CONCENTRATION-1 = 0 DURATION-OF-THE-DISEASE = 2 Then Class is 1 With CF = 87.5 %
If COMPLICATIONS-OF-ULCER = 1 MAXIMAL-ACID-OUTPUT = 2 Then Class is 1 With CF = 83.3333 %	If COMPLICATIONS-OF-ULCER = 1 MAXIMAL-ACID-OUTPUT = 2 Then Class is 1 With CF = 83.3333 %
If VOL-OF-GASTRIC-JUICE-1 = 0 HCI-CONCENTRATION-1 = 1 Then Class is 1 With CF = 90.0 %	If VOL-OF-GASTRIC-JUICE-1 = 0 HCI-CONCENTRATION-1 = 1 Then Class is 1 With CF = 90.0 %
If MAXIMAL-ACID-OUTPUT = 3 COMPLICATIONS-OF-ULCER = 3 Then Class is 1 With CF = 75.0 %	If MAXIMAL-ACID-OUTPUT = 3 Then Class is 1 With CF = 79.4118 %
If COMPLICATIONS-OF-ULCER = 0 HCI-CONCENTRATION-2 = 1 Then Class is 1 With CF = 95.0 %	If COMPLICATIONS-OF-ULCER = 0 HCI-CONCENTRATION-2 = 1 Then Class is 1 With CF = 95.0 %
If DURATION-OF-THE-DISEASE = 2 VOL-OF-RESIDUAL-GASTRIC-JUICE = 1 Then Class is 1 With CF = 92.8571 %	If DURATION-OF-THE-DISEASE = 2 VOL-OF-RESIDUAL-GASTRIC-JUICE = 1 Then Class is 1 With CF = 92.8571 %
If DURATION-OF-THE-DISEASE = 2 COMPLICATIONS-OF-ULCER = 1 Then Class is 1 With CF = 91.6667 %	If DURATION-OF-THE-DISEASE = 2 COMPLICATIONS-OF-ULCER = 1 Then Class is 1 With CF = 91.6667 %

Rules induced by AKA-1	Rules generalized
If DURATION-OF-THE-DISEASE = 2 MAXIMAL-ACID-OUTPUT = 0 HCI-CONCENTRATION-2 = 2 Then Class is 1 With CF = 83.3333 %	If DURATION-OF-THE-DISEASE = 2 MAXIMAL-ACID-OUTPUT = 0 HCI-CONCENTRATION-2 = 2 Then Class is 1 With CF = 83.3333 %
If COMPLICATIONS-OF-ULCER = 4 HCI-CONCENTRATION-1 = 1 Then Class is 1 With CF = 50.0 %	If COMPLICATIONS-OF-ULCER = 4 HCI-CONCENTRATION-1 = 1 Then Class is 1 With CF = 50.0 %
If COMPLICATIONS-OF-ULCER = 0 BASIC-ACID-OUTPUT = 0 Then Class is 1 With CF = 91.6667 %	If COMPLICATIONS-OF-ULCER = 0 BASIC-ACID-OUTPUT = 0 Then Class is 1 With CF = 91.6667 %
If COMPLICATIONS-OF-ULCER = 0 VOL-OF-GASTRIC-JUICE-1 = 1 SEX = F Then Class is 1 With CF = 96.6667 %	If COMPLICATIONS-OF-ULCER = 0 VOL-OF-GASTRIC-JUICE-1 = 1 SEX = F Then Class is 1 With CF = 96.6667 %
If MAXIMAL-ACID-OUTPUT = 3 VOL-OF-RESIDUAL-GASTRIC-JUICE = 0 VOL-OF-GASTRIC-JUICE-2 = 1 Then Class is 1 With CF = 91.6667 %	If MAXIMAL-ACID-OUTPUT = 3 VOL-OF-RESIDUAL-GASTRIC-JUICE = 0 VOL-OF-GASTRIC-JUICE-2 = 1 Then Class is 1 With CF = 91.6667 %
If VOL-OF-GASTRIC-JUICE-1 = 0 SEX = F BASIC-ACID-OUTPUT = 2 Then Class is 1 With CF = 92.8571 %	If VOL-OF-GASTRIC-JUICE-1 = 0 SEX = F BASIC-ACID-OUTPUT = 2 Then Class is 1 With CF = 92.8571 %
If COMPLICATIONS-OF-ULCER = 3 DURATION-OF-THE-DISEASE = 2 SEX = F AGE = 1 HCI-CONCENTRATION-1 = 2 VOL-OF-GASTRIC-JUICE-1 = 1 VOL-OF-RESIDUAL-GASTRIC-JUICE = 0 BASIC-ACID-OUTPUT = 2 HCI-CONCENTRATION-2 = 1 VOL-OF-GASTRIC-JUICE-2 = 1 MAXIMAL-ACID-OUTPUT = 0 Then Class is 1 With CF = 25.0 %	If DURATION-OF-THE-DISEASE = 2 SEX = F AGE = 1 HCI-CONCENTRATION-1 = 2 VOL-OF-GASTRIC-JUICE-1 = 1 VOL-OF-RESIDUAL-GASTRIC-JUICE = 0 BASIC-ACID-OUTPUT = 2 HCI-CONCENTRATION-2 = 1 VOL-OF-GASTRIC-JUICE-2 = 1 MAXIMAL-ACID-OUTPUT = 0 Then Class is 1 With CF = 25.0 %
If COMPLICATIONS-OF-ULCER = 2 DURATION-OF-THE-DISEASE = 1 Then Class is 2 With CF = 75.0 %	If DURATION-OF-THE-DISEASE = 1 Then Class is 2 With CF = 8.33333 %
If COMPLICATIONS-OF-ULCER = 2 HCI-CONCENTRATION-2 = 1 Then Class is 2 With CF = 75.0 %	If HCI-CONCENTRATION-2 = 1 Then Class is 2 With CF = 11.3636 %
If COMPLICATIONS-OF-ULCER = 4 VOL-OF-GASTRIC-JUICE-1 = 0 Then Class is 2 With CF = 50.0 %	If COMPLICATIONS-OF-ULCER = 4 VOL-OF-GASTRIC-JUICE-1 = 0 Then Class is 2 With CF = 50.0 %
If COMPLICATIONS-OF-ULCER = 2 HCI-CONCENTRATION-1 = 1 SEX = F Then Class is 2 With CF = 75.0 %	If HCI-CONCENTRATION-1 = 1 SEX = F Then Class is 2 With CF = 26.9231 %
If COMPLICATIONS-OF-ULCER = 3 SEX = M Then Class is 2 With CF = 50.0 %	If COMPLICATIONS-OF-ULCER = 3 SEX = M Then Class is 2 With CF = 50.0 %
If DURATION-OF-THE-DISEASE = 0 AGE = 1 Then Class is 2 With CF = 50.0 %	If DURATION-OF-THE-DISEASE = 0 AGE = 1 Then Class is 2 With CF = 50.0 %

Rules induced by AKA-1	Rules generalized
If MAXIMAL-ACID-OUTPUT = 1 HCI-CONCENTRATION-1 = 1 COMPLICATIONS-OF-ULCER = 0 Then Class is 2 With CF = 50.0 %	If MAXIMAL-ACID-OUTPUT = 1 HCI-CONCENTRATION-1 = 1 Then Class is 2 With CF = 25.0 %
If COMPLICATIONS-OF-ULCER = 3 VOL-OF-GASTRIC-JUICE-1 = 1 AGE = 1 SEX = F DURATION-OF-THE-DISEASE = 2 HCI-CONCENTRATION-1 = 2 VOL-OF-RESIDUAL-GASTRIC-JUICE = 0 BASIC-ACID-OUTPUT = 2 HCI-CONCENTRATION-2 = 1 VOL-OF-GASTRIC-JUICE-2 = 1 MAXIMAL-ACID-OUTPUT = 0 Then Class is 2 With CF = 25.0 %	If COMPLICATIONS-OF-ULCER = 3 AGE = 1 SEX = F DURATION-OF-THE-DISEASE = 2 HCI-CONCENTRATION-1 = 2 VOL-OF-RESIDUAL-GASTRIC-JUICE = 0 BASIC-ACID-OUTPUT = 2 HCI-CONCENTRATION-2 = 1 VOL-OF-GASTRIC-JUICE-2 = 1 MAXIMAL-ACID-OUTPUT = 0 Then Class is 2 With CF = 25.0 %
If HCI-CONCENTRATION-1 = 1 VOL-OF-GASTRIC-JUICE-1 = 2 DURATION-OF-THE-DISEASE = 2 VOL-OF-RESIDUAL-GASTRIC-JUICE = 0 Then Class is 2 With CF = 75.0 %	If HCI-CONCENTRATION-1 = 1 VOL-OF-GASTRIC-JUICE-1 = 2 DURATION-OF-THE-DISEASE = 2 VOL-OF-RESIDUAL-GASTRIC-JUICE = 0 Then Class is 2 With CF = 75.0 %
If DURATION-OF-THE-DISEASE = 0 SEX = M Then Class is 3 With CF = 75.0 %	If SEX = M Then Class is 3 With CF = 9.375 %
If COMPLICATIONS-OF-ULCER = 3 HCI-CONCENTRATION-1 = 0 Then Class is 3 With CF = 50.0 %	If HCI-CONCENTRATION-1 = 0 Then Class is 3 With CF = 8.33333 %
If VOL-OF-RESIDUAL-GASTRIC-JUICE = 2 DURATION-OF-THE-DISEASE = 1 Then Class is 3 With CF = 50.0 %	If VOL-OF-RESIDUAL-GASTRIC-JUICE = 2 DURATION-OF-THE-DISEASE = 1 Then Class is 3 With CF = 50.0 %
If VOL-OF-GASTRIC-JUICE-2 = 2 VOL-OF-RESIDUAL-GASTRIC-JUICE = 1 SEX = F Then Class is 4 With CF = 50.0 %	If VOL-OF-RESIDUAL-GASTRIC-JUICE = 1 SEX = F Then Class is 4 With CF = 5.55555 %
If VOL-OF-GASTRIC-JUICE-2 = 2 VOL-OF-GASTRIC-JUICE-1 = 2 HCI-CONCENTRATION-2 = 2 COMPLICATIONS-OF-ULCER = 0 Then Class is 4 With CF = 75.0 %	If HCI-CONCENTRATION-2 = 2 COMPLICATIONS-OF-ULCER = 0 Then Class is 4 With CF = 7.89474 %
If VOL-OF-GASTRIC-JUICE-2 = 0 HCI-CONCENTRATION-2 = 0 COMPLICATIONS-OF-ULCER = 0 Then Class is 4 With CF = 50.0 %	If VOL-OF-GASTRIC-JUICE-2 = 0 HCI-CONCENTRATION-2 = 0 COMPLICATIONS-OF-ULCER = 0 Then Class is 4 With CF = 50.0 %
If VOL-OF-GASTRIC-JUICE-1 = 2 DURATION-OF-THE-DISEASE = 1 HCI-CONCENTRATION-2 = 0 COMPLICATIONS-OF-ULCER = 0 Then Class is 4 With CF = 50.0 %	If DURATION-OF-THE-DISEASE = 1 HCI-CONCENTRATION-2 = 0 COMPLICATIONS-OF-ULCER = 0 Then Class is 4 With CF = 16.6667 %

A KNOWLEDGE ACQUISITION SYSTEM THAT AIMS AT INTEGRATING INDUCTIVE LEARNING AND EXPLANATION-BASED REASONING

Katsuhiko Tsujino
Morikazu Takegaki
Shogo Nishida

Department of Information and System Science
Central Research Laboratory
MITSUBISHI Electric Corporation
Amagasaki Hyogo 661 JAPAN

This paper describes the design philosophy and current issues of a knowledge acquisition system named KAISER. This system aims at integrating similarity-based learning and explanation-based learning by guiding inductive learning with theoretical and heuristic domain knowledge interactively. It first learns classification knowledge inductively then analyzes the result and process based on domain knowledge. After that, it asks sophisticated questions for acquiring new knowledge according to the analysis. The queries stimulate a human expert and help him to revise the learned results, control the learning process and prepare new examples and domain knowledge. This interactive induce-evaluate-ask cycle provides rational interview to accelerate incremental acquisition of formalized domain knowledge as well as efficient induction of operational and reasonable knowledge proved by the domain knowledge.

1. INTRODUCTION

Knowledge acquisition and learning are inseparable functions of intelligent systems. The learner must notice improper state of learning and the results that suggest the lack of knowledge or an error in order to demand new knowledge to resolve this state. He must also examine the acquired knowledge and organize it into his knowledge through learning it. Such a cycle achieves intelligent and efficient knowledge acquisition.

This paper describes a knowledge acquisition system named KAISER. As the acronym implies, KAISER (a Knowledge Acquisition Inductive System promoted by Explanation-based Reasoning) inductively learns classification knowledge in the form of a decision tree and analyzes results and processes with domain and task specific knowledge to detect improper conditions and mismatching between the theory and induction results. Then it asks suggestive questions to eliminate the improprieties and acquires new domain knowledge and various information for the next induction cycle.

2. BASIC IDEA

Inductive learning is one of the most powerful techniques for constructing knowledge-based systems. It enables full (or semi) automatic formalization of knowledge base. In other words, if we specify proper knowledge representation, hypotheses, conclusions and enough examples (i.e., define a proper inductive bias [1]), we don't have to make efforts to determine the details of the knowledge. Quinlan proposed an efficient induction algorithm named ID3 [3] to generate a classification knowledge in the form of a decision tree out of some examples represented in OAV-form. Although it provides quite reasonable knowledge statistically, the learned trees are not yet developed enough from the expert's point of view. That's why the system knows nothing about the target domain.

When a human expert inspects the induced result, he can easily point out some 'improper' conditions in it. Let's call them improprieties. An impropriety is not an error nor a fault statistically but something that seems strange to an expert. By interpreting the reason why the improprieties arise, the expert can recommend new examples and new constraints according to his domain knowledge. Nevertheless, such knowledge is difficult to acquire beforehand by top-down reflection. This fact suggests that the inductive learning can be an efficient mental stimulus for knowledge acquisition as well as a powerful technology to get reasonable classifiers automatically. This is the basic idea of KAISER.

The fundamental issues of KAISER include: (1) What does the expert think improper in the result?; (2) Why does he feel so? What kind of knowledge is it based on?; and (3) How does he override the induction? What kinds of choices are there? By investigating these issues, we can derive the interview strategy that interprets the induction process and associates it with interviewing. The first issue is indispensable to clear up the features of impropriety. The second issue is necessary to develop an interpretation and obtain an explanation of the impropriety. The third issue makes it possible to provide considerate choices to resolve the impropriety.

3. RELATED WORKS

There are three aspects of KAISER: (1) an explanation-based [4] knowledge refinement system; (2) an inductive learning system utilizing domain knowledge to shift inductive bias; and (3) a role-limiting [5] intelligent interview system. This section discusses KAISER from these points of view.

3.1 KAISER in terms of learning

In the last few years, EBL [4] has become one of the hot issues of machine learning. It provides a new inference technique to generate operational knowledge by generalizing the explanation process of why a particular example can be an instance of the target concept. Although it can generate reliable and efficient chunk of knowledge based on the domain theory, it's not concerned with acquisition and formalization of the theory, which seems quite difficult. Furthermore, the knowledge may be a correct but useless generalization of a special and far-fetched explanation among many possible ones, because the explanation process is not under any (e.g., statistic) control. On the one hand, traditional inductive learning methods, say SBL methods against EBL, are able to induce generalized knowledge without any domain theory. Although it's a very fascinating idea that we can get a complete knowledge base automatically when we specify enough examples, we actually have to examine the learning algorithm and the features for examples (i.e., inductive bias) based on domain knowledge to learn 'good' concepts from noisy and partial examples.

There are two approaches to cope with these problems. One is controlling explanation by inductive inference, and the other is guiding induction by domain knowledge. A number of proposals [6,7,8] have been made based on the former approach. For instance, Lebowitz developed a concept learning system named UNIMEM [6] which stands for the former idea, while KAISER stands for the latter. Although both approaches aim at the same goal of integrating SBL and EBL, there is an essential difference in their tasks and demands. As UNIMEM aims at concept clustering, that is finding out some sets of examples corresponding to some reasonable concepts, it tries to understand as many examples as possible, but it doesn't have to cover all of the examples. So understanding the explanation of the examples is more important for UNIMEM than their classification. This peculiarity makes UNIMEM take the former approach. On the other hand, KAISER aims at defining a good classifier that satisfies all of the examples if possible, but it doesn't have to understand the classifier completely. This makes KAISER take the latter approach for a more inductive facility.

3.2 KAISER in terms of knowledge acquisition

Another aspect of KAISER is role-limiting, a model-based knowledge acquisition system. A system of this type conducts top-down interview to acquire new knowledge by analyzing known knowledge with task-specific strategic knowledge. MORE [9] and MOLE [10] point out some causal defects in diagnostic knowledge and get missing knowledge according to several interviewing strategies. For example, differentiation strategy tries

to get a new distinctive symptom when many hypotheses can be expected from a symptom. SALT [11] adopts propose-and-revise strategy to ask and resolve interdependent constraints among many parameters by instantiating them one by one. For example, when no reasonable solution can be found, it asks an expert proper inference step to backtrack to. ETS [12] and its successor AQUINAS [13] employ personal construct theory [14] to generate classification knowledge by interviewing an operator. It tries to get distinctive features from the operator by giving him three examples and asking an attribute that distinguishes one of the examples from the other two. It also asks the ratings of each attribute for all examples and derives classification knowledge by analyzing the statistics among the ratings. Among these systems, AQUINAS will be one of the new-type systems that provides high inductive faculty, whereas most of traditional systems propose less knowledge by themselves and aim at pointing out missing knowledge based on their models. From this point of view, KAISER is the same type of system as AQUINAS. They induce classification knowledge and ask missing information by analyzing the results. Furthermore, while AQUINAS evaluates the ratings with only heuristic strategies derived from statistic knowledge, KAISER utilizes domain knowledge as well to supplement the inductive power of the statistical learner. This approach allows KAISER a more persuasive interview based on domain knowledge.

4. ARCHITECTURE OF KAISER

Figure 1 illustrates a conceptual block diagram and the knowledge acquisition cycle of KAISER. KAISER consists of four sub-modules and three databases. The inductive learner adopts ID3 to generate a decision tree from the examples stored in the example database. The impropriety detector evaluates the decision tree based on the knowledge stored in domain knowledge base and impropriety knowledge base to detect the

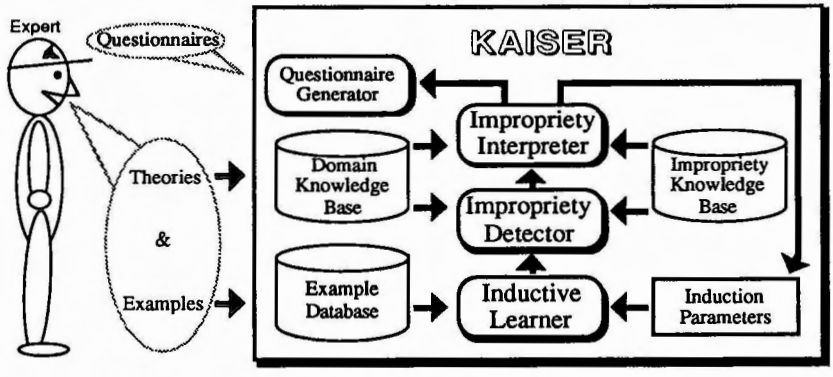
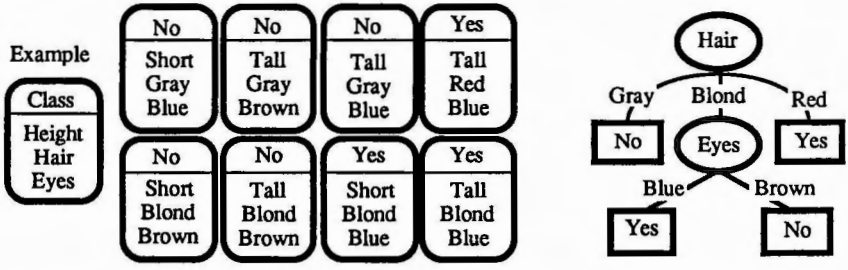


Figure 1. Block diagram of KAISER.

improprieties such as unreliable conditions in the tree and mismatching between the decision tree and domain knowledge. The domain knowledge base contains abstract causal knowledge that the target knowledge should satisfy. This knowledge base can be initially empty or specified by a human expert to the extent that he can recall. The impropriety knowledge base contains knowledge about the improprieties that are categorized into two types, structural improprieties that require no domain knowledge and semantic improprieties that refer to the domain knowledge. This knowledge can be prescribed beforehand, since neither of them depend on any domain knowledge although the latter refers to the knowledge. Even if the domain knowledge base is empty, KAISER can acquire both examples and domain knowledge by applying the domain-independent structural knowledge in the impropriety knowledge base. The impropriety interpreter analyzes the detected improprieties and tries to suggest the reason why they arise and the way to eliminate them. The interpretation results are converted into some queries and choices by the questionnaire generator. This questionnaire gives the human expert some mental stimulus that helps him recall missing examples, correct domain knowledge and set proper induction parameters.

4.1 Induction of decision trees

As mentioned above, KAISER adopts Quinlan's ID3 algorithm for its inductive learner. ID3 accepts examples represented in OAV-form and generates classification knowledge in the form of a decision tree as shown in Figure 2. Figure 2(a) shows eight examples represented by three attributes 'Height', 'Hair' and 'Eyes'. The upper right example, for instance, means that a boy likes a girl whose height is tall, hair is red and eyes are blue. Figure 2(b) illustrates the induced condition about his type of girls. It means that the boy likes a girl whose hair is blond and eyes are blue. As this tree does not refer to the height of the girl, we can say that it will be a generalized condition about the type of the boy.



(a) Examples represented in OAV-form. (b) A decision tree induced from the examples.

Figure 2. Input and output structures of ID3.

Although ID3 provides simple and powerful methods to learn fairly excellent classifiers without any domain knowledge, it has several inductive problems including: (1) it sometimes generates ignorant leaves that don't specify any class when we use multiple (not binary) attributes; (2) the criterion for attribute selection has a strong tendency to over-estimate detailed attributes and split the examples into too many sub-examples at once, which makes ID3 less resistant to noisy examples; and (3) it can not deal with numeric values. Quinlan [3] and Tsujino [2] proposed some approaches to these problems using statistic and heuristic approaches respectively. KAISER thinks of these problems as task-specific structural improprieties and copes with them by heuristic detection and modification in a manner similar to Tsujino's approach. This heuristic approach enables KAISER to use various kinds of knowledge to guide ID3.

4.2 Structural improprieties

Improprieties of this type are task-specific and domain-independent, that means, they are not dependent on the target knowledge to be acquired but peculiar to the classification task or employed learning algorithm. They are detected as some generalization failures or some unreliable conditions, e.g., affected by the noisy examples. These improprieties give many kinds of clues for reasonable generalization according to the common knowledge about the classification task. To make the matter clear, let's think of the following two examples: (1) When the price of a T-shirt of size LL and M is $30 but that of S is $28, how much do you pay for an L-size shirt?; and (2) When the price of a T-shirt is $32 while that of others on the same bargain rack is $30, do you pay $32 for that shirt without doubt? Most of us will feel a kind of impropriety for these situations. The former example illustrates such impropriety of missing information, and the latter of noisy examples. This sort of knowledge is common in all classification tasks. Currently, KAISER provides the following five types of structural improprieties that have their own peculiar elimination methods.

- *Noisy node impropriety* A node has a few examples, e.g., less than twice of the average leaf size. This is a typical clue for pruning. It recommends to change the node to a leaf of its major class if there is no other impropriety at the node.
- *NIL node impropriety* The class of a leaf can't be determined by induction because of the lack of examples. This often arises when we use multiple valued (not binary) attributes. The native elimination method of this impropriety is to ask the correct class and/or to get new examples that will fit the leaf.

- *Inseparable example impropriety* Some examples can not be separated only with given attributes. This impropriety possesses both features of NIL node and noisy node improprieties.
- *Similar node impropriety* Two brother nodes refer a same attribute, their entropies are near, and they have similar component classes. This is a structural clue to generalize the attribute of their father by merging the links to them. It recommends to reconstruct an inferior tree by merging the examples in the nodes.
- *Similar class impropriety* More than one nodes try to separate same set of class at another places in a decision tree. This is a clue for a new attribute. It recommends to induce a new decision tree from the classes that may discriminate them efficiently as well as to ask an expert such a new attribute.

4.3 Representation of domain knowledge

In order to detect semantic improprieties, we must know about the domain and explain the decision tree by mapping the domain knowledge onto the nodes in the tree.

4.3.1 Abstract relationships between a class and attributes

Although it's difficult to describe exact rules, a human expert can prescribe explicit and abstract relationships between a class and the attributes. This knowledge is essential for the explanation-based reasoning of KAISER. KAISER formalizes this knowledge with the following three types of conditions: (1) essential conditions (Econd); (2) desirable conditions (Dcond); and (3) permissible conditions (Pcond). For example, it's an Econd for an insects that they have six legs. Their Dcond will be that they have four wings and the Pcond may be that they have eight legs.

4.3.2 Inter-attribute relationships

Some of the attributes can be derived from other attributes. For instance, the number of claws will be derived from the number of legs. We can define some abstract and virtual attributes such as density from the mass and the volume. KAISER accepts this knowledge in the form of LISP function tables. For example, DENSITY attribute can be defined as (DENSITY (/ MASS VOLUME)). This knowledge is necessary to interpret the descriptors in the abstract knowledge described in *4.3.1*, because MASS and VOLUME may appear in the decision tree instead of DENSITY which is used in the domain knowledge.

4.3.3 Ordinal relationships among attribute values

If an attribute represents an ordinal feature, the values have ordinal relationships. This information is necessary to get adjoining brothers of a node to eliminate many kind of improprieties. For example, (Size-of-

Tshirt L LL M) means that the 'L' link of attribute Size-of-Tshirt can be merged into an another link if it has the rest values 'LL' and 'M'.

4.4 Mapping of domain knowledge

KAISER examines two types of mapping of domain knowledge onto decision tree. One is theory-based mapping (TBM), and the other is result-based mapping (RBM). The semantic improprieties are detected based on these mapping results. The TBM refers to the conditions on a path to a leaf and those in the domain knowledge to presume the class and the belief for each leaves. We call the sets of class and the belief support factors. They are calculated by the following algorithm:

For all domain knowledge and paths in a decision tree,
(1) if a path includes all of the Econd of a domain knowledge, then set the support factor of the path by the knowledge to 1.0;
(2) if a path does not satisfy (1) but includes the conditions generated by replacing some conditions in the Econd by Pcond, then set the support factor by the knowledge to 0.8^N, where N is the number of replaced conditions; and
(3) if a path satisfies (1) or (2), and it also includes some conditions in Dcond, then multiply the support factor by 1.2^N, where N is the number of matched Dcond.

While the TBM infers the class of the path based on the domain knowledge without believing the induction results, the RBM verifies a path to a classified leaf based on the domain knowledge. RBM is examined after TBM for each path whose class is determined by induction but not supported by any TBM, that is, no domain knowledge could cover the path (no explanation improprieties). If a path includes some of the Econd in a domain knowledge that supports the class of the path, then the path is marked as an insufficient path with the missing conditions. In many cases, these conditions become necessary when plenty of training examples are specified.

4.5 Semantic improprieties

Ideally, all of the conditions in a decision tree will be assigned to the descriptors in domain theory, in other words, they will be interpreted in terms of the domain theory. However, this mapping practically fails because of a lack of or fault in examples and domain knowledge. For example, consider the following example: You are trying to acquire a knowledge for discriminating a beetle from other creatures according to their appearances, i.e., number of legs, wings and eyes. You know that an insect has four wings and six legs, and beetles belong to insects. However, you get a decision tree that insists that if a creature has six legs and two

wings, it's a beetle. (Although a beetle really has four wings, let's suppose that it has only two because the outer two look like shells.) This example suggests many improprieties for interviewing. That is, which must be revised, domain theory that insists four wings or examples? Are beetles really insects? A mosquito also seems to have only two wings. Is it a beetle? Etc. Currently, KAISER provides the following five types of semantic improprieties that have their own peculiar elimination methods.

- *Explicable NIL node impropriety* The conditions to a nil leaf are explicable by the domain knowledge. Similar improprieties are also defined for inseparable nodes and noisy nodes. Their native elimination method is to adopt the derived class for their leaves.
- *Contradictory explanation impropriety* The conditions to a leaf are explicable by a domain knowledge that belongs to a different class. We can recommend many methods to eliminate this impropriety. Most characteristic recommendations will be to specialize the Econd or Pcond of the miss-matched domain knowledge, and/or to generalize the Econd or Dcond of the domain knowledge that should match and get higher support factors.
- *Multiple explanation impropriety* More than one explanations are suggested by domain knowledge. This impropriety recommends to change an Econd to a Dcond to lessen the support factor of a knowledge that is not so important, and/or add some condition to Dcond to strengthen the factor of a preferable knowledge.
- *No explanation impropriety* No explanation is given. It recommends to generalize the Econd and Pcond of a knowledge that should match, and/or ask a new domain knowledge for the leaf.

4.6 Impropriety interpretation and questionnaire generation

If it is possible to offer the best recommendation based on the detected impropriety itself, we don't have to separate the impropriety detection and impropriety interpretation. With some simple improprieties this will be possible, but not with most. To understand this matter, let's think of previous examples again. In the T-shirt example, we discussed an unknown leaf problem. Since it was a quite simple problem that requires the knowledge only about the order among the size LL, L and M, we could easily infer the price of size L. However, how about the following similar problem: If a creature has six legs, then it's an insect. If it has eight legs, then it's also an insect, i.e., a spider. Then what is it if it has seven legs? The most common answer will be that there is no such creature, but a witty answer may be a crippled spider, but not a crippled beetle. To even deduce the common answer, we have to know much more about the domain. One major difference between these two examples is that, in the former example, all of the leaves including the unknown L-size leaf are covered by a single domain theory about the size of T-shirt,

while the latter requires individual theories to explain each leaf. Especially, it's remarkable that there is no (common) knowledge that explains seven-legged creatures, which is a typical semantic impropriety. This fact suggests that the combination of detected improprieties, especially a combination of structural and semantic ones, is an important clue for identifying an underlying fundamental impropriety. This is what we want to say by impropriety interpretation. Because of the wide variety of the interpretation knowledge, KAISER has not yet provided enough interpretation knowledge. Current knowledge includes:

(1) invalidation of the twin node impropriety when the twin nodes are explained by another domain knowledge;
(2) cooperation of the inseparable node and multiple explanation impropriety to demand new attribute; and
(3) cooperation of the contradictory and noisy or nil node improprieties to demand new knowledge.

Another point of issue is the query generation. To achieve intelligent knowledge acquisition, sophisticated interviewing is indispensable. Since it's very hard to make human-like conversation in a natural language, KAISER does not aim at it but provides sophisticated questionnaires (i.e., queries and choices) according to the improprieties proposed by the interpreter. Although each impropriety has its own peculiar elimination method, it will not a good idea to present all of them in the questionnaire at once and ask to answer all of them. Currently, KAISER employs the following nine criteria to select some important improprieties. They are examined in this order.

(1) Semantic improprieties take precedence over structural ones.
(2) An impropriety that recommends modification of tree take precedence over those that require refinement of domain knowledge.
(3) An impropriety that recommends structural reconstruction of tree takes precedence over those that only prune or change the labels of nodes.
(4) Any impropriety that asks for selection take precedence over those that require description.
(5) An impropriety that matches a node which is related with more improprieties takes precedence.
(6) Noisy nodes improprieties take precedence over those of NIL leaves;
(7) Inseparable nodes improprieties take precedence over those of noisy nodes, if the inseparable node has more examples than the threshold of noisy node detection.
(8) An impropriety related with more nodes takes precedence.
(9) An impropriety related with upper node (i.e., near to the root) takes precedence.

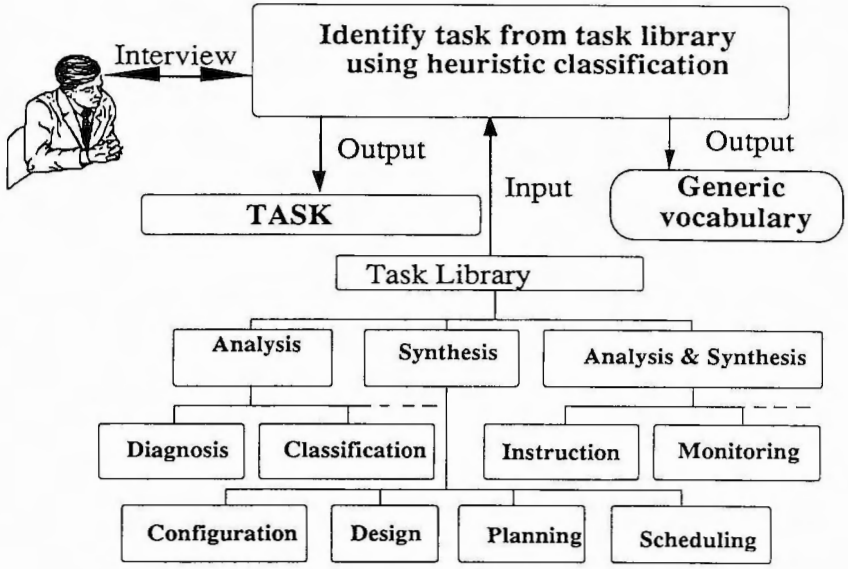


Figure 3.2. Task identification in MULTIS

3.2. Task Analysis and Synthesis

Next step is to analyze and synthesize the task identified in the previous phase. As figure 3.3. illustrates, MULTIS makes use of the generic vocabulary that corresponds to the task identified in the previous step to determine what type of generic tasks can be related to it. Then, with the generic tasks identified heuristics are employed to single out building blocks that can be used as part of the search-control structure for the target KBS through case-bassed reasoning.

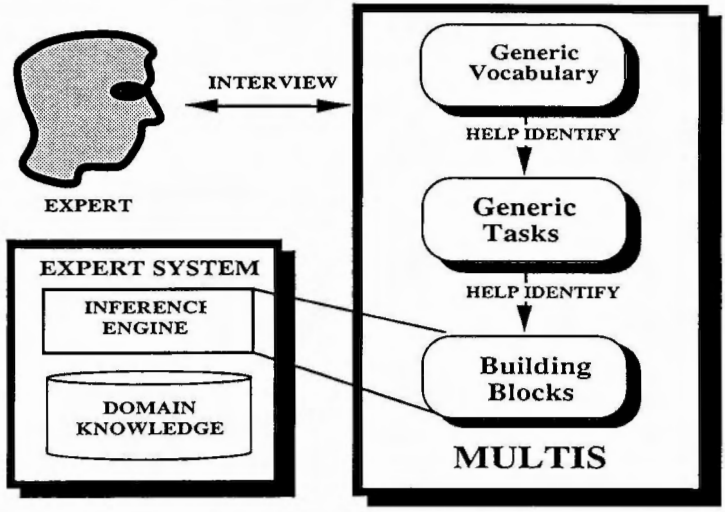


Figure 3.3. Generic vocabulary helps identify generic tasks which help identify building blocks.

MULTIS makes use of case-based reasoning during its interview to reuse previous combinations of generic tasks encountered in past interviews to avoid combinatorial explosion of generic tasks. The vocabulary corresponding to the expert's task and its translation are employed as indexes to search in the case base for a similar case or cases. Later, the generic task combination from the extracted case is modified with use of a graphical interface under development to accurately represent the task being performed by the domain expert. Finally, the generated generic task combination is analyzed heuristically to identify the right mix of building blocks that should be used as part of the search-control structure of the target KBS. The details of case-based reasoning are given in section 4.

PS modeling and inference engine mapping take place in this phase. Interactive modeling is implemented through the graphical interface mentioned after a prestored modifiable model has been found. The inference engine mapping is performed transparently to the interviewee after heuristics are employed to identify building blocks useful to reflect the patterns given by the model.

3.3. Acquisition of Domain Knowledge

In this step, the required additional task-implementation knowledge and domain knowledge is elicited from the domain expert. For this purpose, the assumption is made that each generic task and each building block, related to the task involved, have structures that guide the processes of elicitation of domain knowledge, and task-implementation knowledge respectively. Chandrasekaran and McDermott have made similar assumptions[1, 5]. Put in other terms, a strategy for knowledge acquisition can be directly mapped to each generic task or each building block.

As explained before, the main characteristic of MULTIS is its ability to help acquire knowledge for various tasks. Therefore, we have concentrated mostly in the design of inference engines for those tasks through building block combination. However, acquisition of domain knowledge is also a very important issue that cannot be ignored. Consequently, we are carrying out intensive research on the mapping of existing knowledge acquisition methods to the search-control structures resulting from MULTIS's interview.

3.4. Evaluation

The last phase in the interview strategy is system evaluation. At this point, MULTIS tries to solve a problem already solved by the expert being interviewed. If the expert is satisfied with the solution then no more questions are asked and the target KBS is constructed. Otherwise, MULTIS checks for missing or erroneous domain knowledge, or erroneous search control knowledge.

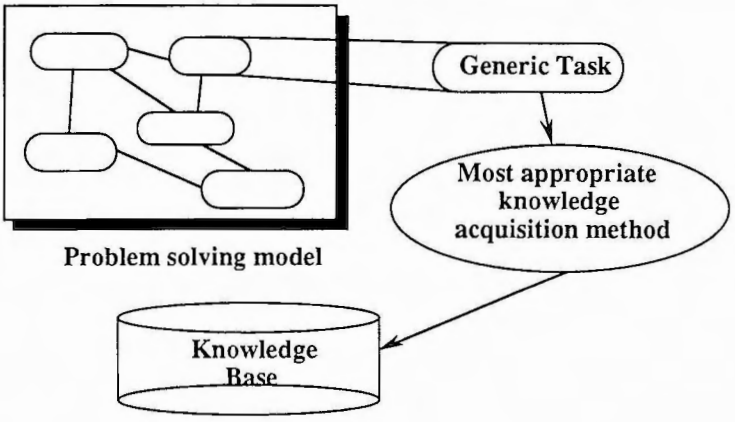


Figure 3.4. MULTIS uses most appropriate knowledge acquisition method attatched to each generic task.

4. Case-Based Reasoning

The job of MULTIS's case-based reasoner is to choose a suitable case saved in a case base, with a cognitive structure (generic task combination) and a search-control structure (building block combination), that can be feasibly modified so as to reflect the PS structure of the interviewee and the corresponding KBS's inference engine respectively. The following subsections give an overview of the issues involved in MULTIS's case-based reasoning (CBR).

4.1. Case Representation

In MULTIS's case base, a case is divided into three parts: the system and expert information, the cognitive structure and the search control structure for the target KBS which includes knowledge-acquisition-method mapping. Cases are kept in case bases according to type of task, that is, for each task that MULTIS handles there is a case base with cases already constructed for that task. This organization makes it possible to increment the number of tasks that MULTIS can manipulate without major modifications to MULTIS's overall structure. Figure 4.1 gives an example of a case, with its three main parts, in MULTIS's case base. This case corresponds to Scheplan, a process scheduling expert system [6].

4.2. Indexing

What can and cannot be used as index for a case is a complex and at the same time necessary decision. For simplicity reasons we have chosen an indexing method similar to Protos's, an earlier interview system [7] that helps keep the number of possible choices for indexes small without losing effectiveness in search. Our indexing method is based mostly on the generality introduced by generic vocabularies. MULTIS uses four kinds of indexes in searching the case base: remindings, censors, prototypicality and difference links.

Remindings are indexes that positively denote similarity between one case and another. In MULTIS the way the generic vocabulary is translated and the pattern they present are treated as reminders. In other words, the interviewee-translated generic vocabulary may correspond to previous ones. After the most important vocabulary has been translated by the interviewee, it may show some resemblance to vocabulary of other cases.

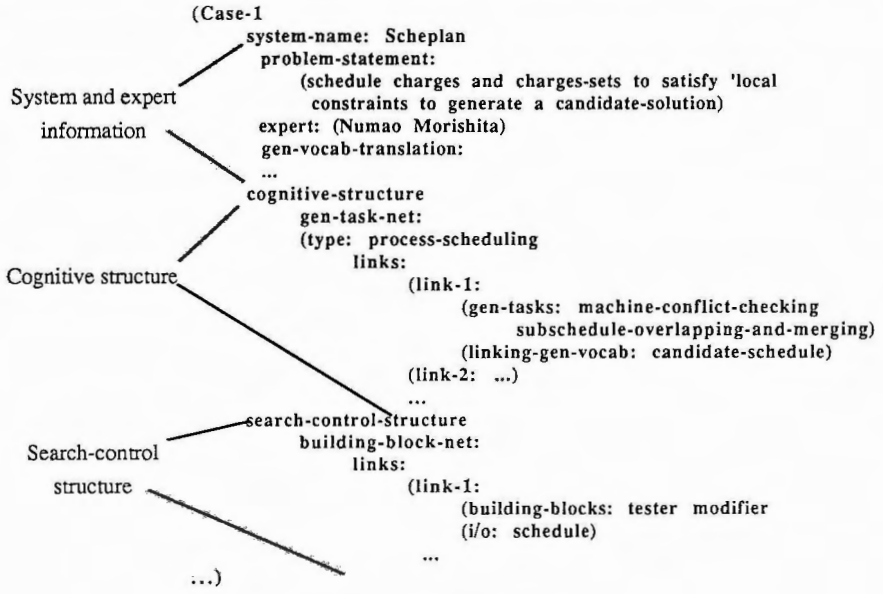


Figure 4.1. A case in MULTIS.

Censors are indexes that diminish similarity between one case an another after it has been established. A censor works in the opposite way of reminders. A censor can be the use of generic vocabulary known to be exclusive to the case or cases being investigated. The fact that a generic vocabulary is a generalization of task's vocabulary doesn't mean that all experts performing that task use the whole of the vocabulary. For this reason, the presence or absence of the same vocabulary from the interviewee and another case may become remindings or censors respectively.

Prototypicality, as the word implies, refers to how typical a case is with respect to others. In MULTIS, prototypicality is represented as a hierarchy tree of cases categorized by prototypicality. The topmost root of the tree corresponds to the type of task the tree represents. The children are the original cases transcribed into the case base when the particular task they are cases of was incorporated. The rest of the children thereof are the cases found through modification of the parent cases.

A difference link is a link that identifies important differences between cases. More specifically, a difference link between two cases tells what important reasons make them different, so that when a new case is being classified further choices are efficiently investigated. A difference link is usually established between two cases when they are similar but the newer one couldn't be constructed as a modification of the older one.

4.3. Case Search

The success in finding the most-similar cases, depends on our indexing method. The purpose of the search is to discover a case or cases similar to the interviewee's case. Meaning that one or more cases can be found to be similar. The most similar case is then selected interactively by the user and the rest of the cases are left as candidates to be investigated if the most similar one fails to satisfy the interviewee.

First of all, MULTIS identifies the case base that corresponds to interviewee's task. Since MULTIS can perform knowledge acquisition assistance for various tasks, the generic vocabulary, generic tasks and case bases corresponding to each task are saved separately. Once the task, its corresponding generic vocabulary and generic tasks have been identified, MULTIS must retrieve the appropriate case base.

Then, with the remindings MULTIS explores which topmost cases in the prototypicality tree are most similar. Since the hierarchy tree represents prototypicality of cases, remindings are used to explore the tree from top to bottom to find cases similar to the interviewee's. This method of searching ensures that most prototypical cases are investigated first, thus adding some efficiency to the search.

Finally, MULTIS uses censors and difference links to further explore and order up to three of the best matching cases. Censors are employed to eliminate cases that may have contradictions with the interviewee's generic vocabulary. Difference links are also used, in interactive interview, to further investigate which cases are most similar to the interviewee's. After the similar cases have been ordered by similarity, only up to three of the cases are returned.

4.4. Case Modification

As one might expect the success of case modification determines that of CBR. In MULTIS, whether a case can be modified or not to reflect the interviewee's PS structure depends on three major factors: 1) the case is similar enough to the interviewee's, 2) the interviewee understands the concepts of generic tasks, and 3) a flexible and easy to use modification interface is provided. For this reason we are developing GTS (Generic Task Simulator), an interface that instructs the interviewee on the concept of generic tasks and generic task networks and provides flexibility for modification. The following three subsections provide an overview of GTS along with some important issues related to its usage.

4.4.1. Generic Task Networks

A generic task network is a network that uses generic tasks to model the PS process carried out by a particular expert of some task. A generic task network is composed of links between generic tasks which represent a relation between them. This relation is given by the kind of data being shared and by whether the relation is of precedence, modularity or modification. The kind of data being shared correspond to special kinds of generic vocabulary as illustrated by the link annotations in figure 4.2, which represents the generic task network of Scheplan. The type of relation is given in the figure by the type of arrow or line. As explained previously, the interviewee should understand and agree with the generic task network that represents his/her task. For this purpose we are currently devising GTS.

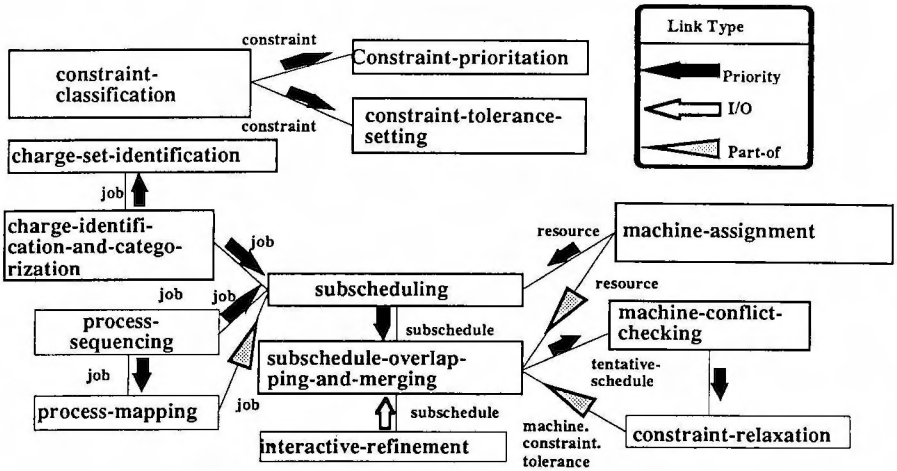


Figure 4.2. Scheplan's generic task network.

4.4.2. Generic Task Simulator

GTS is currently under development by our team, but once completed it should be able to 1) instruct the interviewee on the meaning of a generic task network, 2) show a simulation of the PS process through animation of the generic task network, and 3) provide flexibility to the interviewee to modify a case found to be similar to his/her case.

GTS presents the interviewee up to three similar cases from which he/she has to choose the most similar one. Before the interviewee chooses a case, the notion of generic task networks is explained with the cases step by step. The interviewee can stop the instruction at any time and ask about its contents by clicking on any place on the generic task networks. Once the interviewee has understood the generic task networks, he/she should be able to choose the most similar one. At this moment, the generic task network is posted in the modification window with the untranslated generic tasks so that they can be translated to reflect the interviewee's PS (See Appendix B for an example).

4.4.3. Interactive Modification

GTS provides flexible, interactive modification because the interviewee can navigate any of the similar cases, copy parts of them or delete unneeded parts from the posted generic task network, by simply clicking, dragging the mouse and using pop-up menus. Furthermore, more than one similar generic task network are presented to the interviewee so that he can use them as examples or reference when modifying his own generic task network. On the ideal situation, the interviewee could construct a generic task network form pieces of other networks.

4.5. Case Evaluation

Once the interviewee is satisfied with the modified generic task network, MULTIS analyzes it to find patterns that identify building blocks. Since one or more generic tasks can be mapped to a building block, a knowledge base is kept with heuristics about how generic tasks should be mapped. This mapping process is transparent to the interviewee and its success depends on the validity of the heuristics employed. Because of space limitations we will discuss extensively this issue in a later paper.

4.6. Case Base Update

Once MULTIS has acquired the additional domain knowledge and the interviewee is satisfied with the results of his/her test problem, the need arises to update the case base. First of all, the case must be put in the prototypicality tree under the case from which it was modified. A strong difference link is

established between them to express their differences. Then the generic vocabulary translations are analyzed to identify remindings and censors. Finally, the case base is saved in the appropriate place, that is, the place corresponding to its task.

5. Conclusion

MULTIS is still under implementation on a MacIvory, a Macintosh II computer with a Symbolics Lisp Machine. Once fully constructed it should be able to:

1. Engage in coherent and smooth interview with domain experts with help of generic vocabulary.
2. Model the PS processes of domain experts in various fields in the form of generic tasks networks.
3. Identify which prefabricated search control structures can be implemented to construct a KBS that performs the desired problem-solving.
4. Make use of strategies related to the generic tasks and building blocks to elicit domain and task-implementation knowledge.
5. Help construct or quickly prototype KBSs for various tasks.
6. Make use of CBR to improve efficiency during the interview.

Currently, MULTIS possesses a case base for process scheduling with seven cases. We plan to add other cases that cover other tasks such as planning, monitoring and diagnosis. Also, mapping between generic tasks and building blocks, a very important issue, is being investigated along with the corresponding mapping to domain knowledge acquisition methods.

References

[1] Chandrasekaran, B. (1986) Generic tasks in knowledge-based reasoning—high level building blocks for expert system design. IEEE Expert, pp. 23-30.

[2] Hamanaka, T. and Mizoguchi, R. (1990) Identifying building blocks in knowledge-based systems—solution/solution method taxonomy—. In Proc. of the 4th Annual Conference of the Japanese Society for Artificial Intelligence. (Japanese) pp. 607-610.

[3] Boose, J. H. (1989) A survey of knowledge acquisition techniques and tools. Knowledge Acquisition, 1, pp. 3-38.

[4] Bennett J. S. (1985) ROGET: A Knowledge-Based System for Acquiring the Conceptual Structure of a Diagnostic Expert System, J. of Automated Reasoning I, pp. 49-74 .

[5] McDermott, J. (1988) Using problem solving methods to impose structure on knowledge. Proc. of the International Conference on Artificial Intelligence Applications. pp. 7-11.

[6] Numao, M. and Morishita, S. (1988) Scheplan—A scheduling expert for steel-making process, Proc. of International Workshop on Artificial Intelligence for Industrial Applications, IEEE, pp. 467-472.

[7] Bareiss, R. (1989) Exemplar-Based Knowledge Acquisition. Chandrasekaran, B. (ed). Academic Press, Inc.

Appendix A

The name of generic task is somewhat confusing because most people who first hear the term relate it to a task such as diagnosis or planning (software engineers relate it to software tasks or routines). This term has been used in a similar context by Prof. Chandrasekaran of Ohio University [1], but the authors of this paper give it a somewhat different meaning. In this paper generic tasks are defined as cognitive primitives that can be employed to explain or analyze given problem solving behaviors. An emphasis is made that generic task are of conceptual character and no claim is made that such generic tasks are "the" primitives used in human problem solving, nevertheless they can be employed to explain it to some extent. The following is a non-exhaustive enumeration, in alphabetical order, of generic tasks for planning and scheduling with a brief explanation.

(1). Alternative solution generator (generative task)

In processes scheduling and other tasks, it is customary to generate more than a single solution for a problem. The reason for this is obvious because an alternative must exist in case that the original

solution fails to solve the problem adequately. Usually this task comes as a natural by-product of generating the tentative solution, but in some cases it is generated alternatively to avoid falling in the same problems that didn't permit sucessfull implementation of the tentative solution.

(2). Constraint conflict checking (testing task)

Constraints are in conflict when they require the same resource simultaneously or when two or more specifications or conditions cannot be satisfied. If by trying to satisfy two constraints simultaneously, the related resource capacity is violated then a conflict between those constraints is detected.

(3). Constraint identification and categorization (generative task)

Constraints are usually categorized into categories that reflect the way those constraints are used to reduce the search space. There are constraints at various levels of problem space restriction. Some constraints are used to create a rough solution. Others might be used to refine that rough solution into a better solution. Yet, others might be used to check validity of that solution. Constraints have to be identified before they are categorized. When actions, jobs, objects or facilities are identified, this task is carried out. The reason for this is that somehow all of them restrict problem solving some way or another. Therefore, the problem space for this task changes according to these restrictions. That is, the problem space might consist of identifying job constraints, resource constraints, facility layout constraints, and organizational constraints. Of course, all these constraints might still fall into different constraint categories for problem solving which is the reason for constraint categorization.

(4). Constraint prioritation (generative task)

Priorities are similar to categories but differ in that for each category constraints might have different priorities. Priorities can be thought of as hierarchical importance of constraints in a particular level of problem solving. Priorities have precedence orders.

(5). Constraint relaxation (modifying task)

The input to this task are the constraints in conflict plus their priorities and tolerances along with the differential of resource capacity which created the conflict. The outputs are the successfully relaxed constraints or a conflict notification in case of failure. The problem solving mechanism uses the constraint priority knowledge and tries to relax the constraint with the least priority.

(6). Constraint tolerance setting (generative task)

The problem space in this task consists of identifying the tolerance of each constraint. This tolerances might be represented as bounds. Since tolerances of constraints to partial change depend on other constraints.

(7). Constraint updating (modifying task)

Again, the inputs for this task are the changing constraint (i.e., the constraint being added deleted or modified by the user) and its relation to other constraints. The problem solving mechanism makes use of all other constraint related tasks. It uses the constraint identification task to add a new constraint. It uses constraint prioritation and constraint tolerance setting for each new constraint. It also uses the latter two to change priorities and tolerances of a given constraint.

(8). Goal Modularization (generative task)

Subgoals might be generated by considering each job for processes to be performed on those jobs. There might be one or more jobs when problem solving is taking place. Some of the actions to be performed might be shared by jobs. The purpose of this task is to produce adequate subgoals that could efficiently be carried out for the completion of the final goal (e.i., successful completion of all jobs). The problem solving mechanism usually takes into consideration the jobs and the layout of the facility or processing resources (such as the machinery or a robot's arm). The problem solving mechanism takes job by job and determines the subgoals of each job according the actions known to apply to it. This task forms subsolutions for each job but still their relation has to be checked so that they can be merged together into a final solution.

(9). Job-batching (generative task)

Jobs may sometimes have similar process that, if performed simultaneously, permit more effecient and inexpensive factory operation. In this cases, the batching of jobs for those processes that can be treated the same, is indispensable.

(10). Job identification and prioritation (generative task)

The inputs to this task are all jobs to be scheduled. In scheduling there are several jobs, so resources have to be carefully assigned to avoid conflicts. In action planning there is usually one job and one material resource for which the order of actions to be performed has to be decided.

(11). Job updating (modification task)

The inputs for this task are the current knowledge (priority relations) and data about jobs along with the job that has to be changed (i.e., the job that has to be added, deleted or reprioritized). The outputs are the modified data and knowledge bases. In most cases the relation among jobs change when a job appears, disappears or its priority gets modified. For this reason the problem solving mechanism of this task accepts the job and its related change and incorporates that change into the jobs relation structure. It does so with help of the job prioritation task.

(12). Preference allocation (generative task)

Preferences help enormously when selection of jobs, batches or resources cannot be done with the sole consideration of constraints. Preferences work in a positive manner as opposite from constraints which help the seltection process by negatively imposing restirctions. Therefore, it is necessary to allocate preferences to jobs, batches or resources.

(13). Process mapping (generative task)

In process scheduling the experts usually have a set of possible processes that can be associated with certain jobs or batches. Therefore, experts spend some amount of time mapping the adecuate processes to each job or batch.

(14). Process sequencing (generative task)

After processes have been mapped to jobs or batches, a sequence of those processes must be found to satisfiy the constraints involved. This is probably the most difficult and important of all tasks since it is in this task where a plan is generated. Sequencing has never been an easy task because many factors have to be considered, because both local and global constraints must be considered at one time or another.

(15). Resource capacity analysis and assignment (generative task)

The inputs are the subgoals, the routes and the resources. The outputs are the assignment of the the resources to the subgoals and the updating of the resource capacity data. Resources should be represented in a way that reflect their capacity. The reason for this is that as resources are assigned their change capacity. This is the case for most resources including machinery which usually breakdown when their limit is reached.

(16). Resource updating (modifying task)

Inputs to this task consist of the changing resource and its relation to subgoals (or actions). A changing resource is one for which the level was not the one expected. Or which is no longer available for some reason.

(17). Solution modification (modifying task)

The purpose of this task is to modify the solution of a previous problem so that it can be used to solve a new problem. This task usually follows specification comparison and solution retrieving

(18). Solution refinement (modifying task)

The initial subgoals of each job are the inputs to this task. The outputs are more efficient subgoals. A categorization of subgoals is made to find the relations between the overall existing subgoals. There are three special kinds of subgoals: independent subgoals, serializable subgoals, and non-serializable subgoals. Independent subgoals are those subgoals for which the resources and constraints used in two different subgoals are not shared. Serializable subgoals are those subgoals which are not independent, but that can be performed one after the other without ever disturbing the one performed first. Non-serializable subgoals are those subgoals which must violate previously satisfied subgoals in order to make further progress towards the main goal (Korf, 1987). This knowledge about subgoals is used along with the current state of the subgoals to categorize the existing subgoals into these categories.

(19). Subsolution merging (generative task)

The inputs for this task are the subgoals generated during goal modularization and refined and validated latter by subgoal refinement and validation. The output is the best tentative solution. First of all, problem solving mechanism generates some alternative routes that might satisfy the constraints imposed in each subgoal. After the alternative routes have been generated, one must be chosen using the preference knowledge. This preference knowledge plays an important role since it will be used as a catalyst to choose the optimal route.

Appendix B

This appendix presents an example of an interview session with a hypothetical expert that performs PS in the area of process scheduling. The example has been simplified for space limitations.

B.1. Task Identification

First of all MULTIS gets bookkeeping information from the interviewee such as name, occupation, place of work and so on. Then it identifies what is the main PS task the interviewee performs through interactive interview. For this purpose we have chosen Hypercard which offers a flexible interface. As in figure B.1, the tasks are enumerated with options to get definitions, examples or choose the task. In this case, the interviewee chooses scheduling after investigating other candidates such as planning or configuration. Since scheduling can mean process scheduling or timetable scheduling another card appears in which they both are explained. In this example, the interviewee chooses process scheduling.

At this point, the case-based reasoner retrieves the generic vocabulary that is most peculiar to process scheduling and attempts to translate it to the interviewee's own vocabulary. This is done through random examples as explained in section 3.1. Figure 3.1 corresponds to this example.

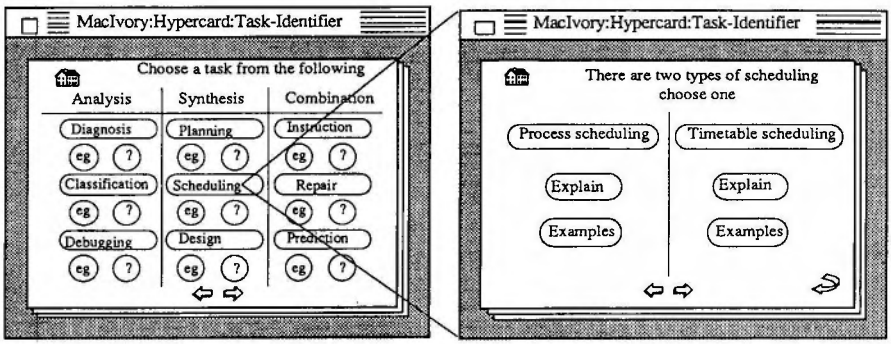


Figure B.1. Task identification with Hypercard.

B.2. Task Analysis

After the case base for process scheduling has been searched with the remindings and censors given by the generic vocabulary translation, the prototypicality of the cases in the case base and the difference links involved, MULTIS's CBR module retrieves the three most similar cases and puts their generic task networks in the GTS window. Figure B.2 correspond to the GTS window for this example. As you can see in the figure, the interviewee has chosen Scheplan as the most similar case, thus its untranslated generic task network is posted in the modification window. Notice that the generic task network of Scheplan corresponds to that of figure 4.2. The interviewee then copies some parts of the other similar cases, deletes others and translates some of the generic tasks in the network resulting in the network in figure B.3.

Next, the unchanged portions of the generic task network that were mapped to building blocks in the previous cases are used to identify useful building blocks for this case. Mapping heuristics are also used to identify patterns in the generic task network that require other building blocks. The result corresponds to the inference engine of the ES being constructed.

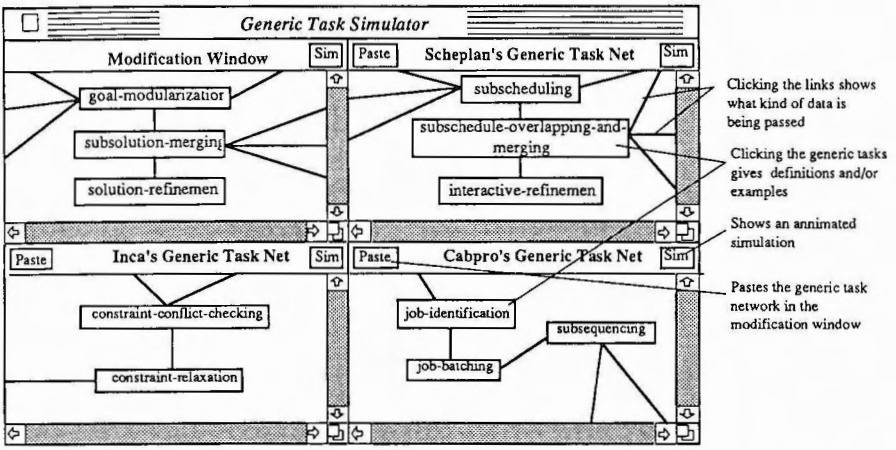


Figure B.2. GTS window.

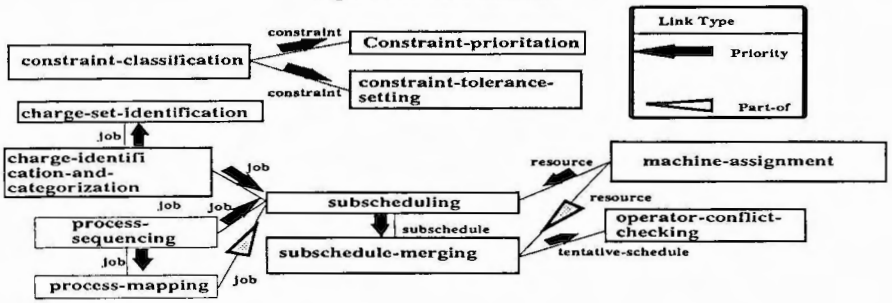


Figure B.3. Resulting generic task network.

B.3. Acquisition of Domain Knowledge

In the generic task network presented above the data being transferred between generic tasks is given in the form of generic vocabulary. That transferring of data and the structures of the generic tasks help guide in the acquisition of domain knowledge. For example, instantiation of the generic vocabulary to specific instances provide data useful for PS. In this case, the expert system knows it must ask for jobs to be scheduled. Other generic vocabulary and tasks help in a similar fashion.

B.4. System Evaluation

After MULTIS performs acquisition of domain knowledge, it asks the interviewee to input data for an already solved problem. It then makes the schedule with the knowledge available, and asks the interviewee if he/she is satisfied. If the interviewee is not satisfied, the domain knowledge is checked for insufficiencies or errors. Otherwise, MULTIS outputs the expert system and updates the case base and the indexes so the new case can be used in future interviews. In this example, the new case is saved in the process scheduling case base and the prototypicality tree for this case base is updated.

Hierarchical Knowledge Representation based on Approximations

Kenichi Yoshida Hiroshi Motoda
Advanced Research Laboratory, Hitachi, Ltd.
Hatoyama, Saitama 350-03, Japan
yoshida,motoda@harl.hitachi.co.jp

Abstract

It is difficult even for an expert to fully understand the behavior of a complex system. Human beings can view systems at different levels of abstraction based on it's functional structure. The way we understand a complex system is mostly hierarchical.

The primary objective of this research is to develop a method for hierarchical representation in a complex system, which enables acquisition and simultaneous utilization of knowledge expressed at multiple levels with different abstractions based on approximations. A new method is also proposed, which is an extension of the existing explanation-based learning method, to support the construction of the consistent hierarchical knowledge base which complies with the proposed representation scheme.

The examples are taken from the domain of analog/digital circuits to explain the proposed representation and the method to construct a consistent hierarchical knowledge base. Also discussed is how the hierarchical knowledge base can be used in problem solving.

1 Introduction

It is difficult even for an expert to fully understand the behavior of a complex system. Human beings can view systems at different levels of abstraction based on it's functional structure. Furthermore, s/he tries to identify the role of each part in the working mechanism of a system. S/he then identifies a set of elements as one meaningful composite and regards it as a single entity. A complex system is most often understood and described as a hierarchy.

Hierarchical knowledge representation is needed to realize the way an expert understands a complex system. Qualitative reasoning can be regarded as a method to express physical systems [1, 2, 3]. The need for a hierarchical approach in describing a complex system was already addressed in an early phase of the research [4, 5]. Here, the concept of hierarchy is based on the difference in the degree of approximation in functional descriptions. The works of Doyle [6], Bennett [7], Mozetic [8] and Falkenhainer and Forbus [9] are the important prior research related to this issue. These studies decrease the complexity of the described system using various approximations.

Few efforts, however, have been carried out to construct a consistent hierarchical knowledge base. Almost all of the works related to this issue rely on the user effort to achieve a consistent knowledge base. Component consistency is very important. For example, a knowledge compilation method [10] can be applied to

make a diagnostic system from a hierarchical knowledge base. However, inconsistency between each hierarchical level will result in a wrong answer or lack of an answer. Also, a top down refinement design system [11] can be developed using a hierarchical knowledge base. The problem of inconsistency in this case is that the system can not refine a design plan if the knowledge base dose not maintain a consistent relationship between each hierarchy.

To construct a consistent hierarchical knowledge base, two important issues arise. First, information about approximations used in each level should be stored and retrieved as necessary. Without this information, contradictions introduced by the approximation between each level decreases the functional ability of the knowledge base. Second, complex systems have many aspects, and there are many domain theories to apply to each aspect. A representation method should be expressed clearly to manipulate this information.

The primary objective of this research is to develop a method for hierarchical representation of a complex system which maintains consistency among the representations in each level of the hierarchy. This method is an extension of the existing explanation-based learning method [12, 13], and is applicable to an intractable domain theory.

In the next section, the proposed hierarchical representation scheme is described, together with the knowledge acquisition method to support the construction of a consistent hierarchical knowledge base. Also discussed is how this hierarchical knowledge base can be used in problem solving. In the third section, some acquired knowledge is explained, as well as the detailed acquisition process to show how this scheme represents the knowledge. Also explained is how the computer program constructs a consistent hierarchical knowledge base using an EBL-like method.

2 Describing a System by Hierarchical Representation

2.1 General Framework of Hierarchical Knowledge Representation

The knowledge structure of a complex system is shown in Figure 1. Complex systems have many aspects, as well as many applicable domain theories. In this figure, ellipses on the left represent the functional description of a complex system. In the domain of a digital circuit, these ellipses represent the functional description in various aspects. For example, there are three aspects in case of a NOR circuit (Figure 7 to be explained later). The lowest ellipse represents the analog behavior of the circuit. It includes equations each of which describes some aspect of physical principles that holds in the circuit; Ohm's law, Kirchhoff's law and the relationship between the voltage and the current of the transistor node. The second lowest ellipse represents the behaviors of switches. Each transistor can be seen as a switch. The knowledge at this level is the behavior of each switch and their connections. The uppermost ellipse represents the logical behavior of the NOR circuit which are constructed from switches. Hierarchical knowledge of a complex system consists of these types of knowledge. Here, the concept of deep/shallow is based on the

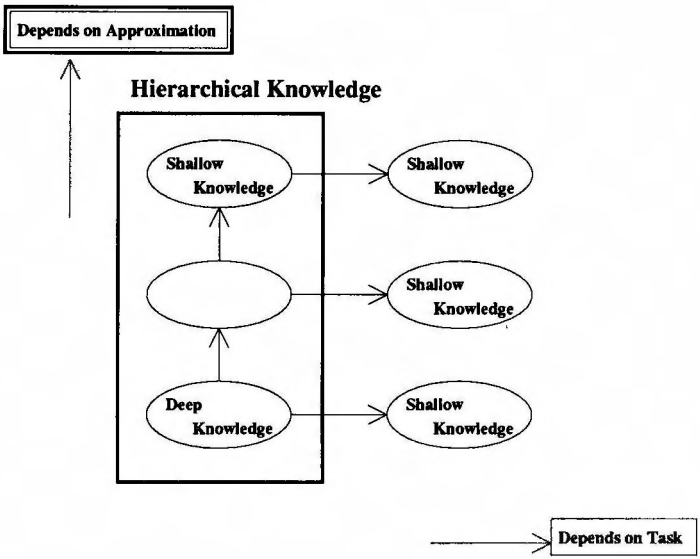


Figure 1: Knowledge Structure of a Complex System

difference in the degree of abstraction which we treat as approximations in the functional description.

Considering the two adjacent levels, the upper level knowledge (shallow knowledge) can be regarded as the specification, and the lower level knowledge (deep knowledge) can be regarded as the implementation. This nature is recursive; the description of some knowledge level can be regarded as a set of specifications of the deeper level knowledge, and also can be regarded as the implementation of the shallower level. For example a switch (specification) is implemented by a transistor (implementation) and a NOR circuit (specification) is implemented by three switches (implementation).

The ellipses on the right represent the task dependent knowledge. Sembugamoorthy and Chandrasekaran [10] developed a method to compile task dependent knowledge from a functional description. In conventional usage, shallow knowledge refers to task dependent knowledge, and deep knowledge refers to a functional description. The words "deep" and "shallow" are used to distinguish between functional hierarchies in this paper. In both cases, shallow knowledge can be constructed from deep knowledge using additional information. The conventional shallow knowledge is constructed using the task knowledge as this additional information; however, the information about approximations is used in this study. This paper is limited to the hierarchy of the functional description and the related issues.

The approximation process, which concerns the knowledge hierarchy, is shown in Figure 2. The lower/deep knowledge in Figure 1 assumes less approximation,

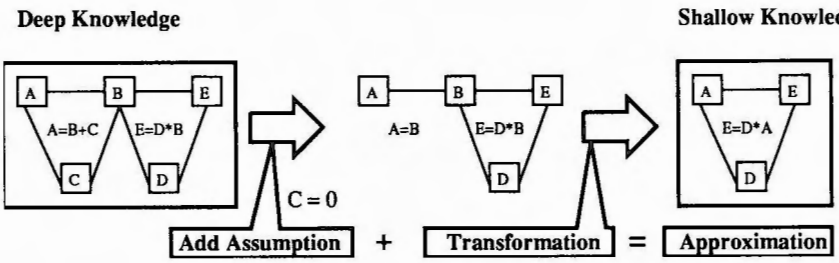


Figure 2: Approximation Process

and the upper/shallow knowledge is transformed from the deep knowledge by introducing some new approximations. In this study, approximations are processed in two steps. First, some assumptions are added to the deep knowledge, depending on the function of the whole or a part of the system. Next, a logically correct transformation is performed to make the shallow knowledge.

The shallow knowledge generated through this process is easy to use compared to the corresponding deep knowledge. The tasks which can be processed with this shallow knowledge will use it efficiently. The need for deep knowledge still remains, as it is a source for more detailed information. Information about the assumptions and the transformation is also needed to check the correspondence between deep and shallow knowledge.

2.2 Hierarchical Knowledge Representation and Problem Solving

Problem solving schemes using a hierarchical knowledge base are explained in Figure 3.

The output of the acquisition system is regarded as a set of consistent relationships between the specification (shallow knowledge) and the implementation (deep knowledge) at multiple levels. The entire hierarchy can be constructed from the deepest level implementation using these relations.

In a simulation system, the system can choose the appropriate knowledge level, depending on its purpose. The shallow level simulation uses less computing resources compared to the corresponding deep level simulation. In simulation whose purpose is to predict a summary of the complex component's behavior, the simulation system uses the shallow level knowledge for efficiency. However, when the purpose is to obtain detailed behavior, it must use deeper level knowledge. Simulation that simultaneously uses multiple knowledge levels is also possible. The consistency of the knowledge base is extremely important. Here, consistency refers to the explicit information about the assumptions. With this information, the simulation system can eliminate meaningless results which are based on contradicting assumptions.

The same information about assumptions is also required in the design system. Without this information, the top down refinement design system can not eliminate illegal design plans that include contradictions. This aspect of the design system suggests a new research issue: in order to achieve a top down refinement design

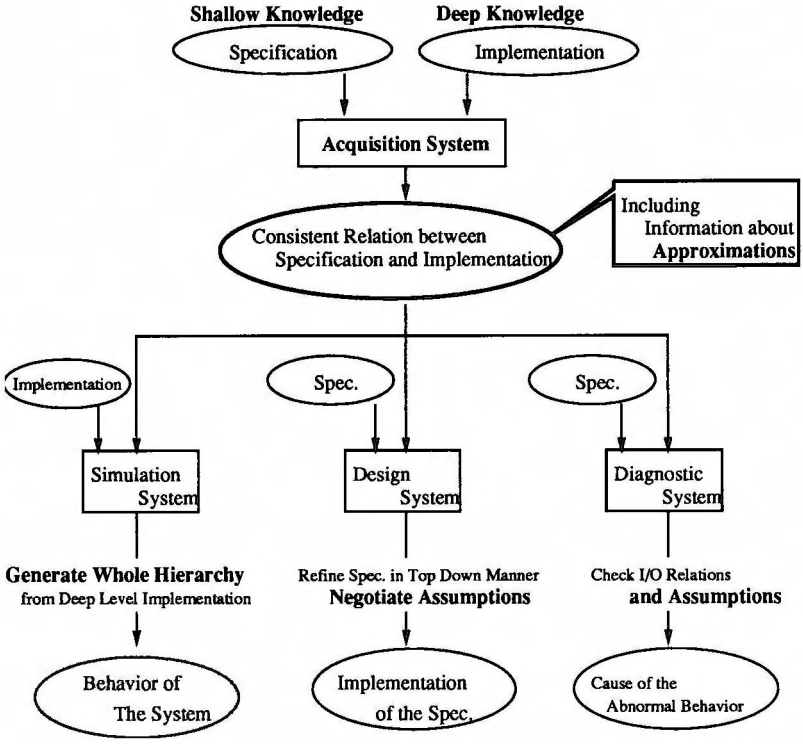


Figure 3: Hierarchical Knowledge Base in Problem Solving

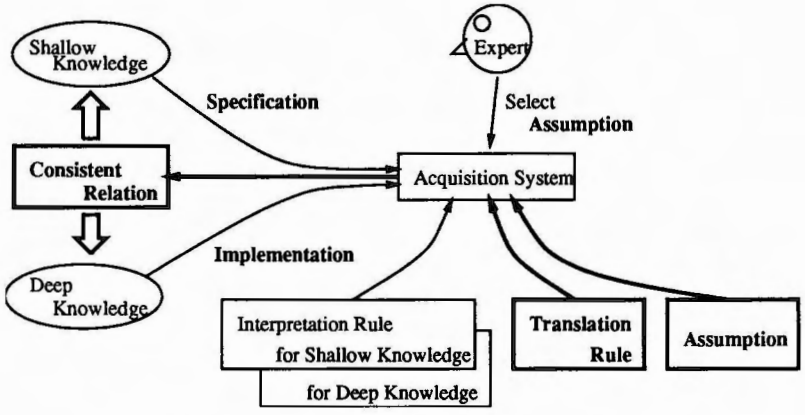


Figure 4: Configuration of Knowledge Acquisition System

system with a hierarchical domain model that involves assumptions in lower level knowledge, an efficient method to negotiate between contradictions that come from various assumptions should be developed. Most of the conventional design systems which use a top down refinement method propagate constraints which specify the function of the lower level parts. These constraints are kept in the following lower level design process. Steinberg [11] studied this constraint propagation. When the hierarchical domain model involves assumptions in a lower level, the design system sometimes comes across a totally new requirement in the lower level design process. The constraint propagation of this new requirement seems to degrade system performance, and a new method for the propagation should be developed. However, this is beyond the scope of the current research. Also, to ensure the functional ability of the design system, the implementation of each specification should be stored in the knowledge base. Thus, in this case, the requirement for consistency is even more important.

The simple diagnostic system checks the relations between input and output of the component. Additionally, it checks information about the environment necessary for the components to function. The assumptions recorded in the consistent knowledge base correspond to this environment. Thus, the diagnostic system, with this consistent knowledge base, clarifies not only the cause of the malfunction but also incorrect usage of the component by checking the environment. A hierarchical knowledge base enables top down diagnosis and pinpoints the cause of the malfunction efficiently. The malfunctioning component is investigated further to find the exact cause of the malfunction. However, a knowledge compilation method with a hierarchical knowledge base for the diagnostic system should be reconsidered to utilize a hierarchical/approximated nature of the component's knowledge. This issue is also beyond the scope of the current research.

2.3 A Knowledge Acquisition Method of a Consistent Knowledge Base

The configuration of a knowledge acquisition system [1] to support the construction of the consistent hierarchical knowledge base is shown in Figure 4. It includes a simulation system as a subsystem to check the behavior of the specification and the implementation of the component. The output of this system is the consistent relations between deep and shallow knowledge. Other elements are input to the system. This system acquires consistent relations in the following manner.

1. First, the system selects the deepest level as the implementation level, and the second deepest level as the specification level.

2. The system receives the implementation and the specification of the component.

3. The system symbolically simulates the behavior of the component both in the deep / implementation level and the shallow / specification level.

   In the knowledge representation scheme, information about the relationship between data is stored in the implementation/specification level knowledge. For example, each equation which represents some aspect of physical principles describes only the name of the relationship between physical data. Interpretation rules supply the actual form of relationship as well as the inference rules at each level, such as th qualitative or the logical simulation rules. With these interpretation rules, the simulation system can calculate the value of the data.

   Initial conditions of the simulation are exhaustively generated for each datum, if not specified [2]. The set of the value is supplied as a part of the interpretation rules.

4. Deep level simulation sometimes fails due to ambiguity arising from a lack of necessary information. For example, ambiguity of the qualitative simulation is widely known. If this happens, the system will generate an assumption to reduce ambiguity, and ask a user for confirmation. User confirmation is needed to reduce the search space of the assumptions [3].

5. After the system finds the correspondence between behaviors of both levels, the EBL-like process generates a generalized relationship. Translation rules are used in two ways. First, they are used to find the correspondence between behaviors of both levels. As there is sometimes a difference between the vocabulary of each level, the translation rule converts the vocabulary to find a correspondence. After the system gets user confirmation, translation rules are also used to perform logically correct transformations.

6. If there is another knowledge level in the component, the acquisition process is continued by returning to step 2 in the deep to shallow manner sequence.

[1] The functionality of this acquisition system was checked through the examples in the next section.

[2] To keep the number of initial conditions low, numerical information is treated using qualitative representation.

[3] Initial values for data can also be regarded as assumptions. The simulation system that is currently used requires the initial value, so the current acquisition system generates assumptions in two steps. This distinction between assumptions in step 3 and 4 is not important.

After the system acquires enough relations between the deep and shallow knowledge, the deepest level knowledge is sufficient to generate the entire hierarchical and consistent knowledge base of a component. The EBL-like process used in the above steps has the following distinguishing characteristics.

1. It is easy for a domain expert to distinguish between important parts of the component. By acquiring the relations between the specification and its implementation from the basic to the complex parts, the expensive chunks which cause degradation are naturally avoided. The knowledge of a complex component is described using the knowledge about the simpler components, each of which is recursively constituted from the further simpler components. Thus, the resulting chunk is simpler than that made directly from knowledge about the simplest components.

   Because of the characteristics of the pattern match that the system must use, a long chunk excessively degrades the system performance. For example, receiving the deep level equation "$A = B$," and the generalized equation in the chunk "$X = Y$", the pattern matcher must check two correspondences: "X=A,Y=B" and "X=B,Y=A". This requirement affects the performance far beyond the linear increase. Thus, a number of short chunks are better than a corresponding single long chunk.

2. The acquisition system can automatically check the necessary environment for the part to be functional, using the information about the approximations. The assumptions which are used in the approximation process correspond to this environment.

3. The explanation of the implementation level behavior is extended to suppress overgeneralization which is introduced by the approximations. In the conventional EBL process, the chunk is made using dependency information on the input data by the resulting data. In our system, in addition to the conventional information, the dependency on data which are related to the assumptions is included in the chunk.

3 Application to Circuit Description

In this section, the knowledge structure of analog and digital circuits and the acquisition process are described as examples to exhibit how the proposed idea actually works.

3.1 Resistor with a Bypass Capacitor (Analog Circuit)

The circuit shown in Figure 5, which is part of a radio circuit [4], is used as a bias stabilizer for temperature compensation. The equations of the deep knowledge describe the underlying physical principle of this circuit. [5]

[4] Application of the proposed scheme to another part of the radio circuit is explained in [14].

[5] Examples of the rules and the equations are reformed to increase readability. Deep/Shallow knowledge in the Figures 5 and 6 is represented in terms of actual relationship among data.

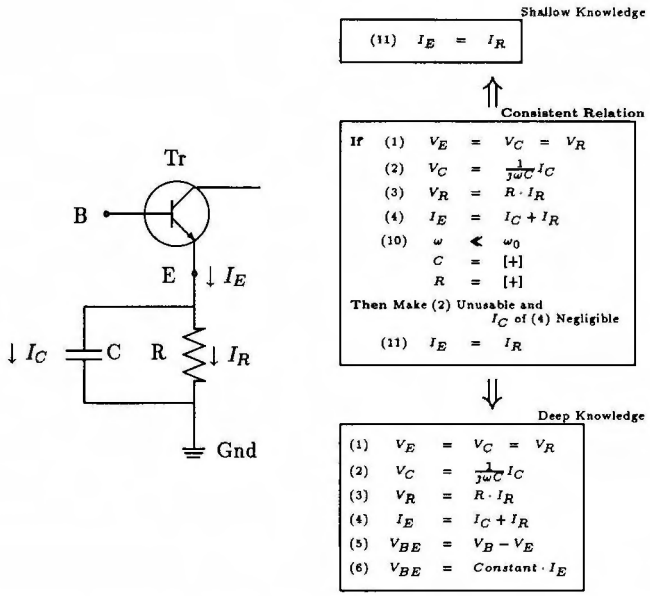


Figure 5: Knowledge Structure of Amplifier

The shallow knowledge describes the representative behavior of a resistor with a bypass capacitor and has no ambiguity. The qualitative simulation using these deep level equations does have ambiguity. The acquisition system finds the relation between both knowledge types by checking the correspondence of behaviors in both levels.

After starting the simulation of behavior in the deep level induced by the increase in I_E, it instantly gets stuck because it does not know how to propagate the change in I_E in equation (4). To suppress ambiguity, assumption "I_C is negligibly small" is needed. With this assumption, the translation rule, shown in Figure 6, can make a new shallow level equation, "$I_E = I_R$", and there remains no ambiguity.

The qualitative simulation using both levels of knowledge has no ambiguity, and negative feedback is detected as shown below.

$$I_E \uparrow \Rightarrow I_R \uparrow \Rightarrow V_R \uparrow \Rightarrow V_E \uparrow \Rightarrow V_C \uparrow \Rightarrow I_C \uparrow \Rightarrow I_R \text{...negative feedback}$$

This process can be simulated using the equations and the interpretation rules. This negative feedback is a necessary behavior for this analog circuit to have bias stability for temperature change. Thus, equation (11) in Figure 5 can be regarded as a specification of this circuit.

Now, after the system finds the correspondence of each behavior, the EBL-like process generates a relation between both levels using the equations which were used in the simulation processes. With this new relation, the system can recognize negative feedback of the behavior of circuits with the same structure. One important point here is that this relation includes information about the simulation of I_C's value, that is, equations (1,2,3,10). This information is included

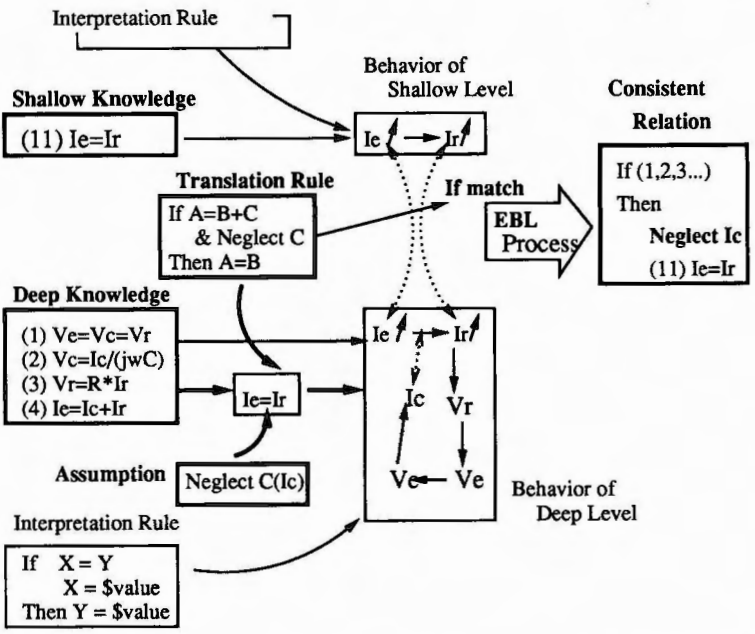


Figure 6: Knowledge Acquisition Process

in the chunk that is the conditional part of the relation shown in Figure 5, because I_C is related to the assumption "I_C is negligibly small". By this information, the system can distinguish this type of circuit from a similar circuit, like an imaginary "resistor with a bypass coil".

3.2 NOR Circuit (Digital Circuit)

Figure 7 shows a simple digital circuit which is made up of three transistors (one pull up and two pull down). The bottom level is a world of analog circuits and its behavior is described by twenty two qualitative equations. The second level is a world of switches and the behavior of this level is described by nine equations. The top level is a world of logic. In the bottom level world, change needs a certain amount of time. In the second and top level worlds, time is represented as a sequence of events. This example has the following two important aspects.

- First, the vocabulary and inferences of each level are completely different. Thus, the system needs appropriate interpretation rules to perform the task in each level. The translation rules are also required to find the correspondence of events which are described using different vocabularies.

- The step by step acquisition scheme is required. The top level logical behaviors require a lot of computing resources, if they are to be computed using the bottom level qualitative relations. However, this system first acquires the

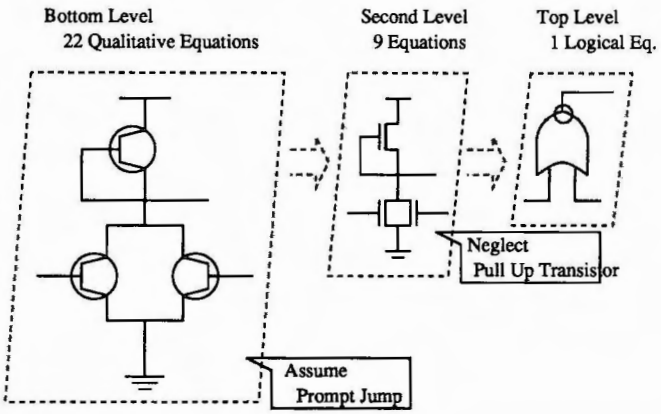


Figure 7: Knowledge Structure of NOR Circuit

relationships between the bottom level knowledge and the second level knowledge, and then the relationships between the second level and the top level so that fewer computing resources are required.

The acquisition process for the relationship between the bottom and the second level knowledge, "A transistor acts as a switch", requires the knowledge about the qualitative simulation that was used in the previous example in Section 3.1. The following second level knowledge and interpretation rules describe the world of the switch (pull down transistor).

Equation switch(base,in,out)
I.Rule
If switch(base,in,out)
and base = [true] at time a
and in = $value at time a
Then out = $value at time b (next time of a).

If No driving input for data
and data = $value at time a
Then data = $value at time b (next time of a).

The following event sequences are made as the simulation results.

at time t0	at time t1
base=[true] in=[true] out=[false]	base=[true] in=[true] out=[true]
base=[false] in=[true] out=[false]	base=[false] in=[true] out=[false]
...	...

The bottom level simulation can be performed using a qualitative reasoning technique. Thus, the interpretation rules are different from those for the second level.

Equation MOS Tr.1(Vb,Ve,Ie)
MOS Tr.2($\dot{V}$e,Ie)

I.Rule Next Ve = Ve + $\dot{V}$e
If MOS Tr.2($\dot{V}$,I)
and I = $value at time a
Then $\dot{V}$ = $value at time a.

The results for the bottom level, as pointed out before, require a certain amount of time for the change.

at time t0	at time t1'	at time t2'
base=[high] in=[high] out=[low]	base=[high] in=[high] out=[middle]	base=[high] in=[high] out=[high]
...	...	...

Assumptions that neglect the state during the change are introduced in the bottom level to find the correspondence between the bottom and second levels. Also, translation rules are used to check the correspondence. For example, the following rule was used to find the correspondence of the voltage and the true/false value.

If A = [high]
Then A = [true]

After the system finds the relationship between the transistor and the switch, it generates the description of other second level switches from the bottom level description. Ambiguity still arises at the second level behavior as to the direction of the change in *Out'* for "Base = [high]". In this case, both the pull up transistor and pull down transistor are closed switches. A new assumption which specifies that the effect of the pull up transistor is negligible is needed to find the correspondence between the second level and the top level behaviors.

3.3 Carry Circuit (Digital Circuit)

Figure 8 shows part of an adder which is a slightly more complicated digital circuit than the NOR gate shown above. It receives three inputs, $\neg K$, $\neg P$ and C_{in} (Carry-in), and produces one output, C_{out} (Carry-out), where K and P are nor and xor of the two one-bit inputs to the adder [15]. The level of the approximation at the bottom level where the behavior is described by forty equations is equal to that at the second level of the NOR circuit. Finding that the effect of the pull up transistors can be neglected at this level, the system generates eight equations at the second level. The second and the top level simulations can be started after assuming the initial values of the above four variables. When the initial conditions are "$K = P$ = [true], C_{in} = [true]/[false] and C_{out} = [false]", the result of the top level simulation gives "C_{out} = [true]", and that of the second level simulation gives "C_{out} = [false]". From this result, the system concludes that it is necessary to precharge C_{out} to ensure the top level specification $C_{out} = \neg K \wedge ((P \wedge C_{in}) \vee \neg P)$. In this case, the initial value of C_{out} is the assumption that is needed to make this carry circuit functional.

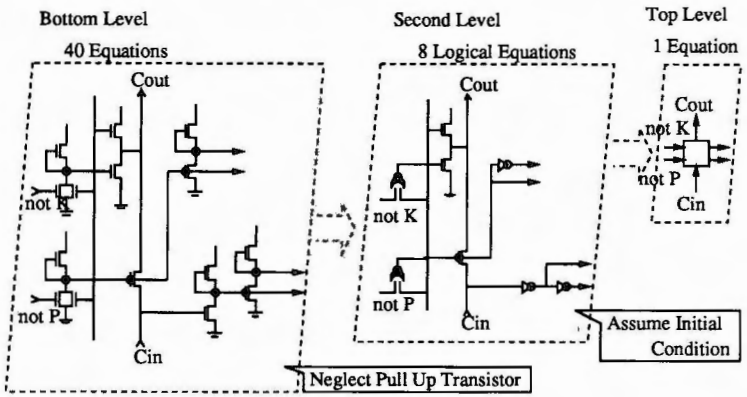


Figure 8: Knowledge Structure of Carry Circuit

In the Figure 8 example, the acquired relation between the top level and second level knowledge includes eight logical equations in the chunk. When this relation is directly learned from the transistor level knowledge, i.e., the bottom level of Figure 7, the chunk includes 112 qualitative equations. As discussed in Section 2.3, this long chunk excessively degrades the system performance. By acquiring the relations between the specification and its implementation, from the basic parts to the complex parts, the proposed knowledge acquisition scheme avoids this expensive chunk. This is a clear example that shows the advantage of this method.

4 Related Works

Prior to this work, the utilization of approximations was widely studied for intractable domains. The work presented in this paper builds on this line of research.

Keller [16] and Ellman [17] improved efficiency of planners and learners by ignoring some search paths in game trees. These works have a close relation to this study. However, these studies did not utilize an explanation structure to check the correctness of learned heuristics for approximations. The learned heuristics must be examined in an empirical manner. Tadepalli [18], Mostow and Prieditis [19] and Unruh and Rosenbloom [20] also aimed at improving the efficiency of planners and learners in game domains by approximations. But, their works can not be directly used in the construction of a hierarchical knowledge base.

Bennett [7] used approximations to simplify mathematical domain theory. His approach utilized numerical information to ensure the correctness of the approximate solution. The present study generalizes approximation type from the numerical one to a more general form. Doyle [6] and Mozetic [8] also used approximations to simplify domain theory. Their domains are intractable, but have relatively simple structures. Their prespecified inference engine was enough to handle the domain theory, without the additional information which is required in our study as interpretation rules.

Falkenhainer and Forbus [9] investigated multigrain, multislice models to rep-

resent a large system. In their work, the user described hierarchy of the system to dramatically reduce the complexity of the qualitative reasoning task. But, the user had to carefully design the whole hierarchical structure to achieve the consistency of the whole hierarchy. The currently proposed method can be used to construct this type of knowledge base more easily.

5 Conclusions and Future Work

A hierarchical knowledge representation which enables acquisition and simultaneous utilization of knowledge expressed in multiple levels with different approximation has been proposed. The characteristics of the proposed representation scheme can be summarized as follows.

1. Information about approximations which are used in each hierarchy are memorized in this hierarchical knowledge base. Without this information, contradictions which are introduced by the approximation between each level decrease the functionality of the knowledge base.

2. The multiple domain theory and the relation between descriptions of different levels are treated to express many aspects of the complex system.

3. A new method which is an extension of the existing explanation-based learning method is proposed to support construction of the consistent hierarchical knowledge base complying with the proposed representation scheme.

The examples which are given in the domain of analog/digital circuits show the expressiveness of the proposed representation and the functionality of the knowledge base construction method.

Despite the progress in the representation scheme and the construction method of the consistent hierarchical knowledge representation, a number of issues remain to be addressed. The knowledge compilation method for the diagnostic system with this type of hierarchical knowledge base should be reconsidered to utilize the hierarchical/approximated nature of the system's knowledge. To realize a top down refinement design system with approximated domain model, an efficient method to negotiate contradictions should be developed.

Acknowledgment

The authors are deeply indebted to Dr. Masami Yamasaki and Dr. Masaki Suwa of ARL, Hitachi Ltd., for their useful discussion on qualitative reasoning and suggestions for the explanation-based learning method. They also extend their appreciation to Dr. Eiichi Maruyama of ARL, Hitachi Ltd., for his encouragement throughout this work.

References

[1] Johan de Kleer. How Circuits Work. *Artificial Intelligence*, 24:205–280, 1984.

[2] Brian C. Williams. Qualitative Analysis of MOS Circuits. *Artificial Intelligence*, 24:281–346, 1984.

[3] Benjamin Kuipers. Qualitative Simulation. *Artificial Intelligence*, 29:289–338, 1986.

[4] Ramesh S. Patil, Peter Szolovits, and William B. Schwartz. Causal Understanding of Patient Illness in Medical Diagnosis. In *IJCAI-81*, pages 893–899, 1981.

[5] Tom Bylander and B. Chandrasekaran. Understanding Behavior Using Consolidation. In *IJCAI-85*, pages 450–454, 1985.

[6] Richard J. Doyle. Constructing and Refining Causal Explanations from an Inconsistent Domain Theory. In *AAAI-86*, pages 538–544, 1986.

[7] Scott W. Bennett. Approximation in Mathematical Domains. In *IJCAI-87*, pages 239–241, 1987.

[8] Igor Mozetic. The Role of Abstractions in Learning Qualitative Models. In *Fourth International Workshop on MACHINE LEARNING*, pages 242–255, 1987.

[9] Brian Falkenhainer and Kenneth D. Forbus. Setting up Large-Scale Qualitative Models. In *AAAI-88*, pages 301–306, 1988.

[10] V. Sembugamoorthy and B. Chandrasekaran. Functional Representation of Devices and Compilation of Diagnostic Problem-Solving Systems. In J.Kolodner and C.Reisbeck, editors, *Experience, Memory, and Reasoning*, pages 47–73, Lawrence Erlbaum Associates publishers, 1986.

[11] Louis I. Steinberg. Design as Refinement Plus Constraint Propagation: The VEXED Experience. In *Proc. of AAAI-87*, pages 830–835, 1987.

[12] Tom M. Mitchell, Richard M. Keller, and Smadar T. Kedar-Cabelli. Explanation-Based Generalization : A Unifying View. *Machine Learning*, 47–80, 1986.

[13] Gerald DeJong and Raymond Mooney. Explanation-Based Learning : An Alternative View. *Machine Learning*, 145–176, 1986.

[14] K. Yoshida and H. Motoda. An Approach to Hierarchical Qualitative Reasoning. *Japanese Society for Artificial Intelligence (in Japanese)*, 4(4), 1989.

[15] C. Mead and L.Conway. *Introduction to VLSI Systems.* Addison-Wesley Publishing Company, 1980.

[16] Richard M. Keller. Concept Learning in Context. In *Machine Learning-87*, pages 91–102, 1987.

[17] Thomas Ellman. Approximate Theory Formation : An Explanation-Based Approach. In *AAAI-88*, pages 570–574, 1988.

[18] Prasad Tadepalli. Lazy Explanation-Based Learning : A Solution to the Intractable Theory Problem. In *AAAI-89*, pages 694–700, 1989.

[19] Jack Mostow and Armand E. Prieditis. Discovering Admissible Heuristics by Abstracting and Optimizing : A Transformational Approach. In *AAAI-89*, pages 701–707, 1989.

[20] Amy Unruh and Paul S. Rosenbloom. Abstraction in Problem Solving and Learning. In *AAAI-89*, pages 681–687, 1989.

A Knowledge Acquisition Method for Distributed Information Systems: A Multi-DataBase Approach

Ahmad A. Barfoursh M.H.Rogers

Department of Computer Science,
University of Bristol,
Bristol BS8 1TR.
England

Abstract

It is becoming increasingly common for the users of information systems to formulate queries which can only be answered by accessing a number of different databases, either locally or remotely. Advances in networking capabilities mean that access to remote databases is generally a straightforward matter but less attention has been paid to the problems of generating appropriate requests, receiving and collecting the responses in a convenient form for the user.

Several tasks are important in the design and implementation of a Multi-DataBase Enquiry Service (MDB-ES). One of the crucial stages in the development of the MDB-ES is the extraction of the relevant data from the responses that the system has obtained after remote communication with selected databases and correctly combining the partial results from individual queries into a single answer. Obviously we cannot assume that the remote systems offer the user relevant items only.

In the present work we are considering the problem of designing a Knowledge Acquisition System*(KAS)* which elicits, analyses and translates the responses which have been obtained from remote databases.

One of the major problems in this area concerns missing information and the inability to cope with this satisfactorily has been one of the causes of the lack of success of some Information Systems currently available. Particular attention has been paid to this problem in the work presented here, both in determining what information is missing and how this can be handled.

1 Introduction

Improved networking capabilities allows access to several databases, in particular to databases that provide large amount of information about similar topics. Many organisations have a pressing need to access data rapidly and reliably. This need is very difficult to satisfy, when the data is stored in many independent databases.
One of the crucial stages in the development of a Multi-DataBase Enquiry Service "MDB-ES" is the extraction of the relevant data from the responses that the system has obtained after remote communication with selected databases, and correctly combining the partial results from individual queries into single answers [1].
In this context Information Retrieval *"IR"*, which is traditionally involved in library organisation seems to be a good test bed [2]. IR is defined as a method and procedure for recovering specific information from stored data [3]. In recent years the number of online retrieval systems has increased dramatically, each system with its own characteristic As more and more information is stored electronically, the need for intelligent methods of accessing this information became increasingly critical.

The study of the effectiveness of the retrieval systems by Blair and Maron [4] indicates that only 20% of the relevant items were found by careful searchers who used commercially available full-text retrieval systems. Dissatisfaction with the current state of affairs and the increasing awareness of the importance of IR as an application area are the main factors in the recent upsurge of interest in intelligent IR [5]. In order to improve retrieval effectiveness and because of the realisation of the need to incorporate and use knowledge within IR systems, researchers have worked on automatic indexing and retrieval systems, which also aim to incorporate and use knowledge; this use is termed Intelligent Information Retrieval *"IIR"*, and is a particular class of Artificial Intelligence *"AI"*.

Sparck Jones [6] describes an IIR system as a system with a knowledge base and inferential capabilities which can be used to establish connections between a request and a set of documents. Access to one or more databases of documents is also recognised as a desirable feature for any IIR [7]. IR systems offer excellent opportunities for research in AI. Previous work on AI techniques for IR systems includes different attempts to integrate several techniques in more or less sophisticated systems, some of the most important approaches can be found in [8, 9, 10], and the references given in [11, 7]. It is clear that most of the previous work related to the incorporation of AI techniques into IR mainly concerns Natural Language Processing *"NLP"*, Knowledge Representation and reasoning.

In this work we concentrate on Knowledge Acquisition which is an important task in building expert systems [12, 13]. The task of Knowledge Acquisition has long been cited as the *"bottleneck"* which slows down the expert systems development process [14]. Knowledge Acquisition has been defined by Waterman [15] as "the process of extracting, structuring and organising knowledge from several sources, usually human experts, so it can be used in a program".

Current methods for transferring knowledge into a knowledge base are time consuming, expensive and inconsistent with the system development process [16]. In the MDB-ES we consider an automated knowledge acquisition method, based on experiences in this field which have been used and investigate the automated knowledge acquisition methods as described in [17], [18], [19]. This method allows the user to add new knowledge to the knowledge base and it is also able to obtain, evaluate and prepare knowledge from the databases to the network to add to the knowledge base.

2 The Knowledge Acquisition System *"KAS"*

Knowledge acquisition is an important task in building intelligent systems [12, 13]. In the MDB-ES we consider an automated knowledge acquisition method which is based on earlier work in this field which has used knowledge acquisition software [17]. This software may allow the user to add new knowledge to the knowledge base and it should also be able to obtain, evaluate and prepare knowledge from the network's databases to add to the knowledge base. The aim of this system is the transformation of the *facts* into a form suitable for representation. Figure 1, illustrates the components of a knowledge acquisition system which performs this task.

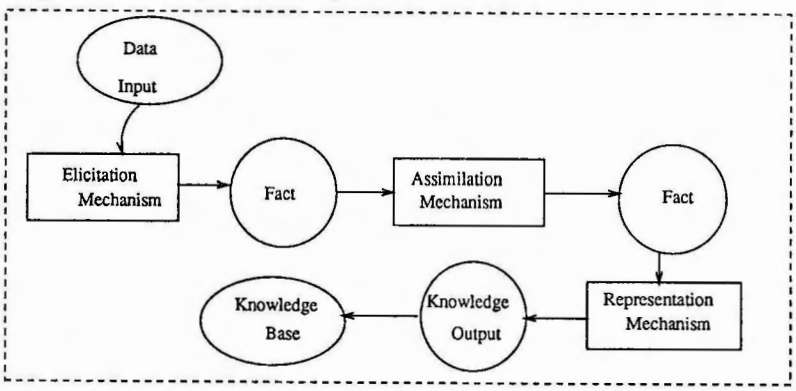


Figure 1: Knowledge Acquisition components.

The system includes an *elicitation* mechanism which contains a natural language processing capability. The *elicitation* mechanism is used to extract facts from the responses of the various sources (databases).

These facts are then passed to the *assimilation* mechanism which is capable of handling missing and incomplete information. The *assimilation* mechanism is used to transform these facts in accordance with the user's intentions .
The *representation* mechanism performs a further transformation of these facts into a suitable machine representation.

The prototype model of the MDB-ES stores knowledge in the knowledge base mainly as a set of *facts*, but the knowledge base must also be able to accommodate knowledge which is represented by *rules*.

2.1 Elicitation Mechanism

The elicitation of data within a particular knowledge domain is one of the main problems in building knowledge bases; very little work has actually been done on this issue. Most works in this area is more concerned with how the knowledge is actually represented in some form of knowledge base, rather then how it was elicited.
Some take this problem one step further [20], stating that: "In short, apparently little or no systematic research has been conducted on the question of how to elicit the expert's knowledge and inference strategies".

Since we are accessing different databases and as each database may use a different format to answer the same query, there needs to be some method of recognising the necessary items for eliciting them from the source.

To illustrate the problem we are addressing, let us look at Figure 2, from the University of Bristol on-line library enquiry service, which shows some output in different formats.

```
TITLE ENQUIRY

                                    Full Display
    Salton, Gerard
       Automatic information organization and retrieval / by Gerard Salton.
       New York : McGraw-Hill, 1968. - xiii,514p : ill,forms ; 24cm. -
       (McGraw-Hill computer science series)
       bibl p485-498. - 0-07-054485-9
```

```
SUBJECT KEYWORD ENQUIRY        Brief Display        records 1 to 5 (of 144)
"automatic information"
-------------------------------------------------------------------------
1  Salton, Gerard.  Automatic information organization and retrieval / by
   Gerard Salton.  1968

2  Martin, John.  Information - automatic operations / prepared for the
   [Systems Management] Course Team.  1974
.
.
.
```

Figure 2: Outputs in different formats from the University of Bristol on-line library service.

As can be seen from this figure two different formats have been used for displaying data, namely, for subject enquiry and title enquiry. The main goal is to produce facts from these responses in the knowledge domain specified by the user enquiry. The sequences illustrated in Figure 3, display the general model of the elicitation mechanism. Particular applications will require the use of particular components in both the tokenisation and recognition mechanisms.

In processing both programming languages and natural languages, it is necessary to separate sentences(sequences of characters) into words (*Tokenisation*), and then to apply a set of rules for recognising valid string sequences (*parsing*).

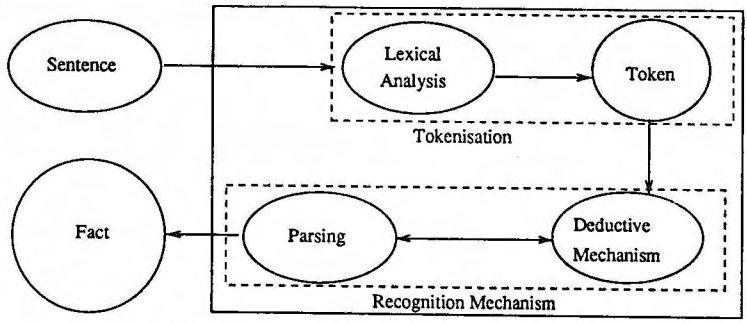


Figure 3: Sequence of Elicitation mechanism.

2.1.1 Tokenisation

To find the validity of responses, the input, which are sentences in a certain way, must be analysed; and it is much easier for logic grammars to analyse the lists of words rather than sentences. So what we need is a *tokeniser*, which converts natural input in character string form, into a list of words (tokens) which can then be analysed.

For example:
Salton, Gerard Automatic information organization and retrieval / by Gerard Salton. New York : McGraw-Hill, 1968. - xiii,514 p : ill,forms ; 24 cm. - McGraw-Hill computer science series bibl p485-498. - 0-07-054485-9.
which may be converted to:
[salton,gerard,automatic,information,organization,and,retrieval,/,by,gerard,salton., new,york,:,mcgraw-hill,,1968.,-,xiii,514,p,:,ill,forms,(;),24,cm.,-,mcgraw-hill, computer,science,series,bibl,p485-498.,-,0-07-054485-9]

This process includes a lexical analysis of the text. The task of a lexical analysis is to specify how characters are grouped into meaningful tokens for syntactic analysis, while at the same time removing characters which do not contribute to the meaning of the sentence. This level covers operations such as common word deletion, dictionary processing of individual words and the replacement of words by synonyms.

2.1.2 Recognition Mechanism

It must be possible to recognise the knowledge from the result file in order to make the MDB-ES useful for the end user. For this purpose a recognition system has been designed as part of the knowledge acquisition system. Figure 4, gives an overview of the organisation of the prototype system.

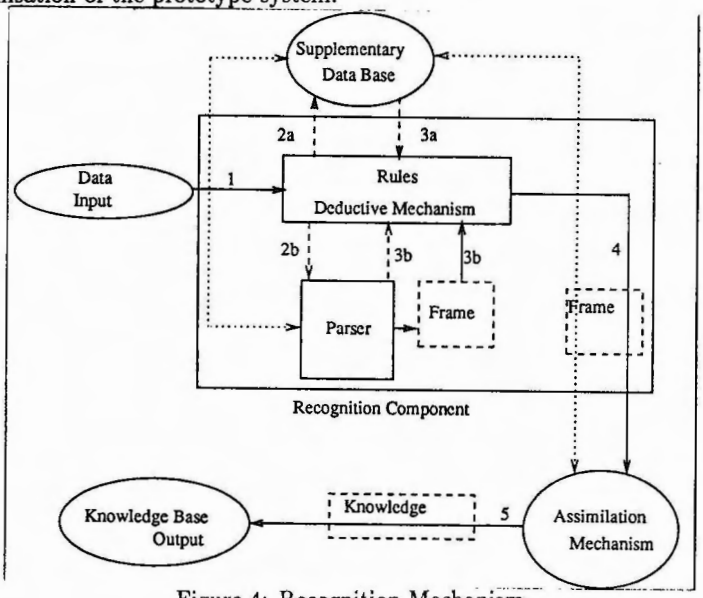


Figure 4: Recognition Mechanism.

The *deductive mechanism* supervises the processing of each data element in the recognition mechanism. Data element are the responses from the participant systems which are collected in the file termed *result file.*

Data from the tokenisation component indicated by path number "1" in the Figure 4. The *deductive mechanism* first identifies the recognition level, as shown in Figure 4, and passes the data element, which at this stage is of the form :

$$sentence([CC, T, A, P, L, D, S, I])$$

where each element of the list $[CC, T, A, P, L, D, S, I]$ contains information about the

user query, to either the *parser* or *supplementary* database indicated by path numbers "2a"and "2b".

The parser accepts the data element from the *deductive mechanism* and analyses this to determine whether or not acceptable data have been received from the result file. If the input is, the parser then returns a set of predicates as a frame to the *deductive mechanism* as shown by path number "3a" in Figure 4. The results of these deductions are then passed to the rules for consistency checking: "*Assimilation* and *Representation* mechanisms". The result of this checking, in the form of predicates which contain knowledge, will be passed to the knowledge base, indicated by path numbers "4 and 5".

In the event of an unacceptable input sentence, the *deductive mechanism* uses the rules either to ignore the input and get a new sentence or to generate an appropriate message for the user.

All the rules are coded in the form of production rules which allow procedure calls on their right hand-side.

2.1.3 Frame definition and recognition of slots

In the context of Distributed Information Systems we observed two major problems which exist in representing knowledge in the knowledge base:
First, the differences in database formats and second, inconsistencies in the detailed information associated with them.

In order to tackle these problems, it is necessary to define a common format which contains common items. For example, in implementation of the prototype system, the necessary items in each source after recognition is represented by the predicates *(ref)* defined as :
ref(< *title* >, < *author* >, < *doc* >, < *pub* >, < *place* >, < *date* >, < *source_id* >)

< *title* > is a list of terms representing the title of the document.

< *author* > is a list of terms representing the author/s or editor of the document.

< *doc* > is a list of terms representing the kind of the document (proceedings, book, journal, etc).

< *pub* > is a list of terms representing the name of the publisher.

< *place* > is a list of terms representing the place of the publication.

< *date* > is a constant that identifies the month/year of the publication.

< *source_id* > is a list of terms representing the name of the source (database).

We used a declarative method, *frames*, for representing the knowledge. They contain a number of slots for different concepts, slots which hold data giving the particular values for the attributes. For example, a slot representing the concept "place" may contain the data "London".

An important feature of the frame is that certain slots may be filled by default. Even if there are no empty slots, it may be advantageous to infer relevant facts. For example suppose we wish to find the place of publication for a book published by"taylor graham" which is known to be "London", we can get this result from the Supplementary-database.

Another advantage of the frame is the use of a common format. There are different formats among the heterogeneous databases because of a lack of standards and autonomous database development and administration. For example, dates can be constrained in a database to be either "sep, 1, 1989" or "1, sep,89". These formats could therefore be standardised by the use of frames and correspondence rules.
The information contained in the system such as Supplementary-databases is used to clarify such question as:

1. What rules and techniques the system should employ in order to extract information from sources for the frame?
2. What are the knowledge domains of the slots in the frame?
3. What kinds of rules about the knowledge are necessary for the systems to determine what information is missing?
4. What rules and techniques the system should employ in order to remove inconsistency?

In order to recognise the slots, the rules (*deductive mechanisms)* explore the recognition network. Figure 5, illustrates the recognition network, which is divided into four levels:

Level-1 identifies which remote database is being consulted.

Level-2 identifies the particular query which is being made: for example, in a library enquiry service, the type of search (author,title,etc.) is identified in this level.

Level-3 extracts the detailed information for the query.

Level-4 checks for the end of the database, and if necessary, causes a new remote database to be consulted, returning to level-1.

We have studied these recognition levels from both syntactic and semantic points of view in a particular library enquiry system (University of Bristol).
At the syntactic level, we consider just the layout using the clues provided by the positions of the various features, the size of spaces, the use of punctuation, numbers, etc.
At the semantic level, words and phrases like"by" and "edited by" could be used to identify the names of author and editors.

2.1.4 Deductive Mechanism

The *deductive mechanism* uses an algorithm which systematically traverses the recognition network shown in Figure 5. There is a set of rules embodying the deductive rules and a method of rule selection. The algorithm has been implemented in Prolog, so that for example, a rule in the deductive mechanism could be:

```
layout('B'):-string('A').
```

which can be read as "If string is 'A' THEN layout is recognised as 'B' ", where 'B' is stored as a fact in the Supplementary-database.

Algorithm:
The steps involved in the algorithm are:

1. Read string (S) from previous system (as a list of words),
   which is in the form of a Prolog fact.

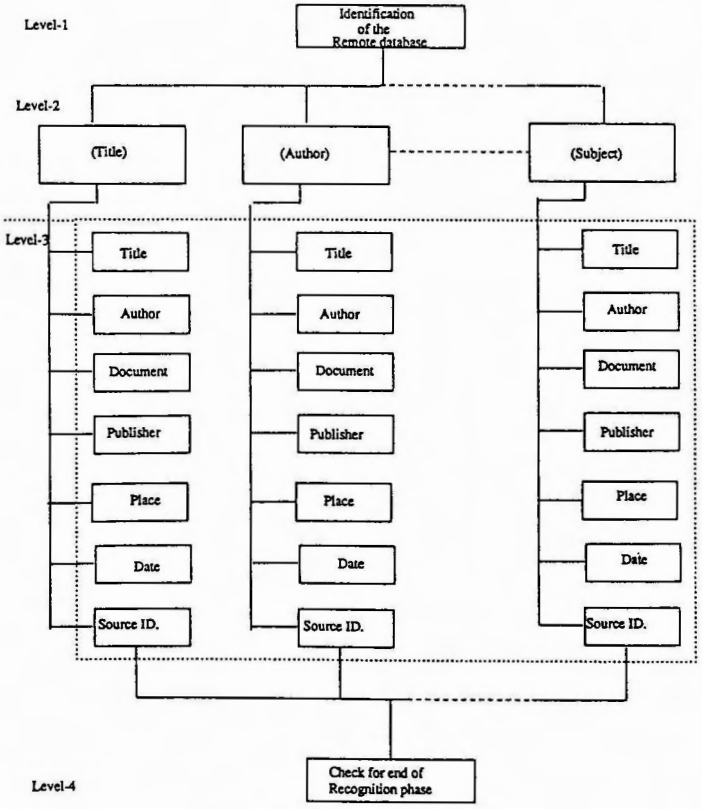


Figure 5: Recognition Network.

2. Check the string (S) with the rules (right hand-side).
3. Selection of a candidate rule.
4. Application of the left side of the selected rule (action).
5. Go to step 1 OR
   Continue.

A brief description of the above algorithm now follows:

Step 1 simply receives sentences(strings as list of words) from the tokenisation and passes them to the deductive mechanism(step 2).

Step 2 consists of the evaluation of the right-hand side of every rule contained in the deductive mechanism, and if right side of the rule is true the algorithm progresses to step 3.

Step 3 is very crucial in the performance of the *deductive mechanism.* A good choice is limiting further backtracking steps corresponding to many misleading attempts. Some control rules may be used to control the activity of the *deductive mechanism* in order to avoid unnecessary attempts such as backtracking.

Step 4 simply performs the actions defined on the left-hand side of the selected rule. Three cases have to be considered, corresponding to the level of recognition as follows:

First, if the recognition of the layout level-1 is not still clear, we then have to stay in the loop and consider another input (the level of recognition is updated in the Supplementary-database).

Second, recognition of layout level-2 is possible through rules using data available in the Supplementary-database.

Third, recognition of the slots in level-3 is achieved by the parser. Finally recognition of the end layout level-4 is possible by using data available in the Supplementary-database.

Step 5 At this stage, control is returned to level-1 to investigate whether or not there are further remote databases to consult; alternatively control passes to the next stage which is parsing.

2.1.5 Parser

The parser accepts a data element from the deductive mechanism and analyses it to determine if any of available data are acceptable. For example in the library enquiry service the data elements

sentence([*The*, *classmark*, *can*, *be*, *keyed*, *in*, *as*, *fully*, *as*, *required*, *provided*, *it*, *has*, *a*, *minimum*, *of*, *one*, *to*, *three*, *letters*, *followed*, *by*, *a*, *number*, *or*, *numbers*])

obtained from the remote database will be ignored by the parser because they do not contain any useful information. The parser searches for a complete input using a top-down parsing algorithm with backtracking, such as given in Clocksin and Mellish [21]. Various output structures that are frequently used in databases were defined to produce the grammar, and the parser tries to match the input sentence with the grammar. The parser returns the frame to the deductive mechanism for further processes such as updating the Supplementary-database and transfer to the assimilation mechanism.

2.1.6 Supplementary-database

This database contains all the information relating to the current state of the system. The complete set of variables, including the initial queries, characteristics and variables for the recognition of layouts in various levels is stored in this database. The database also contains the current state of recognition which is updated in the event of a change.

The data in the Supplementary-database are presented as predicates such as *p(a)* or predicates with two arguments such as *r(a,b)*.

where *p(a)*, means that the entity *a* has property or attribute *p*.

For example:

layout_level_1('B').

means that: 'B' is the string for recognition of layout_level_1, and:

current_state(layout_level_1,recognition_level).

means that:

" *layout_level_1* is the *recognition_level* in the *current_state* "

This will be updated to reflect the current recognition level. Generally, data stored in the Supplementary-database can be categorised as *facts*, and *integrity constraints* which take the form of facts and rules, for use by the assimilation mechanism.

2.2 Assimilation Mechanism

The role of the *assimilation mechanism* is to create the knowledge base which is then available for future use. The facts which have been provided by the *elicitation mechanism* still need to be refined before being stored in the knowledge base. To assure the validity of the knowledge stored in the database, knowledge assimilation is an essential part of the knowledge acquisition process.

Our goal is to judge whether or not input data should be added to the database. In this context the *assimilation mechanism* examines incoming data to see if it is consistent with, or can be deduced from, the current knowledge base. If the input data is not contained within the knowledge base then it is added to the knowledge base at this time. One of the major problems at this stage concerns missing and incomplete information and we now examine this matter in more detail. We use the term *input data* to mean a fact obtained from the elicitation mechanism which is used as input to the assimilation mechanism.

2.2.1 Processing sequence of knowledge assimilation

The basic concept and processing sequence of the knowledge assimilation in the MDB-ES are summarised below:

Existence checking :

IF, input data exists in the current knowledge base, **THEN**, the new knowledge base is identical with the current one.
Let us suppose that the knowledge such as:
$ref(til([automatic, information, organization]),$
$aut([gerard, salton]), doc([book]), pub([mcgraw - hill]),$
$plc([new, york]), dat([1968]), sid([bristol])).$
exists in the current knowledge base as a result of the previous process. The results of the *existence checking* should show that the knowledge base contains the same information, so this information will not be added to the knowledge base, in order to protect the knowledge base from duplication.

Correctness checking :

IF, the information in the input data differs from identical knowledge in the current knowledge base or is different from the integrity constraints **THEN** the results of the *correctness checking* will show which of these two items should be included in the new knowledge base. In this context, the mechanism must be aware of the definition of the identical knowledge and integrity constraints.
In our prototype system, we assume:
Two items are identical when *the title AND the author/s* are identical. The data for the definition of the *integrity constrains* are stored in the Supplementary-database in the form of facts and rules.
Suppose the incoming data is :
$ref(til([automatic, information, organization]),$
$aut([gerard, salton]), doc([book]), pub([mcgraw - hill]),$
$plc([los - angeles]), dat([1968]), sid([bristol])).$
this contains incorrect information which needs to be corrected, because by referring to the Supplementary-database, no branch in 'Los Angeles' has been assigned to 'McGraw-Hill' publication in the Supplementary-database.

In some different cases we may need to ask for expert judgement. Suppose the knowledge such as:

$ref(til([automatic, information, organization]),$
$aut([gerard, salton]), doc([book]), pub([mcgraw - hill]),$
$plc([new, york]), dat([1968]), sid([bristol])).$

is presented in current knowledge base and the incoming data element is:

$ref(til([automatic, information, organization]),$
$aut([gerard, salton]), doc([book]), pub([mcgraw - hill]),$
$plc([new, york]), dat([1983]), sid([bristol])).$

which is recognised as identical to the current one, but they have a different *date* of publication, and no information is available in the Supplementary-database which could verify the correct date.

Completeness checking :

IF, the input data contains missing information **THEN** it needs to be completed before it is added to the new knowledge base by reference to the integrity constraints and the current knowledge base.

Suppose the input data element is:

$ref(til([access, to, academic, networks]), aut([patrick, holligan]),$
$pub([taylor, graham]), plc([]), dat([1986]), sid([bristol])).$

with missing information "$plc([])$". This data element could be completed as "$plc([london])$" using facts in the Supplementary-database which make it clear that the "Taylor Graham publication company is located in London".

Another task which is provided by *completeness checking* is to complete the source slot in the current knowledge base. Suppose the input data element is:

$ref(til([access, to, academic, networks]), aut([patrick, holligan]),$
$pub([taylor, graham]), plc([london]), dat([1986]), sid([bristol])).$

and identical input data such as:

$ref(til([access, to, academic, networks]), aut([patrick, holligan]),$
$pub([taylor, graham]), plc([london]), dat([1986]), sid([bath])).$

is already exists in current knowledge base. The results of the *completeness checking* show that the *"source list"* in the new knowledge base is completed as:

"$sid([bristol, bath])$.

The *assimilation algorithm* thus eliminates redundancy and protects the knowledge base against inconsistencies.

2.2.2 The Assimilation Algorithm

In this section, we first introduce various definitions which will be needed throughout our study, and then we discussed different cases in the context of the knowledge assimilation mechanism.

An *input data* element "I", is a set of atoms in the form of a predicate.

The *integrity constraints* "ICs", are organised as rules and facts which ensure that input data are correct and complete.

A *completed input data* element"CI", is a set of atoms in the form of a predicate, which satisfies the "ICs".

An *incomplete input data* element "NI", is a set of atoms in the form of a predicate, which does not satisfy the "ICs".

A *knowledge* "K", is a set of atoms in the form of a predicate.

A *complete existence knowledge* "CK", is a set of atoms in the form of a predicate, which fully satisfies the "ICs" and exists in the current knowledge base.

An *incomplete existence knowledge*"NK", is a set of atoms in the form of a predicate, which does not fully satisfy the "ICs", but which exists in the current knowledge base.

An *input source* "IS", is an atom which contains information about source of input data.

A *knowledge source* "KS", is an atom which contains information about source of existence knowledge.

A *current knowledge base* "CDB", is a set of facts in the form of a predicate, before, input data has been added.

A *new knowledge base* "NDB", is the set of facts in the form of predicates, after, input data has been added.

Different cases are considered in the knowledge assimilation mechanism as follows:

1. if: $(CI = CK) \wedge (IS \in KS)$
   then: $NDB := CDB.$

2. if: $(CI = CK) \wedge (IS \notin KS)$
   then: $KS := KS + IS,$
   $NDB := CDB + KS.$

3. if: $(CI \neq CK) \wedge (IS \in KS)$
   then: $NDB := CDB.$

4. if: $(CI \neq CK) \wedge (IS \notin KS)$
   then: $KS := KS + IS,$
   $NDB := CDB + KS.$

5. if: $(CI \wedge NK) \wedge (IS \in KS)$
   then: $CK := NK + CI,$
   $NDB := (CDB - NK) + CK.$

6. if: $(CI \wedge NK) \wedge (IS \notin KS)$
   then: $KS := KS + IS,$
   $CK := NK + CI,$
   $NDB := (CDB - NK) + CK.$

7. if: $(NI \wedge CK) \wedge (IS \in KS)$
   then: $NDB := CDB.$

8. if: $(NI \wedge CK) \wedge (IS \notin KS)$
   then: $KS := KS + IS,$
   $NDB := CDB + KS$

9. if: $(NI \wedge NK) \wedge (IS \in KS)$
then: $K := NI + NK,$
$NDB := (CDB - NK) + K.$

10. if: $(NI \wedge NK) \wedge (IS \notin KS)$
then: $KS := KS + IS,$
$K := NI + NK,$
$NDB := (CDB - NK) + K.$

2.3 Representation Mechanism

In order to use the knowledge base, we must choose an appropriate representation. This problem of representation is the central issue in any programming task, and concerns the method to be used for representing the required information in data structures that facilitate efficient searches for specific items of information to be processed efficiently.

Knowledge bases may contain knowledge of many different kinds. Figure 6 illustrates the kinds of knowledge that can go into a knowledge base in the Knowledge Representation Mechanism. The representation and reasoning facilities must be able

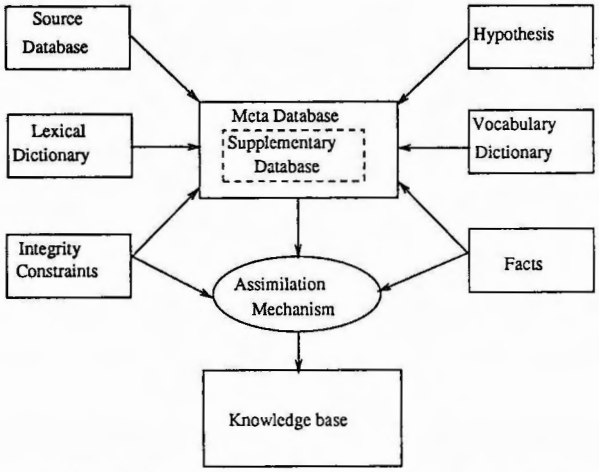


Figure 6: Representation Mechanism and sources of knowledge in the knowledge base.

to integrate different kinds of knowledge into a coherent knowledge base which supports the user requirements. It should also make it easy for us to add, change and delete knowledge. The knowledge in the knowledge base in the MDB-ES can be represented by classifying it into two forms: *facts* and *rules*.

2.3.1 Facts

The ***facts*** are presented as predicates based on frames. A frame provides a structured representation for use in a knowledge base. items(slots) which can be related to other frames. The frame hierarchy is shown in Figure 7.

In order to construct knowledge, a *"bottom-up"* process is used which allows access to particular results in the data structure during the computation.

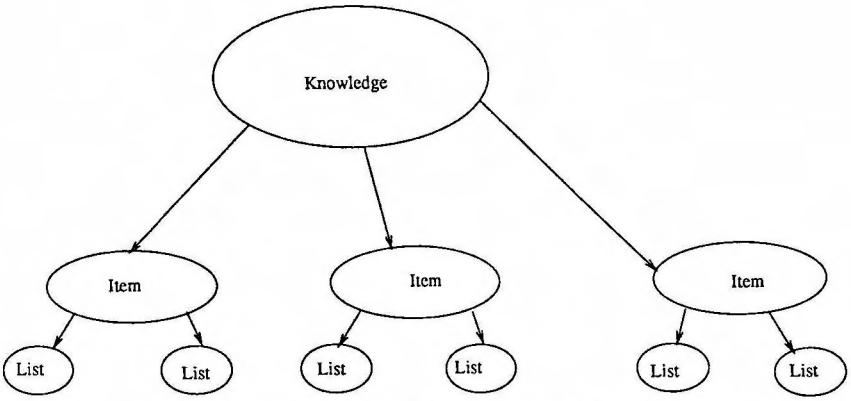


Figure 7: Type hierarchy of the frame-based knowledge.

Consider a record of the knowledge base as follows:

$ref(til([automatic, information, organization]),$
$aut([gerard, salton]), doc([book]), pub([mcgraw - hill]),$
$plc([new, york]), dat([1968]), sid([bristol, bath])).$

In a *"bottom-up"* process we are first presented extracting facts (result of elicitation mechanism) as a predicate which contains two arguments in the form of

$item(< context >, < id >)$

where:

$< context >$ is a list of elments representing the context of the item.
$< id >$ is a constant that identifies the source/s of the item.
In the above example, the possible facts in the early stage of process could be shown as:

$til([automatic, information, organization], 1).aut([gerard, salton], 1).$
$doc([book], 1).$
$pub([mcgraw - hill], 1).$
$plc([new, york], 1).$
$dat([1968], 1).$
$sid([bristol], 1).$
AND:
$til([automatic, information, organization], 2).$
$aut([gerard, salton], 2).$
$doc([book], 2).$
$pub([mcgraw - hill], 2).$
$plc([new, york], 2).$
$dat([1968], 2).$
$sid([bath], 2).$

We call these facts *"uncertain facts"*, but after passing these facts through the assimilation mechanism and combining them together, the final data structure is knowledge which could be stored in the knowledge base.

3 Presentation of the Results

The responses from each participant database system after consistency, correctness and completeness checking, using knowledge assimilation mechanism, are combined into a single knowledge base.

The presentation of the final result to the user, which is the response to a particular query, takes place after the transfer of the appropriate knowledge from the knowledge base to the User Front-End component. In the first case, the system has been asked to perform a subject search for *"Music"* using the University of Bristol on-line library system. Part of the result file in this case is shown in Figure 8. The final display to the user in this case which has been filtered by the knowledge Acquisition System is illustrated in Figure 9.

In the second case, if we have the following output, as a result file for the same query, from the University of Bath on-line system then we will be able to investigate the actions of the Knowledge Acquisition System.

"^M2b PARRY, W.H. 13 CENTURIES OF ENGLISH CHURCH MUSIC. 2nd ed. 1946"

"^M"

"^M3b LONGYEAR, R.M. 19TH. CENTURY ROMANTICISM IN MUSIC. 1969"

"^M"

"^M1 Abraham, Gerald. 100 YEARS OF MUSIC. 4th ed. London 1974"

"^M"

As can be seen this file contains more information, such as place of publication for third item, then the previous one from the University of Bristol, with different syntax.

Figure 10, shows the result of the prototype system in the case which different databases with same data have been consulted.

All the responses from the remote databases are stored in the single permanent knowledge base which could be used in future. Figure 11, illustrates the records *"ref"* which are produced as a result of consulting different databases.

```
Welcome to the University of Bristol Library Computer
*********START BRISTOL*********

        The Libertas system holds information about books and serials in all
branches of the University Library, but it is not a complete catalogue.
Many books acquired before 1978 (especially by branch libraries) are not
present.


SUBJECT KEYWORD ENQUIRY

This enquiry looks for TITLES or SUBJECT HEADINGS containing as many
as possible of the words you enter

--------------------------------------------------------------------------------
-
****here****
1  Abraham, Gerald.  100 YEARS OF MUSIC.  4th ed.  1974

2  PARRY, W.H.  13 CENTURIES OF ENGLISH CHURCH MUSIC.  2nd ed.  1946

3  LONGYEAR, R.M.  19TH. CENTURY ROMANTICISM IN MUSIC.  1969

^M
end
*********END BRISTOL*********
```

Figure 8: Part of the out-put for subject searching at the University of Bristol on-line library system.

```
*********************************
Title=        [100 YEARS OF MUSIC.]
Auther=       [Abraham Gerald.]
Source=
Publisher=
Place=
Date=         [1974]
On_line=      [Bristol]
*********************************
*********************************
Title=        [13 CENTURIES OF ENGLISH CHURCH MUSIC. 2nd ed.]
Auther=       [PARRY W.H.]
Source=
Publisher=
Place=
Date=         [1946]
On_line=      [Bristol]
*********************************
*********************************
Title=        [19TH. CENTURY ROMANTICISM IN MUSIC.]
Auther=       [LONGYEAR R.M.]
Source=
Publisher=
Place=
Date=         [1969]
On_line=      [Bristol]
*********************************
```

Figure 9: Presentation of the output.

```
*********************************
Title=        [13 CENTURIES OF ENGLISH CHURCH MUSIC. 2nd ed.]
Auther=       [PARRY W.H.]
Source=
Publisher=
Place=
Date=         [1946]
On_line=      [Bristol]
*********************************
*********************************
Title=        [13 CENTURIES OF ENGLISH CHURCH MUSIC.]
Auther=       [PARRY W.H.]
Source=
Publisher=
Place=
Date=         [1946]
On_line=      [Bath]
*********************************
*********************************
Title=        [19TH. CENTURY ROMANTICISM IN MUSIC.]
Auther=       [LONGYEAR R.M.]
Source=
Publisher=
Place=
Date=         [1969]
On_line=      [Bath,Bristol]
*********************************
*********************************
Title=        [100 YEARS OF MUSIC.]
Auther=       [Abraham Gerald.]
Source=
Publisher=
Place=        [London]
Date=         [1974]
On_line=      [Bath,Bristol]
*********************************
```

Figure 10: Presentation of the output in MUlti-database Library Enquiry Service.

ref(['13 CENTURIES OF ENGLISH CHURCH MUSIC. 2nd ed.'], ['PARRY W.H.'],A,B,C,[1946],['Bristol']).

ref(['13 CENTURIES OF ENGLISH CHURCH MUSIC.'], ['PARRY W.H.'],A,B,C,[1946],['Bath']).

ref(['19TH. CENTURY ROMANTICISM IN MUSIC.'], ['LONGYEAR R.M.'],A,B,C,[1969],['Bath','Bristol']).

ref(['100 YEARS OF MUSIC.'],['Abraham Gerald.'], A,B,['London'],[1974],['Bath','Bristol']).

Figure 11: The knowledge representation in MULES

4 Conclusions

The Distributed Information System is a practical means of obtaining large amount of information from computer based information systems.

The Knowledge Acquisition System, which is a component of the MDB-ES, was introduced in order to elicit, analyse and interpret the responses which were obtained from remote databases. The Knowledge Acquisition System incorporated several artificial intelligence techniques. It was shown that the incorporation of AI techniques can enable the system to recognise relevant data more effectively, and hence provide greater user satisfaction..

AI techniques will enable users to make more effective use of existing automated systems and on-line databases. They will also help the information systems user to be more productive and efficient in carrying out the many tasks involved in managing an information service centre.

The knowledge assimilation mechanism described in this work was shown to cope successfully with such problems as duplications and incompatibilities in responses. In this context, incompatibilities include differences in conventions, underlying data structures, representation, as well as missing information and conflicting data values.

An important feature of knowledge assimilation in a knowledge acquisition system is the consistency, correctness and completeness checking of the responses from each remote database in order to create a local knowledge base for the user. Building the knowledge base is very expensive and labour intensive. Typically an experienced information scientist is able to add about 100 terms a week. The MDB-ES architecture has the ability to build knowledge base automatically for a particular subject, defined by the user without delay and additional expenses.

References

[1] Barfoursh, A. A.; Lance, G. N.; Rogers, M. H. 'A Model for Effective Computerised Information Systems: Multi DataBase Enquiry Service'. In *Proceedings of the 11th BCS IRSG Research Colloquium on Information Retrieval*, pages 107–139, British Computer Society, Polytechnic of Huddersfield, U.K., 1989.

[2] Fox, Edward A. 'Research and Development of Information Retrieval Models and Their Application'. *Information Processing & Management*, 25(1):1–5, 1989.

[3] Salton, Gerard; McGill, Michael J. *'Introduction to Modern Information Retrieval'*. McGraw-Hill, 1983.

[4] Blair, D.C.; Maron, M.E. 'An Evaluation of retrieval effectiveness for a full-text document retrieval system'. *Communications of the ACM*, 28(3):289–299, 1985.

[5] Van Rijsbergen, C.J. 'Office Automation (Special issue)'. *Information Processing & Management*, 22(2):, 1986.

[6] Sparck Jones, K.P. 'Intelligent Retrieval'. 1983. Volume 7, pages 136-142; ASLIB, London.

[7] Brooks, H.M. 'Expert Systems and Intelligent Information Retrieval'. *Information Processing & Management*, 23(4):367–382, 1987.

[8] Chiaramelia, Y.; Defude, B. 'A Prototype of an Intelligent System for Information Retrieval: IOTA'. *Information Processing & Management*, 23(4):285–303, 1987.

[9] Tong, R.; Askman, V; Cuningham, J. 'RUBRIC: An Artificial Intelligence approach to Information Retrieval'. In *Proceedings of the First International Workshop on Expert Systems*, 1984.

[10] Thompson, R.H.; Croft, W.B. 'An Expert System for Document Retrieval.'. In *Proceedings of the Expert Systems in Government Symposium, Washington D.C.*, pages 448–456, IEEE Computer Society Press, 1985.

[11] Fox, Edward A. 'Development of the CODER system: A test-bed for AI methods in Information Retrieval'. *Information Processing & Management*, 27(4):341–366, 1987.

[12] Kidd, Alison L., editor. *'Knowledge Acquisition For Expert Systems'*. Plenum Press, New York, 1987.

[13] Michie, Donald, editor. *'Introduction Reading In EXPERT SYSTEMS'*. Gordon and Breach Science Publisher., 1982.

[14] Feigenbaum, E.A; McCorduck, P. *'The Fifth Generation'*. Addison Wesley, Reading, Massachusetts, 1983.

[15] Waterman, D. A. *'A Guide to Expert Systems'*. Addison Wesley, Reading Massachusetts, 1986.

[16] Grabowski, Martha. 'Knowledge Acquisition Methodologies: Survey and Empirical Assessment'. In *Proceedings of the Ninth International Conference on Information Systems Minneapolis, Minnesota*, pages 47–54, 1988.

[17] Kitakami, H.; Kunifuji, S.; Miyachi, T.; Furukawa, K. *'A Methodology for Implementation of A Knowledge Acquisition System'*. ICOT Technical Report: TR-037, Institute for New Generation Computer Technology, 1983.

[18] Green, D. P. 'Automated Knowledge Acquisition: Overcoming the Expert System Bottleneck'. In *Proceedings of the Eighth International Conference on Information Systems Pittsburgh, Pennsylvania*, pages 107–117, 1987.

[19] Fellers, J.W. 'Skills and Techniques for Knowledge Acquisition: A Survey, Assessment, and Future Directions'. In *Proceedings of the Eighth International Conference on Information Systems Pittsburgh, Pennsylvania*, pages 118–132, 1987.

[20] Hoffman, R.H. 'The Problem of Extracting The Knowledge of Experts from the Perspectives of Experimental Psychology'. *AI Magazine*, 8(2):53–67, 1987.

[21] Clocksin, W.F.; Mellish, C.S. *'Programming in Prolog'*. Springer-Verlag, 1981.

KNOWLEDGE ACQUISITION IN A DEVELOPMENT METHODOLOGY FOR KNOWLEDGE-BASED APPLICATIONS

Susan Spirgi
IBM Switzerland, Hirschgässlein 11, 4010 Basel, Switzerland

André R. Probst
IBM Switzerland and University of Lausanne, Hohlstrasse 550, 8048 Zürich, Switzerland

Dieter Wenger
Swiss Bank Corporation, Hochstrasse 16, 4053 Basel, Switzerland

Abstract

The proposed methodology is based on the following principles: Knowledge acquisition is result-, meta-model- and prototype-driven. A knowledge based application is seen as an active partner for the user and a system consisting of active autonomous cooperating agents. The agents are connected to object-types and have therefore the abstraction level of the object-types. The system-oriented approach is the base for parallelism, high flexibility and for a natural implementation of truth maintenance. Knowledge in a knowledge-based application is subdivided into domain knowledge and general processing knowledge. It is assumed that a task executed by a person or an application requires a different kind of knowledge. The development of knowledge based applications and therefore also knowledge acquisition is seen as an iteratively carried out task. A methodology is developed as a knowledge-based application; this means that it is active and executable. A methodology is developed with help of itself. Guidelines for developing the knowledge model of the methodology, as well as the consistency checks, are based on an axiomatic defined mathematical theory (this part is not yet complete).

1. INTRODUCTION

The most serious problem in knowledge acquisition is to know which of the elicitated knowledge should be extracted and modeled. It is shown in chapter 2, that knowledge acquisition is driven by the required results, by consistency checks of the knowledge model and by verification and validation of the prototype. The knowledge representation model is developed according to a certain conception of knowledge-based applications (task-oriented applications). An important attribute of the knowledge model is its system-oriented construction. In chapter 3 the position of the knowledge model within the methodology is outlined, the importance of the knowledge model for an efficient knowledge acquisition and prototype generation is emphasized, and the knowledge model components (meta-concepts), that are introduced in chapter 2, are explained in more detail. The significance of the concept structure for the flexibility of an application is explained. Three views are mentioned that make specific aspects of the knowledge model visible and serve as an interface between developer and knowledge model. Chapter 4 shows that this methodology is realized as a knowledge-based application and is developed with help of itself. Chapter 5 describes the most important parts of the methodology, the consistency checks and the guidelines. Some of them are described in detail. At the end of chapter 5, a part of an axiomatic defined mathematical theory for the knowledge model is presented. It is shown that the consistency checks come from theorems that are derived from the axioms. Chapter 6 gives an outlook on future developments.

2. KNOWLEDGE ACQUISITION IN THIS METHODOLOGY

2.1. POSITIONING OF THE METHODOLOGY

Knowledge acquisition, as mentioned by Breuker and Wielinga [2], is a permanent and crucial activity throughout all stages of the development and maintenance of knowledge based applications. It plays therefore a central role in a development methodology for knowledge based applications. In this methodology knowledge acquisition is subdivided in knowledge elicitation, knowledge extraction and knowledge modeling (see Appendix A).

One of the most serious problems is to find which of the elicitated (externalised) knowledge should be extracted and modeled. Therefore there is a need that knowledge acquisition is driven. In this methodology knowledge acquisition is driven by the required results (shown in chapter 2.2), the knowledge meta-model and the prototype (shown in this chapter).

For driving knowledge acquisition it is necessary that criteria exist. One criterion results from the knowledge model, because a knowledge model must be consistent. Consistency restrictions can be derived from restrictions of the relations between the attributes of the meta-concepts. In chapter 3.1. the knowledge meta-model is outlined in more detail and in chapter 5.5 a part of it is axiomatically defined. Two other criteria result from the prototype, because the prototype must be verified and validated. The knowledge model must satisfy the criterion of consistency and the prototype the criteria of verification and validation. Therefore knowledge acquisition is driven by the meta-model and the prototype.

The rapid prototyping approach includes the criteria of verification and validation but not the one of consistency, because there is no knowledge model in the rapid prototyping approach. Therefore rapid prototyping has been rightly critisized. But this does not mean that knowledge acquisition can be done without prototyping. Recently some structured methodologies have been developed that emphasize strongly the knowledge modeling [1], but they run the risk of loosing prototyping capabilities, because sometimes their knowledge models seem to be too complicated, difficult to manage, too restrictive or not adequate to a prototyping environment.

In order to find a powerful but manageable knowledge model, we must have a clear conception of our goal, which is a knowledge-based application.

2.2. THE CONCEPTION OF A TASK-ORIENTED APPLICATION

There are a lot of criteria for classifying data-processing applications. A simple criterion is to distinguish between applications which provide an infrastructure to an organisation unit (organisation-oriented) and those supporting individuals to execute a task (task-oriented). The first ones automate the data-communication between tasks by providing data input facilities and by making data available around the organisation. These are the typical computer applications of today. The second ones are task-oriented applications. Such an application supports a person (user) to perform a specific task in a specific domain. User and application work in a partnership. These are the applications we envisage.

The question how a user should be supported in doing his job is crucial for the knowledge that has to be acquired. Our conception is, that the user and the application execute the task as a team. For the user the application is a partner, who is able to perform parts of the task by himself; who is able to offer many help-activities; who can have controlling capabilities by representing the view of the organisation; who can have knowledge about theories of the task-domains; etc..

Figure 1 illustrates the principle of a task-oriented application. For executing the task, some input is needed and by executing the task, some output is produced. A task is divided into the user and application part. To perform a given task, a set of activities is carried out; some activities are done by the user (user part of the task) some by the application (application part of the task) and some others are shared. It is possible, that there is no user. Then the task is carried out completely automatically. The application part consists of a machine interface (MI), the kernel system and the user interface (UI). The MI and UI take input information from other machines or from the user. The kernel system is the problem solving part of the application which contains the problem solving knowledge. The kernel system processes the information provided by the MI and UI into results (new information). These results are made again available with the help of the MI and UI to other machines and to the user. The

user sees the application by the user-interface (UI). Although this conception about a knowledge-based application is quite general, it helps us to build a powerful knowledge model by providing components for the knowledge model. The components are the meta-concepts of the meta-model.

Originating from this conception, we have to combine two opposite requirements. On the one hand the user must master his task, on the other he should be discharged from many activities belonging to his task. This demands high requirements of the UI. So the machine activities should be transparent to the user as if he was performing the activities himself, and the UI should show models and views of the domain. The important principle is that the user drives the application and not vice versa. Otherwise he is reduced to a sensor whose only responsibility is to provide data to the application.

Because the application consists of the UI, the MI and the kernel system, the knowledge model must also have these three parts. The knowledge model of the UI incorporates the knowledge specifying, how the user wants to interact with the application and the knowledge model of the kernel system incorporates the knowledge describing, what activities and results the user wants from the application.

The UI and the MI have the job to get **events** from the environment. Such an event stimulates the application to react. The application can be seen as a system, that is normally in equilibrium and in a consistent state. By an event, the application is thrown out of balance and forced into an inconsistent state. Now it is the application's job to reach a new consistent state again. For the knowledge model and the application this system-oriented approach is fundamental. The reaction of the system to an event is seen as a reaction done by autonomous but cooperating components called **agents**. The agents are the active parts of the knowledge model. Depending on the event some of the agents react and bring the system to a new consistent state. An **application activity** done by the knowledge-based application is thus carried out by the activated agents reacting to a specific event. It is in the UI's responsibility to show the user the new reached state.

The agents process the information. For this the agents need knowledge. They use two kinds of knowledge: knowledge about the domain (domain-oriented knowledge) and general processing knowledge (general process-oriented knowledge). The general process-oriented knowledge consists of processing techniques such as search algorithm, reasoning techniques, neural nets, etc.. These are the techniques, that are embedded in knowledge-based applications (often called AI-Techniques) and are of course domain-independent.

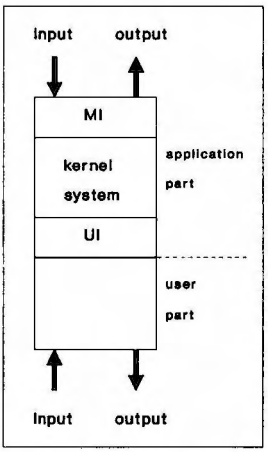


Figure 1 Principle of a task-oriented application

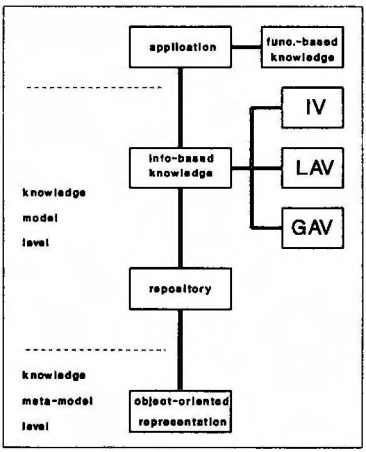


Figure 2 Relationships between application, knowledge model and meta-model

The domain-oriented knowledge, that has to be acquired, is the knowledge used by the agents and is also necessary to perform the activities (remember: an activity is carried out by agents). The general process-oriented and domain-oriented knowledge can be classified in information- and functional-based knowledge. The information-based knowledge is declarative and is built by **concepts** and **attributes**. The functional-based knowledge is operative and is in a lisp-like form.

An activity (application activity) is started by an event and provides some results. In this conception, knowledge is connected to an activity and does not stand for its own. So the question, what knowledge the developer has to acquire, can be answered. It is the domain knowledge which is used to perform an activity for providing the required results. Hence our knowledge acquisition is driven by the required results. In chapter 5.4.2. result-oriented KA is outlined as a guideline.

We now have all three components that drive knowledge acquisition. So first the knowledge is extracted according to the required results, second, as outlined in chapter 2.1, the knowledge in the knowledge model is examined by consistency checks and third the knowledge is verified and validated with help of the prototype. That is why knowledge acquisition in this methodology is result-, meta-model- and prototype-driven.

In the rest of this paper the aspects concerning meta-model-driven knowledge acquisition shall be specially treated.

3. THE KNOWLEDGE MODELS

3.1. THE MODELS, REPRESENTATION, REPOSITORY AND VIEWS

In figure 2 the relationships between application, information-based knowledge, views, repository and its representation can be seen. An application has a part consisting of functional-based and one consisting of information-based knowledge. The information-based knowledge (declarative) builds the knowledge model. The functional-based knowledge (operative) makes the application executable. This knowledge is not included in the knowledge model. The information-based knowledge is the information about the application and is stored in the repository. The object-oriented representation defines what kind of information is stored in the repository. This object-oriented representation is the knowledge meta-model with its meta-concepts. The meta-model may be also called the knowledge representation model (language). The information based knowledge can be represented by three different views, the information view (IV), the local agent view (LAV) and the global agent view (GAV). The views are graphical representations of the knowledge model and serve as the interface to the developer. They represent a specific aspect of the knowledge model. The developer manages the information about the application with the help of these three views. The knowledge model is built up with the meta-concepts and their attributes. Therefore the knowledge model has to be built according to the relationships existing in the knowledge meta-model. It is not allowed to violate any relations of the meta-model.

The knowledge model, because it models the application, is specific to the application and comprises the information-based knowledge, the views and the repository. The meta-model remains the same for all applications. More for academic reasons it shall be added that the object-oriented model is a meta-meta-model of the knowledge model, because the meta-model is built with the concepts of the object-oriented model.

3.2. THE POSITION OF THE KNOWLEDGE MODEL

As outlined the knowledge model is a model of the application and therefore not a model of the domain. The domain is the source for building the knowledge model. There are two reasons for this.

- In the application development, only the domain knowledge is of interest that should be contained in the application. Other knowledge is worthless, even unnecessary weight for application development.
- Only a partial and incomplete model of the domain can be built because of the bad accessibility of real-world knowledge. This seems to be a serious limitation but it is not, because it does not make sense to duplicate the reality. It is not necessary, even wrong, that the application should perform the task in the same way as a person would do it, because a person and a machine have

their own specific capabilities. A task done by a person is adequate to his capabilities and so the knowledge that is used. The same has to be done with the knowledge that is used by the application. It has to be adequate to the specific capabilities of a machine. This implies that the knowledge used by a person or by a machine is different. A good example is the pocket calculator that calculates differently than a person. As a result, knowledge acquisition must also be seen as a process of developing new knowledge. In a knowledge-based application the results are imitated and not the processes.

Figure 3 shows that the domain and the prototype are on the reality level and the knowledge model is on the modeling level.

Because knowledge acquisition and prototype generation are the main subtasks in application development and because they are carried out iteratively, the knowledge model has to conform to an efficient knowledge acquisition and prototype generation. Therefore we have, besides the requirement that a methodology must be computer-based, additional requirements:

- The methodology should help to bring unstructured knowledge into a structured form.
- The methodology should help actively to find missing, incomplete and contradictory knowledge.
- The methodology should allow to transfer the acquired knowledge easily into a prototype for verifying and validating the acquired knowledge.

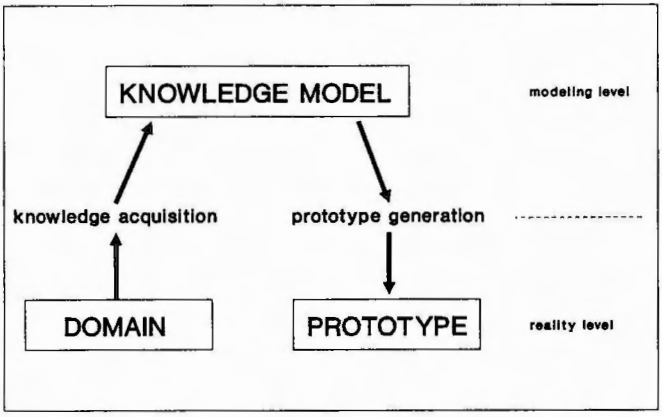


Figure 3 Relationship between domain, knowledge model and prototype

3.3. THE META-CONCEPTS OF THE META-MODEL

The meta-concepts of the meta-model are those mentioned in chapter 2.2. such as concept, attribute, information, event, agent, etc..

A **concept** can be an **objecttype** (e.g. person, car, vehicle) or an **attribute** (e.g. color). The concepts and their relations build the **concept structure**. Three kinds of relations are defined between concepts. First the **attribute-relation** between object-type and attribute (e.g. car has as attribute color); second the **generalisation** (isa-relation) between object-type and object-type (e.g. a car is a vehicle); third the **aggregation** (ip-relation ... is part of) also between object-type and object-type (e.g. an engine is part of a car). The ip- and isa-relation are **abstract-relations**. They build up the concept structure. The **association** is also a relation between objecttypes (e.g. a person visits a town). The associated object-types are on the same abstraction level. **Objects** are instances of object-types (e.g. my car is an instance of the object-type car). An atomic piece of information consists of an object, an attribute and a value (or values) (e.g. my car has attribute color and the color is blue). Such a piece of information is called an

information-unit (infounit). An **information-type** is therefore defined by an object-type and an attribute (e.g. car has attribute color). Also a set of infounits of the same information-type is defined by an object-type and an attribute and is called an information-set (infoset). Hence an infounit is an element of an infoset. The infosets are attached to the concept structure (remember: an infoset is built by an object-type and an attribute). The **dependency** relation is a relation between infosets. Exactly one infoset is the dependent one and one or more infosets can be seen as the causal ones. The dependencies are also attached to the concept structure by the dependent infoset. This is an important point to remember.

The **agents** are the active elements in an application. One agent corresponds to exactly one dependency. An agent processes the infounits of the causal infosets into infounits of the dependent infoset. An agent manipulates the infounits of exactly one infoset. One or more agent(s) can manipulate the same infoset. Manipulation means asserting, modifying or deleting of one or more infounit(s). The infosets that are manipulated by agents are called dynamic.

For producing the output infoset (the dependent infoset), the agent needs different kinds of knowledge:

- **activation knowledge**:
  The information specifying when the agent has to become active.
- **input knowledge**:
  The information about input infosets which have to be processed by the agent for producing the output infoset.
- **constraint knowledge**:
  The information about infounits of the input infosets which have to be selected.
- **process context knowledge**:
  The information that drives the processing (processing parameters).
- **functional knowledge**:
  The knowledge that is in functional form, specifying how the processing is done.

An agent is more than a function, because an agent calls itself. Its activation is in its own responsibility. An agent is never called by other agents. An agent is driven by the concept structure and the information embedded in the concept structure. It is an autonomous active element in the application. As an important consequence, a model, that is built by agents, is a model that allows parallelism. Besides that, an implementation of truth maintenance (propagation of changes) is naturally supported.

As mentioned before, the dependencies are attached to the concept structure and so are the agents, because an agent corresponds to exactly one dependency. The more general an infoset manipulated by an agent is, the more general is this agent. The infosets and agents have the abstraction level of the object-types they are attached to. One of the most important guidelines in knowledge-based application development is to build the most general possible agents: This means to build agents that are attached to the highest possible abstraction level. Following this guideline, flexible and general applications can be built.

The **events** are the information that come from the environment of the application. Because there are different kinds of interfaces to the environment, there are different kinds of events:

- **UI-event**:
  An event caused by the user.
- **machine-event**:
  An event caused by another application or coming directly from the domain. The events are the primary stimulus for the agents.

3.4. THE VIEWS OF THE KNOWLEDGE MODEL

3.4.1. THE INFORMATION VIEW (IV)

The IV contains the meta-concepts object-type, attribute, attribute-relation, isa-relation, ip-relation and association. This view is represented in a object-oriented way and is itself again subdivided in the **abstraction view** and **relational view**. The abstraction view contains all the meta-concepts of the IV but without the association. The relational view contains all the meta-concepts of the IV but without the isa-

and ip-relation (abstraction relation). The relational view corresponds to the relational model in database modeling. In chapter 5.3. an abstraction view (figure 6) and a relational view (figure 7) are shown.

3.4.2. THE GLOBAL AGENT VIEW (GAV)

The GAV contains the meta-concepts agent and event. This view shows the reaction of the system to an event. Hence it shows an application activity. The reaction of the system is an ordered parallel change of certain infosets (done by agents). The order is given by the fact that the change of an infoset may depend on the change of one or more other infoset(s). Because the manipulation of an infoset is done by exactly one agent, the infosets can easily be tranformed into agents. The infosets can be replaced by the agents and we get a GAV of an activity. Two examples of a GAV (figures 9 and 10) are shown in section 'guideline for building the GAV' (chapter 5).

3.4.3. THE LOCAL AGENT VIEW (LAV)

The LAV contains the meta-concept agent. This view carries the information-based knowledge that an agent must have for doing its job.

In figures 4a and 4b the symbols of the LAV are shown. As explained in section 3.3. the agent's information-based knowledge can be subdivided into activation, input, constraint and process context knowledge. One part of the activation, input, constraint and process context knowledge is the information about the infosets that are involved. This part of the knowledge is represented with the help of the LAV. The example shows the input knowledge (figure 4a), where the agent needs the infosets x, y and z as input infosets, and the constraint knowledge (figure 4b), where the agent needs the infosets u and v to constrain the infounits of the input infosets.

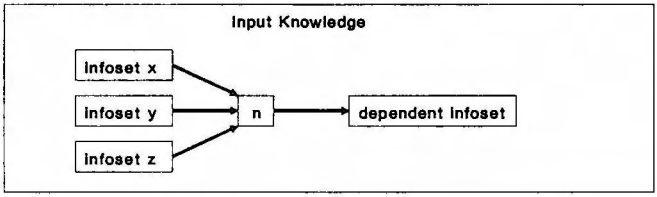


Figure 4a Involved input infosets

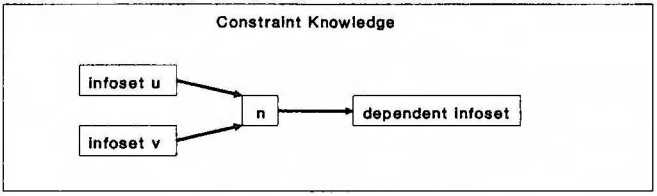


Figure 4b Involved constraint infosets

4. THE METHODOLOGY AND THE TOOL

A developer performs the task 'development of a knowledge-based application'. A methodology supports the developer in doing this task. This is a task like any other task which is supported by a task-oriented application. For this reason it is obvious that the task 'development of a knowledge-based application' should also be supported by a knowledge-based application tool, which has to take into consideration that:

- A developer must not be forced to apply a rigid application, because developing an application is not a step by step processing, but it is a highly interactive cooperation between developer and methodology.
- There are no stages in this methodology, because the application consists of many parts. These parts are developed in parallel and iteratively. So at a certain moment they all have their own state. Therefore stages make no sense.

The methodology is the knowledge of developing a knowledge-based application. This knowledge-based application incorporates the knowledge of the methodology. The application is therefore the executable methodology. In this way the methodology is made executable or operative, and therefore it is an active methodology and a computer-based one. It is called EMA (Executable Methodology for developing knowledge-based Applications). From the outset EMA has been developed as a computer-based methodology and for this reason many of its activities are too cumbersome to perform manually if possible at all, e.g. the consistency checking facilities. Therefore a paper-based version of EMA is incomplete and does not exist.

It makes a difference, whether a methodology is developed as a computer-based one or a paper-based one, because some methodology know-how is not applicable and has no significance in a paper-based methodology, whereas it can be essential in a computer-based one. That is also a reason, why it is dangerous to transform a paper-based methodology in a computer-based one without a complete rethinking and redesign of the methodology. As mentioned EMA is built as a task-oriented application. The important step now is to realize, that a knowledge model for EMA must exist. This model contains the knowledge of EMA and herewith of this methodology. The knowledge model of EMA is the meta-model. This means, by presenting the knowledge model of EMA, this methodology is presented. This means also, by developing the knowledge model of EMA, this methodology is developed. Therefore the knowledge model of EMA and herewith this methodology are developed by this methodology. It is important to realize that not only the associated tool of the methodology is built with help of the methodology, but also the methodology itself. Figure 5 should clarify this. Step 1 and 2 (like a bootstrap) are executed by the developer alone. Step 3 and step 4 are executed by the developer with support of EMA, because the first version of EMA is built. By step 4, a new version of EMA is generated. By executing step 3 and 4, other versions of EMA can be produced.

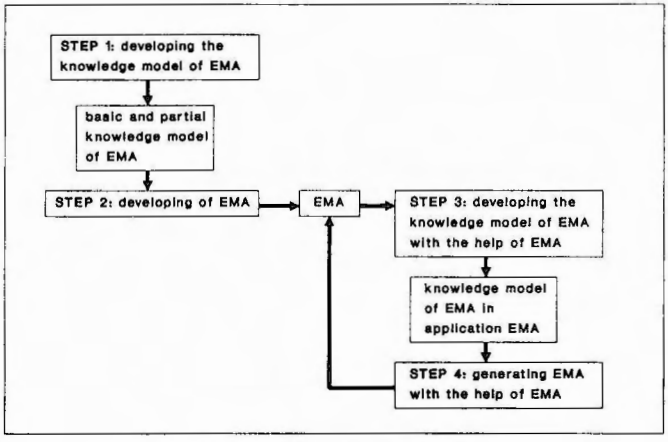


Figure 5 Development cycle for EMA

5. EMA

5.1. THE FUNCTIONALITY OF EMA

The requirements for EMA are those mentioned in chapter 3.2. These requirements should be satisfied by the following functionality:

1) EMA consists of guidelines that support the developer in extracting and modeling knowledge from the elicitated knowledge.
2) EMA consists of consistency checks that directs the developer's attention to missing, incomplete or contradictory knowledge.
3) EMA generates the basic structure of the program code.
4) EMA helps to optimize the program code.

Points 3 and 4 of the functionality are essential for the usefulness and applicability but they are technically oriented and therefore not topics of this paper. The first two points are of interest for knowledge acquisition and will be dealt with in the next chapters.

5.2. DESCRIPTION OF EMA

As mentioned EMA is nothing else than a task-oriented application. Therefore EMA has its own knowledge model. So it is the best way to describe EMA by its knowledge model. This is done by showing the three views of the knowledge model. Unfortunately paper is not suitable for showing these views, especially the GAV and the LAV. Therefore only a part of the IV is shown. Instead of showing the two agent views some of the functions are described in words.

5.3. THE INFORMATION VIEW OF EMA

The IV of EMA is the IV of the meta-model. The elements of the knowledge model and herewith the meta-concepts are:

- concept
- object-type
- attribute
- infoset
- agent
- event

Because the IV is represented in an object-oriented way the meta-concepts are represented as object-types. Figure 6 is the abstraction view of the kernel system of EMA. The object-types possess also attributes which are not shown in figure 6 (e.g. object-type 'concept' has as attributes 'name' and 'description' besides other ones). Figure 7 is the relational view of the kernel system of EMA. As it can be seen the object-type 'attribute-relation' imports the two object-types 'object-type' and 'attribute' (e.g. color of a car).

5.4. DESCRIPTION OF SOME FUNCTIONS OF EMA

The functions that are described are guidelines and consistency checks. The consistency checks are triggered automatically by certain states of the information kept by EMA. They produce hints that draw the user's attention to the special situation. The consistency checks and guidelines build as a whole a powerful help for the developer. On their own they are sometimes rather simple functions. There are a lot of consistency checks and guidelines. Some of them are outlined in detail:

5.4.1. CONSISTENCY CHECK FOR THE CONCEPT STRUCTURE

It is not allowed to define recursive object-type structures, because a structure is built up by isa- and ip-relations and both relate a specific object-type to a more general one (see also chapter 5.5).

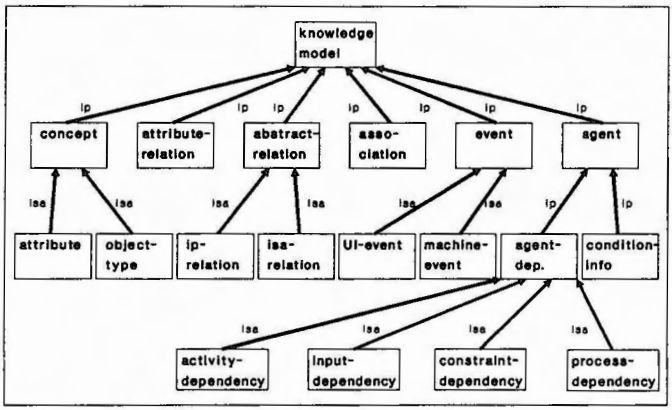


Figure 6 Simplified abstraction view of EMA

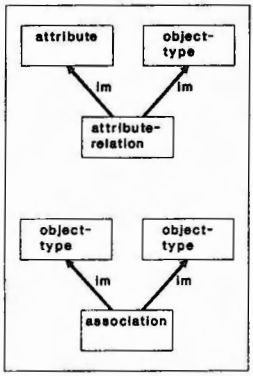


Figure 7 Simplified relational view of EMA

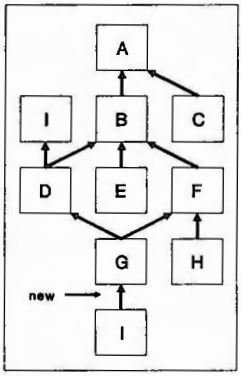


Figure 8 Example of a structure with recursion

Let us imagine (figure 8) "I" is connected to "G" with an is-a relation. The system checks, whether there is a path in direction of more abstraction from "G" to an "I". In the example EMA finds the path "I"-"G"-"D"-"I", so we get an inconsistency. We can infer that one or more of the relations "I"-"G", "G"-"D" or "D"-"I" are wrong. As we don't know which one, EMA must allow the user to relate "I" to "G". Therefore the concept model becomes inconsistent. A hint, created by EMA, draws the user's attention to this inconsistency.

It is easy to understand that there are some further complications e.g.: the user can add more relations and therefore other inconsistencies could occurr. EMA has to pay attention not to fall in a loop, when inconsistencies are checked in a already inconsistent concept structure.

5.4.2. GUIDELINE FOR RESULT-ORIENTED KNOWLEDGE ACQUISITION

The following steps are carried out iteratively:

1) Identify a result that the user requires.
2) Transform the result into one or more infoset(s). This could mean that new objecttypes and attributes have to be created. The infosets become the dependent infosets of agents.
3) Identify agents that produce the dependent infosets.
4) Identify the knowledge used by the agents for producing the required infosets.
5) Classify the kind of the agent's knowledge (activation-, input-, constraint-, process context knowledge) and transform it into infosets. The infosets of the input knowledge of an agent are the results and herewith again dependent infosets of other agents. On these new dependent infosets step 3), 4) and 5) should be applied.

5.4.3. GUIDELINE FOR BUILDING THE GLOBAL AGENT VIEW

A GAV represents an activity which is defined as the reaction of the application to an event.

For building the GAV two kinds of knowledge of the involved agents are used; the input knowledge and the activation knowledge. When a GAV is built, the information about the involved agents on the one hand and the agent's knowledge about input and activation on the other hand are yet incomplete.

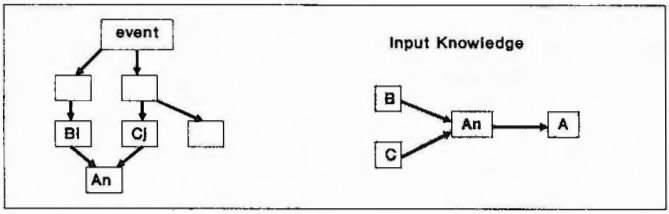


Figure 9 Example of a GAV (on the left) and a LAV

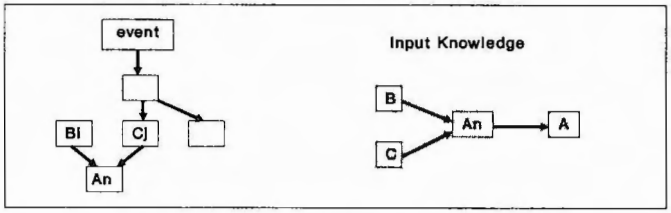


Figure 10 Example of a GAV (on the left) and a LAV

For building a GAV the following steps have to be done:

1.step: done by the developer (user)
Identification of the event.

2.step: done by the developer (user)
Identification of the infosets, that are affected by the event. By keying in these infosets they appear in the window of the GAV.

3.step: done by the developer (user) and the system
Each time an agent is identified other agents may appear on the screen. Assume that a just identified agent manipulates an infoset that is a causal infoset for input or activation of any other agent, then this other agent is displayed by EMA and is connected to the just identified agent.
By this way a first GAV can be built.

4.step: done by the developer (user)
The developer adds additional activation connections, that are necessary for getting the right ordered structure and herewith a synchronisation of the agents.

5.step: done by EMA
The GAV can be checked for completeness. Various different checks are done, one is explained here in detail.

Figure 9 shows on the left hand a part of a GAV and on the right hand the input knowledge part of the LAV of the agent "An". The part of the LAV says, that the agent "An" manipulates infoset "A" and has as input the infoset "B" and "C". For this structure we need an activation information from the agents "Bi" (agent of "B") and "Cj" (agent of "C"), because the agent "An" must not become active before both agents "Bi" and "Cj" have become active, because both depend on the same event (remark: if no explicit activation information exists, an agent activates itself as soon as one of its input infosets has changed).

Figure 10 shows a similar structure where no activation information is necessary, because the infoset "B" is not affected by the event of this GAV. The agent "An" becomes active as soon as agent "Cj" has become active.

5.4.4. CONSISTENCY CHECK BETWEEN ISA- AND IP-RELATION

Between two object-types an isa- or an ip-relation may exist, but not both. It is easy to recognize the inconsistency in figure 11a, but it is not as easy to recognize the one in figure 11b or in any other more complicated structure.

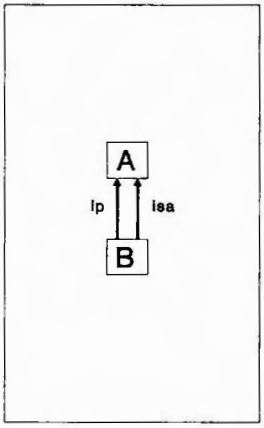


Figure 11a Concept structure

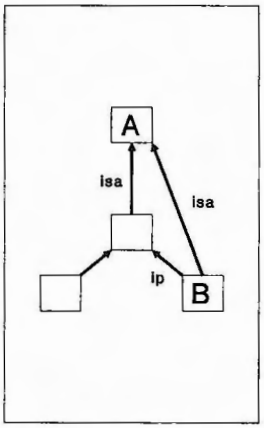


Figure 11b Concept structure

5.4.5. GUIDELINE FOR QUALITATIVE REASONING

In qualitative reasoning one often encounters the problem where first a dependent information depends on an infinite set of causal information (infinite is understood in the sense, that you never can say, this is all the causal information for this dependency), where second no one can say how the dependent information depends on the causal one, and where third there is quite a lot of causal information.

The solution is to decompose this dependency in several dependencies. But how can this be done? The approach is to consider each causal information as an element of a set. The name of the set is put together with the name of the dependent infoset and the prefix 'information to'. In figure 12a, the causal information is A1, A2,..., An. Now we have to find subsets of the set 'information to ...'.

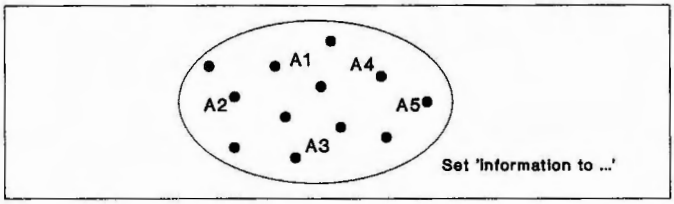


Figure 12a Element-set-relation

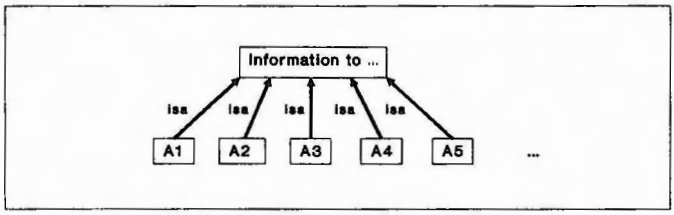


Figure 12b First generalisation structure

The element-set-relation corresponds to the isa-relation. Therefore it is the same to say that each causal information is an object-type and has an isa-relation to the object-type 'information to ...'. So a first generalisation structure (figure 12b) is obtained. Because the relation between subset and set corresponds to the isa-relation, the next step is to find generalisations of the causal information (figure 12c). In this way a structure is obtained, which is now taken as the dependency-structure, where the more general information depends on the direct specific informations. These new dependencies are now manageable, because in the individual dependencies fewer causal infosets are now involved.

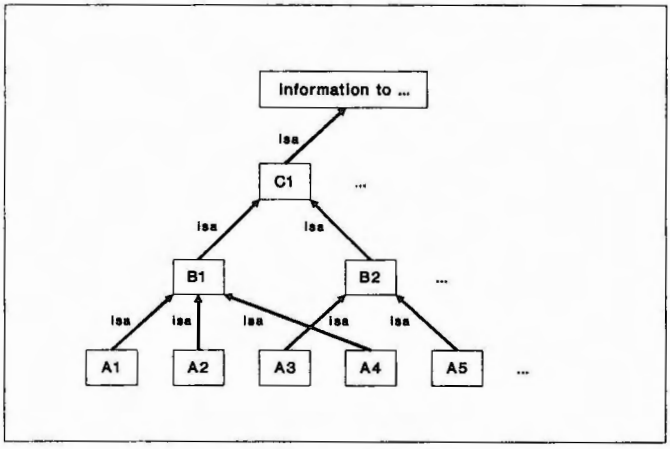


Figure 12c Final generalisation structure

5.5. THE FOUNDATIONS OF THE GUIDELINES AND CONSISTENCY CHECKS

The underlying principle of developing this methodology is to find relationships and restrictions between the infosets in the meta-model (built by meta-concepts and their attributes). Guidelines and consistency checks are derived from these restrictions. The guidelines and consistency checks are an important part of the methodology and are carried out by agents. The more useful relationships and restrictions we find, the more powerful methodology we get. A way of defining and building a methodology is with help of an axiomatic defined mathematical theory. The guidelines and consistency checks can be reduced to a mathematical theory, which is defined by means of some basic concepts and axioms. The axioms are characteristics of the basic concepts. The derived theorems lead to the consistency checks. The next section should clarify this. A part of the theory is outlined.

Basic Concepts:

The set of the basic concepts is {O, IP, ISA, R, ...}.

O is the set of the object-types.

IP is a subset of OxO (cartesian product). It is the set of the object-types related with an ip-relation. An element of IP is written as 'a IP b', where a and b are elements of O.

ISA is a subset of OxO. It is the set of the object-types related with an isa-relation. An element of ISA is written as 'a ISA b', where a and b are elements of O.

R is a set union of IP and ISA. An element of R is written as 'a R b', where a and b are elements of O. Each element of R has its inverse element. They are elements of the set RI (e.g. the inverse element of 'a R b' is 'b RI a').

Axioms:

Three of the axioms are outlined. Assume that b,c,d are any elements of O and are different. The following symbols are used:

- sect ... set intersection
- el ... element of

Axiom 1: (b R c) el IP = > (b R c) not el ISA (b R c) el ISA = > (b R c) not el IP

The interpretation is that, if there is an ip-relation between b and c, then an isa-relation between b and c is not allowed and vice versa.

Axiom 2: ((b IP c) and (c R d)) or ((b R c) and (c IP d)) = > (b IP d)

The interpretation is that, if there is a connecting path from b to c and one from c to d and one of them is in an ip-relation, then a connecting path from b to d exists and is in an ip-relation (the ip-relation has highest priority).

Axiom 3: (b ISA c) and (c ISA d) = > (b ISA d)

The interpretation is that, if there is a connecting path from b to c and one from c to d and both are in an isa-relation, then a connecting path from b to d exists and is in an isa-relation.

Axiom 4: (b R c) el R = > (b RI c) not el RI

The interpretation is that, if there is a relation between b and c, then the inverse relation of b to c is not allowed (e.g. a car is a vehicle, then a car cannot be kind of a vehicle).

Theorems:

The two following theorems can be derived from the axioms.

Theorem 1: R sect RI = {} {} ... empty set

This theorem can be derived from the axioms 2, 3, 4. The interpretation is that recursive concept structures are not allowed (e.g. the structure of figure 8 does not conform to theorem 1).

Theorem 2: IP sect ISA = {}

This theorem can be derived from the axioms 1, 2, 3. The interpretation is that two different object-types are in no relation or in the same relation to each other, independently of the length of their connecting path (e.g. the structure of figure 11a and 11b is not allowed).

6. PERSPECTIVES

The following developments are envisaged in the near future:

- The methodology will be extended to a **general methodology for all kinds of data processing applications**. Although the knowledge model has been developed according to a special conception of a knowledge-based application, the knowledge model is not limited to this kind of application. Because of the system-oriented approach, this methodology is a promising base for a general development methodology for data processing applications (this has already been verified for some small conventional applications).
- **Generic application development** is another field. It is naturally supported by this methodology, because first the knowledge model has a deep abstract structure where the most abstract object-types and agents build naturally the generic part of an application and the more specific object-types and agents build the application-specific part, second because the knowledge model consists of independent parallel working components (agents).
- The building up of a **library of processing techniques** is a permanent task. These processing techniques build the processing knowledge mentioned in chapter 2.2..
- **The construction of an axiomatic defined mathematical theory** is a main task. This theory will be the base for a better structured, more comprehensive and more powerful methodology.
- **Machine learning** and **knowledge acquisition** techniques will be stepwise integrated, because EMA is a framework for these techniques.

7. CONCLUDING COMMENTS

The first experiences in applying EMA are encouraging. Besides the mentioned guidelines and consistency checks, EMA provides a direct manipulation interface (a user driven windowing environment) with graphical editors for building and managing the knowledge model. The design of the graphical editors is based on the knowledge model views discussed in the paper. Building applications with EMA is facilitated within an object-oriented data-driven implementation environment (such as ART-IM from Inference Inc. or TIRS from IBM). The current version of EMA is implemented in ART-IM and runs on DOS-machines with about 8 MB of main memory and VGA.

Appendix

A. KNOWLEDGE ELICITATION AND KNOWLEDGE EXTRACTION

Although it is not the topic of this paper to describe knowledge elicitation and knowledge extraction, it is necessary to lose some words about both of them. Knowledge elicitation is used to get externalised knowledge. This knowledge must be on a medium like paper or cassette. There are many elicitation techniques coming from psychology. The next step is to extract pieces of knowledge (called knowledge extracts) from this externalised knowledge. In EMA, there are some different kinds of knowledge extracts, e.g. the concept, the generalisation relation, the dependency, etc.. These pieces of knowledge are then modeled by the knowledge model.

REFERENCES

[1] Boose, J.H., "A survey of knowledge acquisition techniques and tools", Knowledge Acquisition, 1, 1989, pp. 3-37

[2] Breuker, J., Wielinga B., "Models of expertise in knowledge acquisition", Memorandum 103 of the VF-project "Acquisition of Expertise", August 1988

A FRAMEWORK OF LEARNING SYSTEM RLS

TAJIMA, Morihiko

Electrotechnical Laboratory
1-1-4, Umezono, Tsukuba-shi, 305 JAPAN
E-mail tazima@etl.go.jp

We are developing a learning system of procedural knowledge. The purpose of the system is to find a faulty part of knowledge and fix the part after executing some sequence of procedural knowledge. The system is named RLS (Recursive Learning System). We show the learning model of RLS and the execution mechanism of each knowledge piece. The model is characterized by three kinds of recursiveness. RLS has several aspects including knowledge representation, execution of knowledge, meta-knowledge generation, and learning of target knowledge by meta-knowledge. An example of execution of Othello game and an example of its simple learning are illustrated.

1. INTRODUCTION

We have developed a learning model and are developing a learning system based on the model. The system is named RLS (Recursive Learning System) and is designed to be a basis of a learning framework of procedural knowledge.

Many learning systems have been developed so far. Their abilities are, however, restricted to specific domains. However, very flexible abilities which can be applied to various domains are considered to be necessary for a really intelligent learning system. In order to realize such an intelligent system, structures to modify not only target knowledge but also meta-knowledge (i.e. knowledge to modify heuristics) is necessary, because heuristics to modify knowledge for specific domain must also be modified or newly created when a system are faced with a new situation. Heurisko [1] is a system which modifies heuristics, but Heurisko is not a general purpose system and can not be used as a framework of general learning system.

Knowledge can be divided into two types: conceptual knowledge and procedural knowledge. Conceptual knowledge is usually described in forms of frame, semantic network, logic, etc. which have been proved useful. Procedural knowledge must also be described in a form which is easy to be applied for learning.

Target knowledge which RLS treats is procedural knowledge. In RLS, procedural knowledge is composed of a number of knowledge pieces. One of the main subjects in the learning of procedural knowledge is to find a fault and fix it after an execution of target procedural knowledge fails as

described in [2]. A unit of knowledge called "theory" in [2] corresponds to a knowledge piece in RLS. While only a single theory is treated at a time in the examples in [2], a sequence of knowledge pieces is usually considered at a time in RLS. The learning process of RLS then consists of assigning blame to a specific knowledge piece among a number of knowledge pieces executed and fixing the blame-assigned knowledge piece. The leaning of RLS is appropriate to do such learning process as described later.

RLS is considered as a good framework of learning system for procedural knowledge. In this paper, we describe the outlines of RLS and show a very simple learning process using a Othello game play as an example.

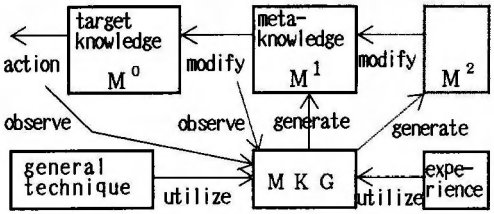


Fig. 1 The learning model

2. RLS

Fig. 1 shows the model of RLS. M^0 is knowledge which does some action in some application domain. The system RLS observes the action. When RLS recognizes that some action fails and that the target knowledge is incomplete, MKG (Meta Knowledge Generator) of RLS generates meta-knowledge M^1, and then RLS lets M^1 modify M^0 utilizing general learning technique or its experiences of past learning. Similarly, when RLS recognizes that M^1 is incomplete, MKG generate M^2 to modify M^1. Therefore, M^1 changes according the learning process proceeds. In general, M^n is generated from M^{n-1} in the same manner.

RLS has several aspects. We call them RLS/K, RLS/A, RLS/G, RLS/L which are as follows:
(a) RLS/K: Knowledge representation of RLS.
(b) RLS/A: Execution of knowledge represented in RLS/K.
(c) RLS/G: Meta-knowledge generation.
(d) RLS/L: Learning of target knowledge by meta-knowledge.
We will explain the structure of RLS/K and the mechanism of RLS/A. Then We will show an example of RLS/G and RLS/L using Othello game play.

Knowledge (both knowledge to be modified and knowledge to modify something) is described in a uniform manner. Knowledge consists of knowledge pieces. Each knowledge piece has a knowledge type. We have several knowledge types shown in Fig.2 which are as follows:

(1) primitive type (Ptype)
Knowledge piece is primitive, i.e. it can not be composed by any knowledge pieces.
(2) successive type (S type)
Knowledge piece consists of several sub-knowledge pieces (actions) which are executed successively.
(3) conditional type (C type)
Knowledge piece consists of condition-action pairs. The knowledge piece

for action corresponding to the knowledge piece of satisfied condition is executed. Conditions are checked from the top to the bottom.

(4) structures

Knowledge piece consists of several actions and an operator. We have four operators at present.

(4-1) OR type

The top action is executed at the first chance. In the second chance after the action is proved to be inappropriate, the next action is employed for execution. Alternative actions are executed similarly.

(4-2) MAX type

Every branch (action) is executed to output some value and the result of the best branch is adopted as the result of this knowledge piece.

(4-3) ORDER type

Every branch is executed to output some value and all branches are sorted in order of the value. Then each branch is executed in the same manner as that of OR type.

(4-4) TOTAL type

Every branch is executed to output some value and the values are totalized.

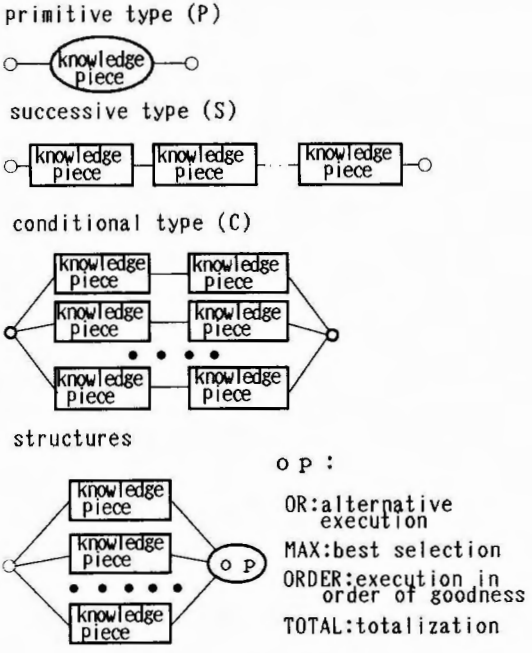


Fig. 2 Types of knowledge piece

3. EXECUTION OF KNOWLEDGE

Operation of RLS/A is fully defined by RLS. The operation of a knowledge piece only depends on which type it belongs to. No extra domain depended mechanism is needed. They are defined as follows:

(a) P type

A knowledge piece of this type is written in some language and can be executed by itself, i.e. the execution of knowledge pieces of P type is

determined by the piece itself. RLS must have some mechanism to call this type of knowledge pieces.

(b) S type
Knowledge pieces are executed successively automatically by the mechanism defined by RLS. Parameter passing mechanism is also defined in RLS to realize information delivery.

(c) C type
Condition check and corresponding action is executed automatically. Parameter passing like that of S type is also defined.

(d) OR type
Alternative execution is done automatically. Application of alternatives is a job of meta-knowledge of the OR type knowledge piece.

(e) MAX type
Max operation is executed by RLS automatically. The selected branch is output.

(f) ORDER type
The execution is a combination of that of MAX type and that of OR type. Sort operation is done between them.

(g) TOTAL type
Totalizing operation is executed by RLS automatically. The total value is output.

To summarize RLS, RLS is characterized by three kinds of recursiveness: (1) meta-knowledge modifies knowledge, (2) knowledge piece is composed of sub-knowledge pieces, (3) MKG generates meta-knowledge of a certain level of target knowledge. Details of RLS/K and RLS/A are described in [3].

4. AN EXAMPLE OF EXECUTION

We will show an example of a Othello game play. Before we show the example, we give a brief explanation of the rule of Othello game. Othello is played on an 8 by 8 board with sixty four dual-colored (one side is black and the other side is white) discs. The initial board configuration has two white discs and two black discs in the center of the board. The two players, black and white, play alternately. A move of a player is to place a disc of player's color on an empty square. A move is legal if the disc can capture some of the opponent's discs. It can capture opponent's discs by bracketing them between the disc itself and an existing disc of the player's color in any of the eight directions. All opponent's discs that can be captured must be captured. Capturing is to flip the opponent's discs to the player's color. The player who has no legal moves is obliged to pass. The player who has at least one legal move must make a move. The winner is the player who has more discs than the other at the end. The details of this game is described in [4].

Fig. 3 shows a part of the structure of Othello playing program RLS-Othello. The program is composed of knowledge pieces of RLS. RLS is written in ESP (an object oriented logic programming language developed by ICOT[5]). Each knowledge piece is realized by an object of ESP. The object name of knowledge piece is shown at each node. Explanations are below the names. You can see the type of knowledge pieces in parentheses. Each knowledge piece of P type is realized as a class object of ESP and denoted by preceding "#". Each knowledge piece of other types is realized as an instance object of ESP and denoted by preceding "$".

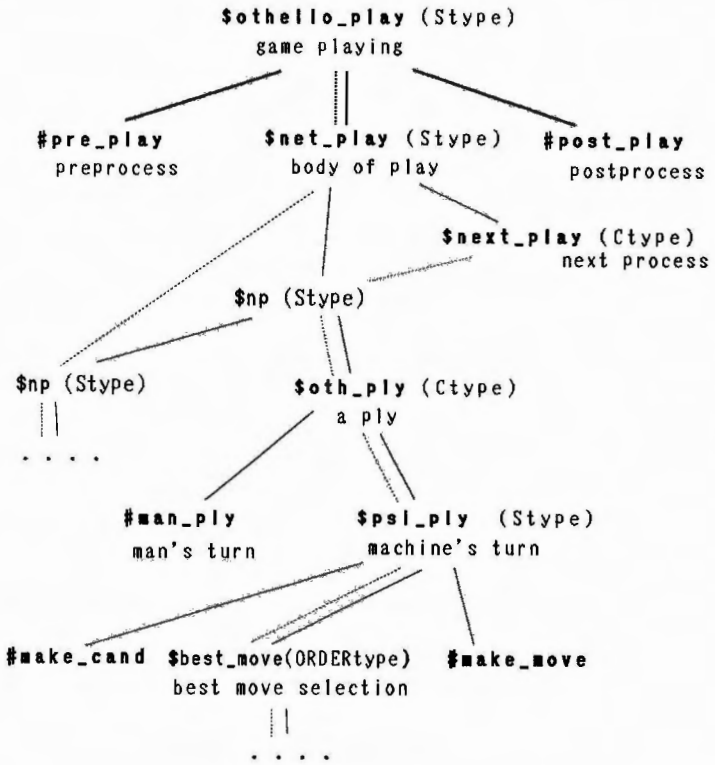


Fig. 3 Knowledge structure of RLS-Othello

The tree of Fig.3 made by real lines shows the composed structure of RLS-Othello. The dotted lines show the creation of sub-knowledge pieces. Since all the necessary knowledge pieces are not known in advance in a program like game playing program, knowledge pieces are created during execution. In order to be modified in a later learning phase, a left recursive structure is created during a game match in this program [6].

Fig.4 shows all kinds of knowledge pieces needed for RLS-Othello. RLS-Othello was given the position after the 25th move (see Fig. 5) and made a move b8. It is waiting for man to make a move now. Knowledge pieces (objects of ESP) are displayed by rectangles. Each instance object can be identified by a serial number assigned to it. Current pointers of execution are shown as black rectangles. Each object has a slot named *result* (shown as "rslt") into which a result of an execution is placed after the execution. If an execution succeeds, result is 1.0, otherwise 0.0 . Each object has a slot named *intention* if necessary which has an intention of the object. For example, instance objects $np have an intention $othello. Instance objects $oth_ply have an intention which is the same as that of its super-knowledge (i.e. $np). Such intention is denoted as "super".

Execution of the knowledge is done automatically by the mechanism defined by each type. In this example, man has black and RLS-Othello has white. The moves made in this example are as follows:

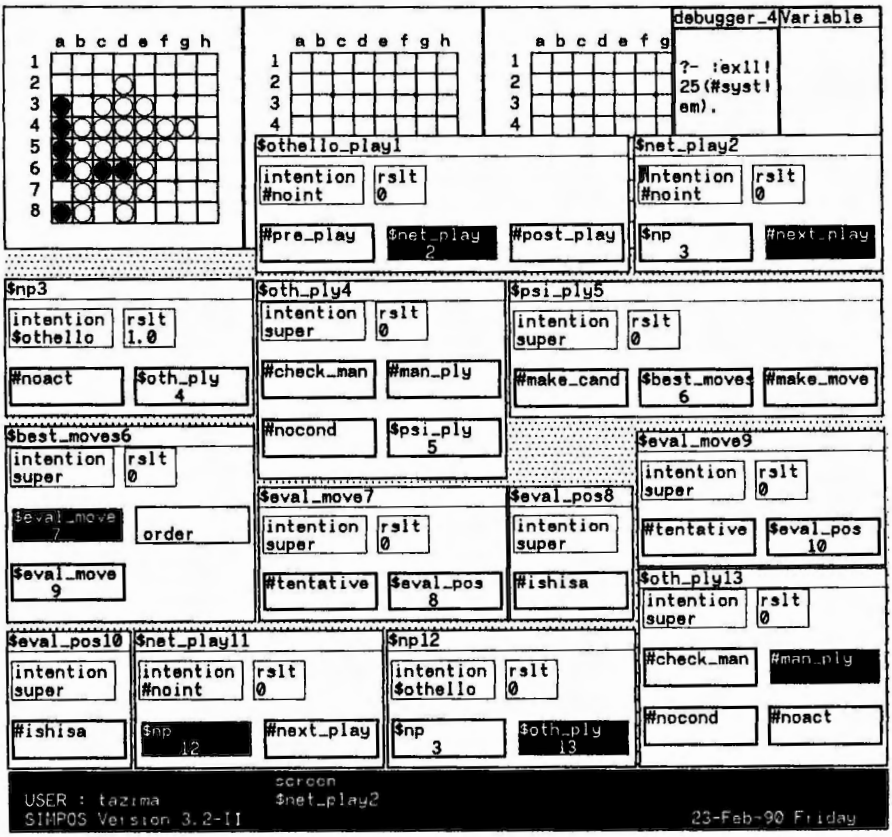


Fig. 4 Knowledge pieces generated by RLS-Othello

26 White b8
27 Black c8
28 White a7
29 Black e8
30 White a2
31 Black a1
32 White pass
33 Black b3.

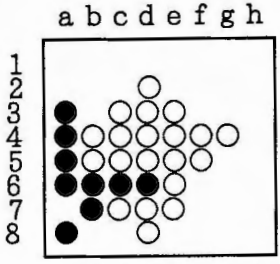


Fig. 5 Initial position

Necessary information of the process of execution is remained in each object executed so that it can be accessed by RLS during the future leaning phase.

Fig. 6 shows the position after the 33rd move and no objects but all $np. During a game play, RLS-Othello constructs a left recursive tree which consists of $np's. After the 33rd move, the value of the slot *result* of knowledge piece $np42 becomes 0.0 . RLS recognizes that the whole knowledge is faulty and must be modified. RLS suspends the execution of RLS-Othello and transfer its control to the leaning phase.

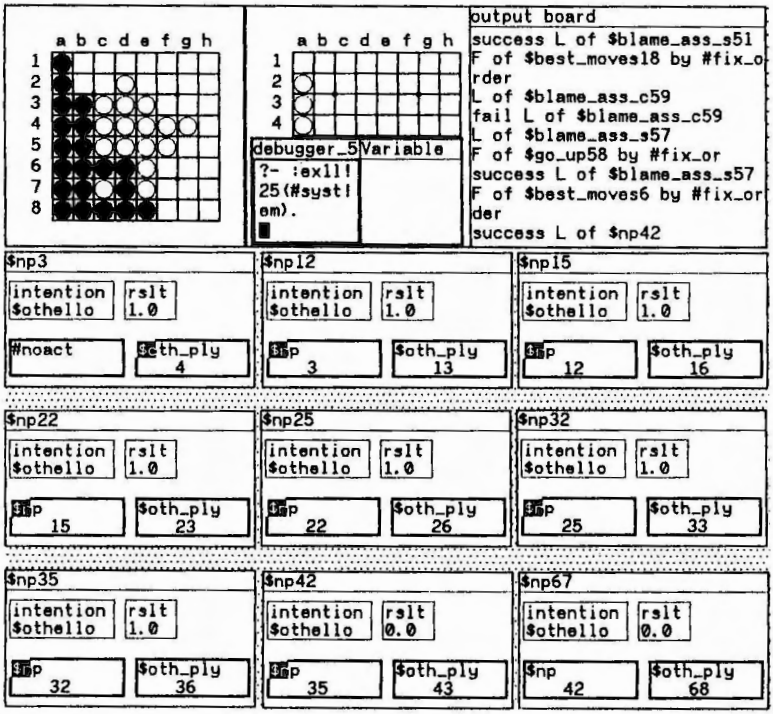


Fig. 6 All $np generated by the moves 26 - 32

5. AN EXAMPLE OF SIMPLE LEARNING

The learning of RLS is a kind of learning through problem solving. Since the model differs from conventional models of knowledge representation and learning, it is desirable to introduce how the learning of RLS proceeds. RLS begins learning when RLS recognizes that the execution of some procedural knowledge failed to realize its intention. It means that the whole knowledge of a specific application domain is faulty. RLS stops learning when RLS has assigned the blame to a certain knowledge piece and fixed it. What are obtained by the learning are modified knowledge pieces or newly created knowledge pieces. Usual way to modify a knowledge piece is to change its status (e.g. values of slots) or to replace its sub-knowledge pieces with other knowledge pieces. It is expected that the learning causes RLS to have an improved knowledge of the specific domain and to show more improved action in the future execution. We consider that this kind of learning is especially necessary in case of decision making problems like game playing, since knowledge of such problems is very complicated and the problem of the blame assignment is very important.

The principles of RLS's learning are as follows:

(a) MKG generates meta-knowledge when a piece of target knowledge is seemed to be faulty. When a faulty knowledge piece is of S type, MKG generates Meta-knowledge corresponding to S type. When it is of C type, MKG generates meta-knowledge corresponds to C type, etc.

(b) The description RLS/K and its action RLS/A of meta-knowledge are not different from those of the target knowledge, so that the execution mechanism and the learning mechanism of meta-knowledge can be the same as those of the target knowledge.

(c) The learning process is divided into two phase: (1) blame assignment phase and (2) fixing phase. Firstly, RLS tries to find a knowledge piece which seems to be the cause of the fault. Then RLS tries to fix the knowledge piece.

Now we show an example of learning for the game play shown in Section 4. The object of this example is to show that RLS can assign blame to a certain knowledge piece corresponding to a move several moves before the last move and fix it only by the execution mechanism defined by each type and the meta-knowledge generation mechanism defined by MKG. Not only meta-knowledge but also meta-knowledge of meta-knowledge (meta-meta knowledge) is generated and employed. RLS-Othello becomes to make another move after the learning. This is not, however, significant learning, since no substantially new knowledge is created.

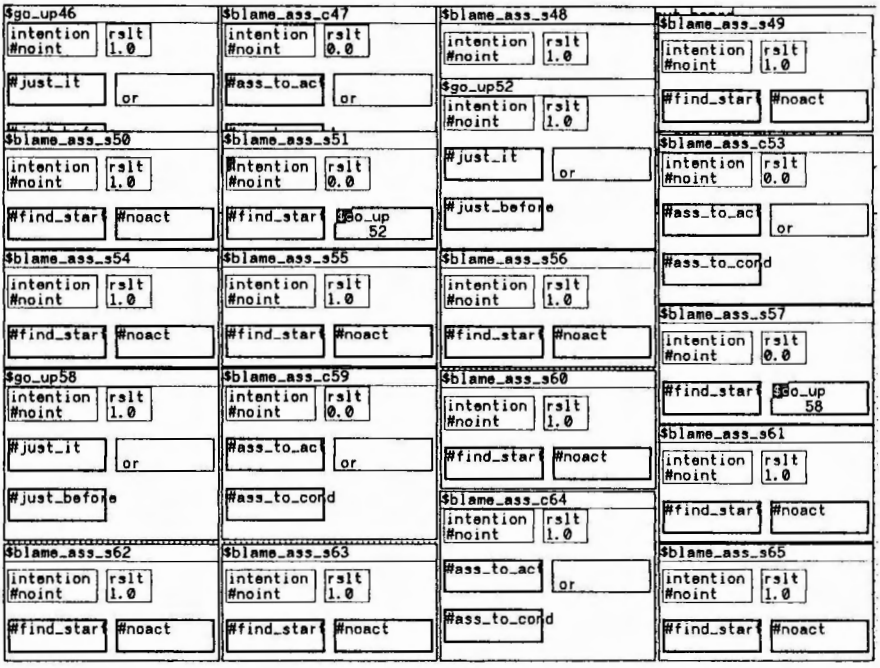


Fig. 7 Generated meta-knowledge pieces

In Fig.6 and Fig.7, you see an example of very simple learning. In this example, meta-knowledge only for target knowledge of S type, C type, OR type, or ORDER type is generated. Corresponding meta-knowledge objects for blame assignment are $blame_ass_s, $blame_ass_c, #blame_ass_or, and #blame_ass_order, respectively. First two of them are instance objects and the rest ones are class objects. Their functions are as follows:

(a) blame_ass_s

Meta knowledge $blame_ass_s assigns blame to the last knowledge piece of the target knowledge at first. If the assigned piece is proved to be not guilty, then $blame_ass_s tries to assign blame to the previous piece, and so on. $blame_ass_s is a S type knowledge piece and consists of two knowledge pieces, #find_start and $go_up. Knowledge piece $go_up is OR type, and its function is to assign blame to the former pieces.

(b) blame_ass_c
Meta knowledge $blame_ass_c is an OR type knowledge piece and assigns blame to the action part of the target knowledge at first. If the assigned piece is proved to be not guilty, then $blame_ass_c may assign blame to the condition part in the future.

(c) blame_ass_or
Meta knowledge #blame_ass_or assigns blame to the branch of the target knowledge piece which has just been executed.

(d) blame_ass_order
At the first stage, meta knowledge #blame_ass_order assigns blame to the branch of the target knowledge piece which has just been executed.

In order to do fixing, meta knowledge objects for target knowledge pieces of OR type and ORDER type are prepared here. They are #fix_or and #fix_order, respectively. Either one is a knowledge piece of Ptype. Their function is very simple. When #fix_or or #fix_order is executed, they select the next branch of the target knowledge piece.

```
output board
L of $np42                                  (1)
F of $best_moves38 by #fix_o                (2)
rder
L of $blame_ass_c47                         (3)
fail L of $blame_ass_c47                    (4)
L of $blame_ass_s45                         (5)
F of $go_up46 by #fix_or                    (6)
success L of $blame_ass_s45                 (7)
F of $best_moves28 by #fix_o                (8)
rder
L of $blame_ass_c53                         (9)
fail L of $blame_ass_c53                   (10)
L of $blame_ass_s51                        (11)
F of $go_up52 by #fix_or                   (12)
success L of $blame_ass_s51                (13)
F of $best_moves18 by #fix_o               (14)
rder
L of $blame_ass_c59                        (15)
fail L of $blame_ass_c59                   (16)
L of $blame_ass_s57                        (17)
F of $go_up58 by #fix_or                   (18)
success L of $blame_ass_s57                (19)
F of $best_moves6 by #fix_or               (20)
der
success L of $np42                         (21)
```

Fig. 8 Output of the learning process

Table 1 Knowledge piece $np and meta knowledge pieces

piece	level	type	target	parts
$np42	0	S	---	$np35 $oth_ply43
$np35	0	S	---	$np32 $oth_ply36
$np32	0	S	---	$np25 $oth_ply33
$np25	0	S	---	$np22 $oth_ply26
$np22	0	S	---	$np15 $oth_ply23
$np15	0	S	---	$np12 $oth_ply16
$np12	0	S	---	$np3 $oth_ply13
$np3	0	S	---	#noact $oth_ply4
$blame_ass_s44	1	S	$np42	#find_start #noact
$blame_ass_s45	1	S	$np35	#find_start $go_up46
$go_up46	1	or	$np35	#just_it #just_before
$blame_ass_c47	1	or	$oth_ply36	#ass_to_act #ass_to_cond
$blame_ass_s48	1	S	$psi_ply37	#find_start #noact
$blame_ass_s49	2	S	$blame_ass_s45	#find_start #noact
$blame_ass_s50	1	S	$np32	#find_start $noact
$blame_ass_s51	1	S	$np25	#find_start $go_up52
$go_up52	1	or	$np25	#just_it #just_before
$blame_ass_c53	1	or	$oth_ply26	#ass_to_act #ass_to_cond
$blame_ass_s54	1	S	$psi_ply27	#find_start #noact
$blame_ass_s55	2	S	$blame_ass_s51	#find_start #noact
$blame_ass_s56	1	S	$np22	#find_start $noact
$blame_ass_s57	1	S	$np15	#find_start $go_up58
$go_up58	1	or	$np15	#just_it #just_before
$blame_ass_c59	1	or	$oth_ply16	#ass_to_act #ass_to_cond
$blame_ass_s60	1	S	$psi_psy17	#find_start #noact
$blame_ass_s61	2	S	$blame_ass_s57	#find_start #noact
$blame_ass_s62	1	S	$np12	#find_start $noact
$blame_ass_s63	1	S	$np3	#find_start $noact
$blame_ass_c64	1	or	$oth_ply4	#ass_to_act #ass_to_cond

Fig.7 shows some of the generated meta-knowledge pieces during the learning process. It shows all kinds of meta-knowledge pieces including $go_up which are the sub-knowledge pieces of blame_ass_s. Table 1 shows all $np and all the instance objects of meta level. The column *level* shows the level of meta knowledge. The level of target knowledge is 0. The level of meta-knowledge is 1. And the level of meta-meta-knowledge (i.e. M^2) is 2. In this example, $blame_ass_s49, $blame_ass_s55, and $blame_ass_s61 are M^2. The column *target* shows the knowledge pieces which are the target of the learning. The column *parts* shows the components which the knowledge piece consists of. Fig. 8 shows the output of RLS on an window named *output board*. Letters "L" and "F" denote learning and fixing, respectively. In this example, long process of blame assignment and fixing continued, and at last leaning of $np42 succeeded.

We explain the details of the learning process referring the each output line by the number attached to the right side of the output in Fig. 8.

(1) Learning of $np42 started. RLS generated meta knowledge $blame_ass_s44 for $np42. Since the latter part of $np42 was a man's ply, the meta knowledge checked $np35 (the former part of $np42).

(2) RLS generated $blame_ass_s45 for $np35. The meta-knowledge first checked $oth_ply36 (the latter part of $np35). It checked into $oth_ply36. RLS generated $blame_ass_c47 for $oth_ply36 and $blame_ass_s48 for $psi_ply37. RLS assumed to assign the blame to $best_moves38 and tried to fix it by #fix_order.

(3) But $best_moves38 has no alternatives. RLS backtracked to the level of $oth_ply36. Its meta-knowledge $blame_ass_c47 had to assign the blame to $oth_ply36 but failed. So RLS started the learning of $blame_ass_c47.

(4) Knowledge piece $blame_ass_c47 is OR type and its meta knowledge is a class object #blame_ass_or. The class object only had to select another alternative of $blame_ass_c47 but it failed to do so.

(5) RLS then backtracked to the level of $np35. RLS recognized that it had failed to do the action of $blame_ass_s45 for $np35. RLS then started the learning of $blame_ass_s45 and generated the meta-meta-knowledge $blame_ass_s49 for $blame_ass_s45.

(6) The meta meta knowledge assigned the blame (of $blame_ass_s45) to $go_up46 (the latter part of $blame_ass_s45) and tried to fix it.

(7) RLS succeeded in the fixing. That is, the second branch of $go_up46, which is the second part of $blame_ass_s45, was employed to execute. It caused the shift of the viewpoint to the previous move. Learning of $np32 started.

(8) - (13) These are the same as (2) - (7) except the viewpoint.

(14) - (19) These are also the same as (2) - (7) except the viewpoint.

(20) The knowledge piece $best_move6, the innermost part of $np3, has another alternative. Meta knowledge #fix_order made the alternative move of $best_moves6 to be selected in the future.

(21) Finally, RLS succeeded in learning of $np42.

The knowledge pieces $np blame-assigned are denoted by small black squares in the upper left corner of rectangles in Fig. 6. The meta-knowledge-pieces blame-assigned by M^2 are also denoted in Fig. 7. The substantial effect is the "fixing of $best_moves6 by #fix_order" shown in line (20) in Fig. 9. From this learning process, RLS-Othello became not to make the same move (b8) at the same position in the future. It would make the alternative move a7.

You might think that the above process is unnecessarily complicated to do such an elementary learning. But we would like to note that this total process is the result defined only by the general structure of RLS which itself has only very simple mechanism of execution and generation. It can be expected that more sophisticated learning is possible when each mechanism supplied by each type of knowledge is improved.

We consider that RLS will be a good tool to realize a learning capability for decision making programs like OTL.PLG [7] which we think is the first high-level knowledge-oriented game playing program.

6. CONCLUSION

We described a new model of the learning of procedural knowledge and the system RLS. We also showed its knowledge representation and its execution of knowledge. The very elementary learning process was shown in Chapter 5. RLS operates very well in the example. We consider that the example shows the effectiveness of the recursive structure of RLS for learning.

In the example, the learning is not an ideal one, because the alternative move a7 in place of b8 is also a bad move. The truly bad move exists before the move b8. RLS must continue learning by playing the game many times and fixing bad knowledge pieces utilizing more sophisticated meta-knowledge. RLS needs the ability of generating meta-knowledge to modify MAX type or TOTAL type knowledge pieces in addition to S, C, OR, and ORDER types. It is expected that various conventional learning algorithms including the five methods in [2] can be incorporated into RLS. We consider that the framework of RLS has potentials to incorporate many sophisticated learning methods to realize very flexible learning systems.

ACKNOWLEDGMENT

We would like to acknowledge the guidance and support of Dr. Nakajima. This research is a cooperative research with ICOT. Dr. Uchida and his group supported this research in my PSI machine utilization. We would like to thank the members of Machine Inference Section of Electrotechnical Laboratory for many useful comments.

REFERENCES

[1]Lenat, D.B.: The role of heuristics in learning by discovery: three case studies, in: Machine Learning, pp.243-306, Tioga, Palo Alto (1983).

[2]Hayes-Roth, F.: Using proofs and refutations to learn from experience, in: Machine Learning, pp.221-240, Tioga, Palo Alto (1983).

[3]Tajima, M.: A Learning Oriented Procedure Representation and Conception of a System, Framework of AI system symposium (1977) (in Japanese).

[4]Rosenbloom, P.S.: A World-Championship-Level Othello Program, Artif. Intell., 19, 3, pp.279-320 (1982).

[5]ICOT: ESP Manual of Small Sized PSI (1988) (in Japanese).

[6]Tajima, M.: Generation of Left Recursive Structure by Right Recursive Execution, Proc. 38th IPSJ Conference (1989) (in Japanese).

[7]Tajima, M.: OTL.PLG -- A Knowledge Oriented Program of the Game of Othello, Trans. of IPSJ, 25, 4, pp.597-605 (1984) (in Japanese).

KNOWLEDGE REPRESENTATION SERVERS: A GENERIC TECHNOLOGY FOR KNOWLEDGE ACQUISITION AND KNOWLEDGE-BASED SYSTEMS

Brian R Gaines, Killam Memorial Chair, & Director, Knowledge Science Institute

University of Calgary, Alberta, Canada T2N 1N4

gaines@cpsc.ucalgary.ca

The semantics and implementation structure of a knowledge representation server in the KL-ONE, KRYPTON, CLASSIC family are described. The server is designed as an open architecture module that can be used as a stand-alone service or embedded in other systems. It accepts the constraints necessary to make subsumption and recognition tractable, and maintains a careful distinction between definitions and assertions. It is implemented as a class library in an object-oriented language using generic, reusable objects. The approach taken to the integration of the server with external knowledge representation servers of similar or dissimilar types is analyzed. The server supports default reasoning through computation of a three-valued subsumption relation that is able to determine the possibility of further inferences if more assertions are made. It is now possible to develop knowledge representation servers on a par with floating-point arithmetic units and numeric libraries, as modules with well-defined functionality and fast, reliable performance. Such servers provide a generic knowledge representation technology for knowledge acquisition and knowledge-based systems.

1 INTRODUCTION

Adequate knowledge representation is an essential requirement for all areas of artificial intelligence research and development. In recent years there have been major advances in the theory and practice of KR systems originating from semantic nets. In particular the series of KR schemes and tools commencing with KL-ONE [6], developing through KRYPTON [4] and currently culminating in systems such as CLASSIC [2, 3] and LOOM [30] has reached a maturity of technology which offers the promise of KR 'utilities' or 'services' in Levesque's [27] terminology. The logical foundations of the subsumption relation, techniques for its correct and complete calculation, and the interaction between representation power and the tractability of subsumption algorithms have been widely studied [5, 37, 42] and are becoming reasonably well-defined. It is now feasible to develop KR servers on a par with floating-point arithmetic units and numeric libraries, as software (and perhaps ultimately hardware) modules with well-defined functionality and fast, reliable performance. As with arithmetic units, such servers by no means solve all the problems of a particular application domain, but they do greatly reduce the burden of system development, allowing effort to be focused on the specifics of particular systems rather than on what should be general utilities.

The 'application' domains have high degrees of generality. Subsumption-based KR schemes offer many-sorted logics and operations upon them that are known to be important components of effective theorem provers [21], machine learning algorithms [29], default reasoning systems [41], commonsense reasoning [26], knowledge acquisition [36] and so on. The motivation for the work in this paper has been to make use of developments in knowledge representation to place a clean, fast and effective KR server at the heart of a very general knowledge support system combining a wide and heterogeneous range of knowledge acquisition techniques [14]. One of the frustrations common to many areas of research in AI is the amount of effort necessary to develop adequate general-purpose KR systems before the 'real work' can begin. Similar considerations apply to porting systems that have a high degree of generality of potential application but whose KR sub-systems have been written to be domain-specific and are inextricably intertwined with other functionality.

This paper reports on the implementation of a KR server, KRS, that took CLASSIC as its initial model, meticulously maintained the terminology/assertion distinction of KRYPTON, and emphasized software engineering goals of clarity, portability, modularity, scalability and integratibility. It presents guidelines for the development of KR servers that will simplify the tasks of others in implementing such systems. Even with these practical goals, features of the implementation raise a number of theoretical issues, some of which are deep both in terms of logic and philosophy. Subsumption-based KR servers are essentially implementations of intensional logics [28, 47] of abstract objects [10, 33] and design decisions at the coding level have to be guided by a proper implementation of such logics, some aspects of which are still open questions [32, 48]. Well-structured computer code is itself a language for communicating about the relevant theoretical issues. Minimal implementations in code, data and time resources usually correspond to natural underlying algebraic and logical structures, and non-obvious design decisions often correspond to underlying philosophical problems.

2 KNOWLEDGE REPRESENTATION SERVER REQUIREMENTS

The design goals for a KR server are:

- To offer a representation system that is general-purpose and rich enough in its descriptive capabilities to serve in a wide variety of applications.
- To derive correctly and completely the deductive consequences of constraints expressed in the representation, accepting such limitations on the constraints expressible as are necessary to achieve this.
- To act as a light-weight service to other modules that use heuristics to overcome these limitations at the expense of correctness or completeness.
- To be cleanly structured so that data functionality, structures, and algorithms are visible and easily checked.
- To have a well-defined interface that is callable from a wide range of languages so that the server is readily embedded in other systems.
- To support a number of knowledge bases, closely and loosely coupled, homogeneous and heterogeneous—this raises significant implementation and theoretical issues that will be discussed in detail.
- To support both small and large knowledge bases, with ease of modification of some underlying data structures if necessary.
- To be portable at the code and dataset level, both conceptually and practically.
- To re-use as much existing general-purpose code as possible, for example for set manipulation, and to make as much of the additional code as possible re-usable for other

purposes—maintenance and portability are both supported by the use and generation of building bricks rather than special-purpose code.

- To offer a range of supporting utilities, such as parsers and graphers, on a modular basis as options separable from the KR kernel.

The initial implementation of KRS has been in a dialect of C++ but most of the design considerations apply to any implementation and we have parallel developments in SmallTalk, OBJ3, Lisp and Prolog. The use of an object-oriented language is important for its modularity rather than any intrinsic contribution to the object-orientation of the resultant knowledge base [9]. However, the data structures and methods of the implementation objects are very important in the role of the code as a communication medium.

3 FUNDAMENTAL MODEL OF THE SERVER

There are at least four conceptual frameworks for the design of a KR server:

- A logical framework which maps concepts, roles and individuals into some logical system based on first-order predicate calculus, modal logic, free logic [24, 25, 32] or some form of intensional logic of abstract objects [10, 39, 48]—these logical links are significant but none of those available matches exactly the features of a CLASSIC-like system—however, the known capabilities and problems of these logics generate very important tests of the design.
- A philosophical framework which views the system as providing ontological and epistemological facilities for representing and reasoning about the world—again inadequate, but detailed metaphysical schemas such as that of Grossman [20] provide a test for missing features.
- A cognitive framework which views the system as emulating human knowledge and belief systems and mental processes—again inadequate but providing a range of test data including mentalistic analysis of logics [23, 46] and models of the dynamics of belief systems [27].
- An algebraic framework which views the system as expressing and manipulating constraints on sets.

The algebraic framework currently provides the simplest definition capturing the semantics and structure of a CLASSIC-like system. The basic structures are:

- A set, I, of *individuals*
- A set, F, of *fillers* defined as $\mathcal{P}\mathrm{I} \cup \omega$, where $\mathcal{P}\mathrm{I}$ is the powerset of I and ω represents an open filler
- A set, R, of *roles* with cardinality r
- A *filler values* mapping, V, from I to the Cartesian product, P, of r filler sets, $V : I \rightarrow \prod_{r} F$, with a projection operator R_n from $p \in P$ to its n'th component, $p_n \in \mathcal{P}\mathrm{I}$ [$R_nV(i)$ is the set of role fillers for role R_n in individual i]
- A poset, C, of *concepts* ordered such that for some $c, d \in C$, $c < d$ [the order relation is used to represent the lack of self-reference in concept definitions]
- A *filler restrictions* mapping, S, from C to the Cartesian product, Q, of r members of C, $S : C \rightarrow \prod_{r} C$, with a projection operator R_n from $c \in C$ to its n'th component, $c_n \in C$, and

such that $R_nS(c) < c$ [$R_nS(c)$ is the concept applicable to role fillers for role R_n in concept c]

- A set of *constraints*, E, on subsets of I, with a basic recognition relation, ρ, between constraints in E and subsets of I such that there is a well-defined entailment relation, $\Rightarrow$, generated on E by the condition that $\forall$ e, f $\in$ E, e $\Rightarrow$ f iff $\forall$ $p \in \mathcal{P}$, e$\rho p \supset$ fρp [the constraints specify some test of the existence, cardinality or the set membership of the subset, or that all members satisfy one set of arbitrarily assertable predicate (primitives) and do not satisfy another set of these predicates (those primitives disjoint from the first set)—note that $\forall$ e, e$\rho\omega$ is false, that is, open fillers cannot satisfy a constraint]
- A *constraints* mapping, D, from C to E [D(c) is the primary constraint applicable to a subset of I, p, to be recognized as being a c with secondary constraints defined recursively for the role filler subsets of i $\in$ p, that $R_nV(i)$ is recognized as being a $R_nS(c)$]
- A *templets* mapping, T, from I to C [T(i)$\in$ C is the concept providing an intensional templet for i$\in$ I]
- A *subsumption* relation, $\Rightarrow$, extending the entailment relation on constraints to one on concepts such that $\forall$c, d $\in$ C, c $\Rightarrow$ d iff D(c) $\Rightarrow$ D(d) and for $1 \leq n \leq r$ $R_nS(c) \Rightarrow R_nS(d)$ [the partial order, <, on C ensures this recursive definition is determinate]
- A *recognition* relation, ρ, between concepts in C and subsets of I extending the basic recognition relation between E and $\mathcal{P}$ such that $\forall c \in C$, $p \in \mathcal{P}$, cρp iff D(T(i)) $\Rightarrow$ D(c) or this is not so but D(c)ρp and for $1 \leq n \leq r$ $R_nS(T(i)) \Rightarrow R_nS(c)$ or a particular entailment is not so but $R_nS(c)\rho R_nV(i)$
- An auxiliary *recognition* relation, ρ, between concepts in C and individuals I extending that between C and $\mathcal{P}$ such that $\forall c \in C$, i $\in$ I, cρi iff $\exists$ $p \in \mathcal{P}$ such that i $\in$ p and cρp [this is necessary because recognition has been defined in terms of subsets of I because of cardinality and set inclusion constraints]

Subsumption is expressed as an intensional relation to allow it to be computed for concepts that do not have extensions and correspond to 'impossible objects' [39]. The definition corresponds to the normal one for concepts with extensions. The tractability of subsumption in CLASSIC-like representation systems corresponds to the restriction of constraints to be expressible in a non-interacting canonical form such that they may be independently tested. The definitions of the subsumption and recognition relations correspond to a test of the *necessity* of an individual being recognized by one concept if it is recognized by another. They also allow for the *impossibility* of an individual being recognized by a concept (for example, typically because of cardinality or disjoint primitive constraints). Calculating this is important in plausible reasoning with open fillers in checking whether additional information might change the current conclusion.

Note that cardinality constraints have been included in the definition of concepts and that concepts are taken to apply to sets of individuals. This is helpful in implementation because a single subsumption calculation may be applied when checking roles without a separate check of cardinality constraints. It also allows constraints on slot fillers involving cardinality to be fully expressed as named concepts. The algebraic definition is useful computationally because the primitives of sets, and constraints, lead to simple data structures and "obvious" algorithms. It is also a useful framework for proving completeness and tractability of subsumption algorithms [40].

4 KNOWLEDGE REPRESENTATION SERVER ARCHITECTURE

Figure 1 shows the architecture of KRS. It is designed to manage multiple standard knowledge bases and to interface to other servers either of the same or differing architectures. The interface protocols to allow the standard server to integrate other servers fully into its subsumption calculations are discussed in a later section.

The various sub-modules attached to the server are options offering specific additional functionality. The modules are (reading clockwise from the top left):

- Interface modules to other KR servers and knowledge bases, including databases.
- A hypermedia module allowing informal knowledge structures in text and images to be captured, accessed and linked. The linkage structure itself is held as a knowledge base.
- A text analysis module allowing documents to be analyzed in terms of word usage, and associations between significant words to be graphed—based on TEXAN in KSS0 [12, 15]. This enables protocols and technical documents to be used to initiate knowledge acquisition.
- A repertory grid expertise transfer module allowing graphic definition of concepts and graphic creation and editing of individuals—based on the elicitation screens of KSS0.
- A conceptual clustering module allowing interactive definition of new concepts—based on the hierarchical and spatial clustering from KSS0.
- A knowledge editing module allowing the interactive development and editing of knowledge structures through a visual language.
- A conceptual induction module creating rules about specified subsets of individuals and transforming them to a minimal set of concepts and default rules—based on the Induct algorithm [13].
- A problem solving module supporting frame, rule and case-based inference from the knowledge structures.
- A grapher laying out specified parts of the concept subsumption graph, concept structures and individual structures—based on an incremental layout algorithm by Watanabe [45] that can be used interactively to support the production of clear visual knowledge structures.
- A language interface accepting and generating definitions and assertions in formal knowledge representation languages, both textual and visual.

The language interface supports a table-driven top-down recursive-descent compiler for CLASSIC-like languages. The tables can be changed dynamically in a prelude to the compiler allowing a syntax that makes clear distinctions between definitions and assertions to be used in examples, and a less verbose language to be used for system development. The same tables are used to output definitions and assertions in the language so that graphically entered concepts and individuals may be described, stored and ported to other systems in a textual form. Like all the peripheral modules, the language interface module is easily changed for another, and there is no implied commitment to our current language.

The language interface also supports visual knowledge representation languages similar to those of the graphic knowledge editor but allowing offline drawings to be parsed as formal specifications. This is particularly useful for standard ontologies which may be presented effectively using drawing tools but where the drawing themselves are also the formal specification.

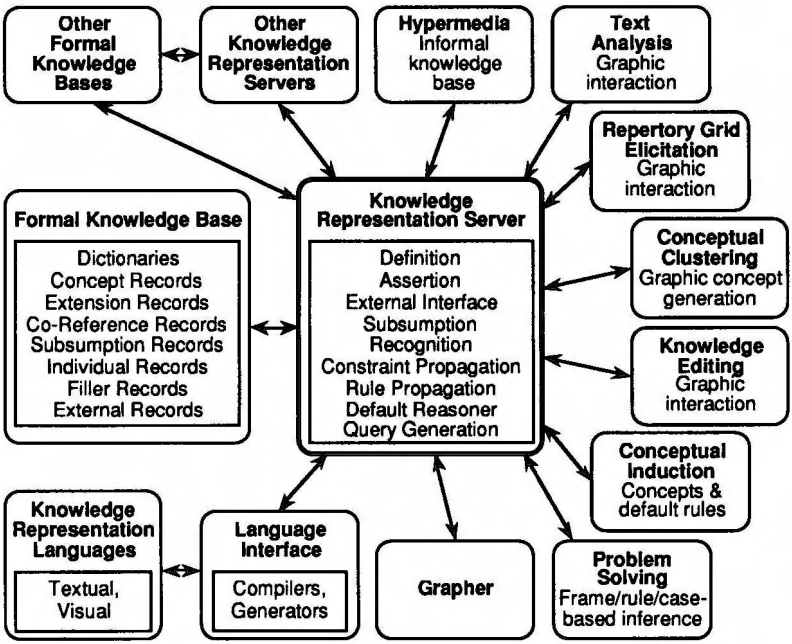


Fig.1 Architecture of the knowledge representation server

5 DATA STRUCTURES

Figure 2 shows the objects used in the server implementation and their data structures. Our basic class library provides variable-length record objects which are sub-classed to provide dictionaries, sets and record objects for concept and individual records. The objects are:

- *Concept records* hold the constraint defining a concept: its primitives and those disjoint from them; its extensions as the subsets of individuals that must be, and may be, included in any subset recognized; its cardinality and co-reference constraints on any subset recognized; its rule and release concepts, if any, which will be asserted of any individual recognized, or impossible to recognize, respectively; and, for each role constrained in the concept, the concept constraining it.
- *Individual records* hold the templet concept defining intensional constraints upon an individual and, for each role constrained in the templet or filled in the individual, a specification of the role fillers.
- *Filler records* hold subsets of individuals that fill slots.
- *Dictionaries* recognize concept, role and individual names and generate an accession number used to index other structures. Syntactically there is a clear separation between concepts, roles and individuals and a lexical item can occur in more than one dictionary with different meanings.
- *Extension records* hold subsets of individuals, typically those that may occur in a role filling set and those that must occur in it.
- *Data records* hold subsets of external individuals that are explicitly recognized by the server, notably integers, reals, dates and strings.

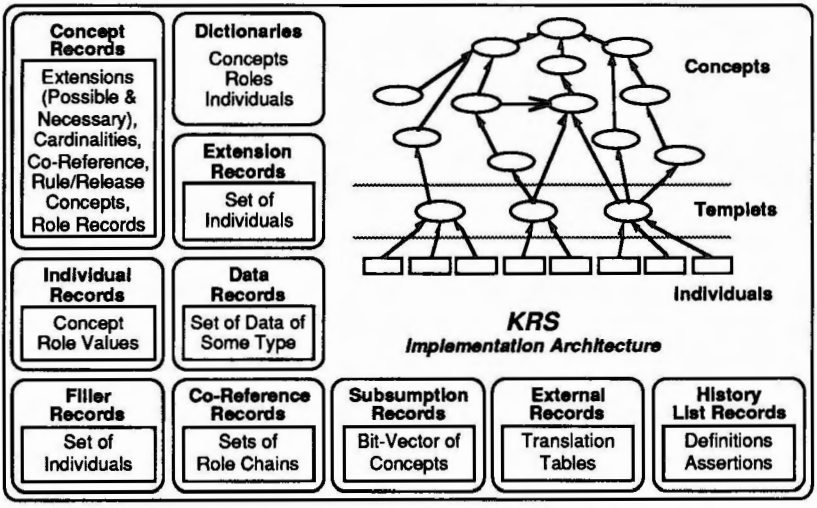


Fig.2 Data structures in the knowledge representation server

- *Co-reference records* hold equivalence subsets of role chains expressing co-reference constraints.
- *Subsumption records* hold computed subsumption relations on a three-valued basis, that one concept subsumes another, does not, or cannot.
- *External records* hold tables translating between internal and external concepts and individuals.
- *History list records* keep track of explicit and implicit definitions and assertions in such a way that the knowledge base may be rolled back to any previous state.

These data structures correspond very closely to the algebraic model of the server already described. A number of optimizations are applied in the implementation to minimize storage and maximize speed. For example, special codes are used for the empty, universal and singleton sets such that the set objects never store them, and subsumption is calculated in such a way that only a fast lookup is needed to check subsumption between role concepts rather than a recursive call. One advantage of the object implementation is that these optimizations can be put in override methods in sub-classes so that the kernel implementation is small and easily checked but potentially slow, and the full implementation can be checked against the kernel for correctness and optimized for speed.

The server supports the propagation of the consequences of definitions, assertions and retractions at several different levels:

- Extensional constraints in definitions can cause cardinality constraints to be adjusted.
- Co-reference constraints in definitions cause the corresponding role constraints at the ends of the role chains to be merged.
- Assertion and retraction of role fillers causes the role constraints to propagate to the filler individuals.
- Assertion and retraction of individual constraints causes any resulting role constraints to propagate to the existing filler individuals.
- The closure of a set of role fillers is computed based on defined or asserted constraints.

- Recognition of an individual by a concept with an attached rule causes the consequential concept of the rule to be asserted as applying to the individual.

The last three propagation conditions can interact and the server continues propagation until a fixed point is reached (which can be shown to be unique and well-defined). A good test of the server is MacGregor's [30] example of how such propagation enables the recognition of recursively defined concepts, such as lists, even though self-reference and forward reference is not allowed in concept definitions. The definition/assertion separation and the constraint propagation also satisfy Frixione, Gaglio and Spinelli's [11] requirements for the proper treatment of individual intensions.

6 INTEGRATION OF EXTERNAL KNOWLEDGE BASES

The practical requirements of knowledge acquisition, particularly the need to support elicitation from multiple experts in multiple interacting domains [15], and the support of heterogeneous knowledge bases combining many different sources and media, dictated that we pay careful attention to knowledge base integration. CLASSIC allows external individuals, such as integers, to be used as slot fillers and provides a TEST primitive that allows external concepts to be used to constrain these fillers, but does not expect external individuals to have roles or external concepts to support subsumption. The server described here supports both these extensions but does not require them, thus allowing it to operate correctly with a range of external knowledge support servers including other instances of itself.

It was a design requirement for KRS that a knowledge base within the system be partitionable into internal and external knowledge bases in such a way that no functionality changed. It is assumed that external knowledge bases have their own interfaces to which messages may be passed, and that they have various levels of capability of responding to queries. The keyword EXTERNAL in the server language is used to indicate messages to external knowledge bases and internal references to external concepts and individuals. It is followed by the name of the external knowledge base referenced, the 'realm' in CLASSIC terminology. The record structures of Figure 2 make provision for references to external concepts and individuals. An implementation distinction is made between slots filled with internal individuals only, those filled with external individuals in a single realm only (which are managed by the server for that realm) and heterogeneous fillers where individuals are mixed across realms (which are managed by the external records object as sets of individuals tagged by realm).

This system is effective for knowledge bases with individuals whose role fillers are from other knowledge bases. The only additional complication is that of supporting individuals part of whose record structures are maintained in another realm. For example, one might keep knowledge about cars in a car knowledge base, but their general roles as transportation vehicles and physical objects might be inherited from, and managed by, a more general knowledge base. It is not possible to determine the knowledge base referenced by context since roles may be designated for fillers that have not previously been specified in the templet, and the same role name may occur with different meanings in interacting knowledge bases.

The external interface assumes that external knowledge bases can answer queries about subsumption and recognition. If they cannot the queries are answered at the interface in terms of identity only which is the default in CLASSIC's TEST. The correctness and completeness of subsumption and recognition can be maintained provided the external server is able to support it within its realm. If not, only correctness will be maintained.

7 EDITING CONCEPT DEFINITIONS

KL-ONE-style knowledge structures are notoriously difficult to edit because they are based on the 'top-down' definition of general concepts and their stepwise refinement to more specific concepts. This is reasonable in well-established domains where the ontology has been developed as part of a professional discipline, but it is rarely a suitable basis for knowledge acquisition in novel domains. Experts often develop their knowledge structures 'bottom-up' from cases through commonality to general ontologies, and find a 'top-down' approach very unnatural. There are a number of ways of overcoming this problem—the conceptual clustering and induction tools develop concept hierarchies 'bottom-up' from cases—but general concept editing services are difficult to define.

The problem is to design an interface allowing the person editing to adequately express his or her intentions. For example, if a concept is edited it is simple enough to maintain referential integrity by disallowing use of concepts defined after it in the definition, and even this constraint can be relaxed in a more intelligent editor that places a new or edited concept in an appropriate place in the referential order if one exists. However, it is difficult to determine whether the user intends, when editing, that subsumption relations be preserved and hence that edits should be propagated, and, if so, whether the user really wants full propagation or the option of checking what is occurring.

The server currently provides an 'undo' and controlled 'redo' facility based on the history list. The knowledge-base can be rolled back to any previous state and then rolled forward with a modified history list. This is clean in implementation and reasonably ergonomic in terms of user activity. It does not satisfy all editing requirements but we do not yet have a good model of what this 'all' might be. An interesting theoretical foundation for the history-list approach in terms of the algebraic model is that an instance of the knowledge base may be taken as an automaton and the subsequent history list as its input-output behavior. There is then a category-theoretic adjunction between the knowledge base and its editing [19]. This approach is particularly insightful because it corresponds to the viewpoint of a user of an existing knowledge base attempting to change its structure by edits and assertions. The various constraints already existing may make the desired state unreachable. There is also a significant psychological literature concerned with this problem, for example, Kelly [22] distinguishes "core constructs" which an individual is highly reluctant to change despite evidence that they are inappropriate. In our terms, this corresponds to edits of early concepts where the propagation effects may be very difficult to control in terms of the user's intentions.

The BLIP knowledge acquisition tool developed at the Technical University of Berlin to support 'sloppy modeling' [35] provides a conceptual system development and editing tool that computes all consequences of definitions dynamically, including induction from cases. For the initial stages of knowledge acquisition even this may be too strong in that experts often generate inconsistent knowledge structures but are not ready to resolve the inconsistencies until they have a developed a fairly large-scale model giving them an overall perspective on the task and domain. The graphic knowledge editing tool shown on the center right of Figure 1 supports this relaxed approach. Knowledge structures can be developed and edited using drawing tools providing a natural visual language. The structures can extend across multiple screens supporting modularity and ontology 'libraries.' Structures can be freely duplicated and definitions may be fragmented across screens. The consistency of the definitions is determined only when they are compiled into a formal knowledge structure and inconsistencies are graphically highlighted.

Figure 3 shows a screen from the graphic knowledge editor on which the minimal five rule default reasoning solution of Cendrowska's [7] contact lens prescription problem has been developed. This screen compiles directly into the set of assertions and rules shown in

Figure 4. The top diagram on the screen defines the concept of a patient with possible attributes and values. The center diagram defines the concepts of a lens recommendation and three potential recommendation. The lower diagram defines the default rules.

The visual language used in Figure 3 is precisely defined. Concepts are ovals, individuals rounded-corner boxes, roles (and annotation) unboxed text. The interpretation of the standard arrows with filled heads is well-defined by the types of the objects at their head and tail. Arrows with open heads denote rules. Lines without arrows connecting concepts denote that the concepts are disjoint. The compiler traces concepts and individuals through their outgoing arrows, builds a dependency table for concepts, highlighting circular links that it will not use, and generates concept definitions and individual assertions. Role objects are only noted when on paths and hence may also be used for annotation. The text analysis module generates such 'annotation' which may then be developed into a formal graphic specification by specifying objects as concepts and individuals.

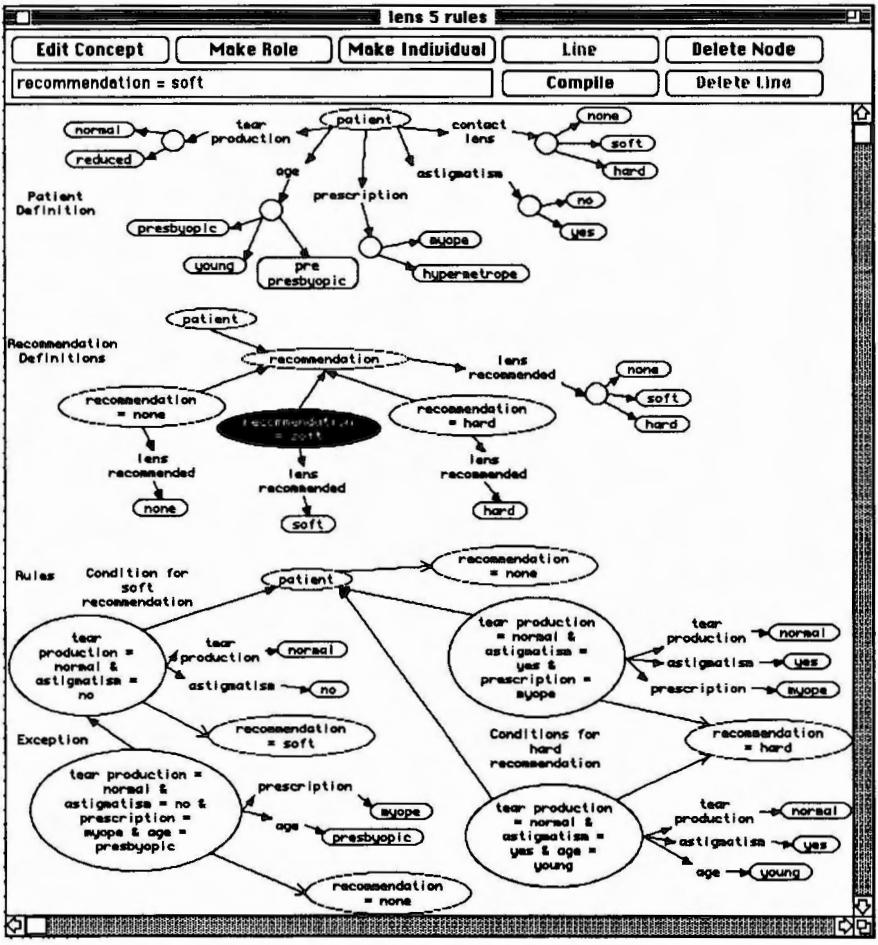


Fig.3 Graphic knowledge editor

```
concept(patient,
  (ALL tear production, (IN normal, reduced))
  (ALL astigmatism, (IN yes, no))
  (ALL prescription, (IN myope, hypermetrope))
  (ALL age, (IN young, pre-presbyopic, presbyopic))
  (ALL contact lens, (IN none, soft, hard))
)
concept(recommendation,
  patient,
  (ALL recommendation, (IN none, soft, hard))
)
concept(recommendation = none,
  recommendation,
  (ALL lens recommended, (INCLUDE none))
)
concept(recommendation = soft,
  recommendation,
  (ALL lens recommended, (INCLUDE soft))
)
concept(recommendation = hard,
  recommendation,
  (ALL lens recommended, (INCLUDE hard))
)
concept(tear production = normal & astigmatism = no,
  patient,
  (ALL tear production, (INCLUDE normal))
  (ALL astigmatism, (INCLUDE no))
)
rule(tear production = normal & astigmatism = no,
  recommendation = soft,
)
concept(tear production = normal & astigmatism = no & prescription = myope & age = presbyopic,
  patient,
  (ALL tear production, (INCLUDE normal))
  (ALL astigmatism, (INCLUDE no))
  (ALL prescription, (INCLUDE myope))
  (ALL age, (INCLUDE presbyopic))
)
rule(tear production = normal & astigmatism = no & prescription = myope & age = presbyopic,
  recommendation = none,
)
rule(patient, recommendation = none)
concept(tear production = normal & astigmatism = yes & age = young,
  patient,
  (ALL tear production, (INCLUDE normal))
  (ALL astigmatism, (INCLUDE yes))
  (ALL age, (INCLUDE young))
)
rule(tear production = normal & astigmatism = yes & age = young,
  recommendation = hard,
)
concept(tear production = normal & astigmatism = yes & prescription = myope,
  patient,
  (ALL tear production, (INCLUDE normal))
  (ALL astigmatism, (INCLUDE yes))
  (ALL prescription, (INCLUDE myope))
)
rule(tear production = normal & astigmatism = yes & prescription = myope,
  recommendation = hard,
)
```

Fig.4 Knowledge structures compiled from the diagrams of Figure 3

By contrast to the problems of consistent editing of concept definitions, the truth maintenance problem of propagating assertions and retractions of constraints and facts about individuals is relatively easy. There are still subtle problems, e.g. the same assertion may have been made a number of times through direct entry and various forms of propagation—is the user attempting to retract the fact or the overt assertion? However, these are easily overcome. The server tags assertions and inferences distinctly and maintains inverse role linkages. This makes it easy to propagate a retraction to potentially affected individuals, reset them and re-inference.

8 DEFAULT REASONING

One important application of the server in knowledge acquisition is to support the restructuring of rules inductively derived from elicited datasets as a minimal set of concepts and rules that can be presented to the expert in an easily understood form for checking and validation. This involves default reasoning since the best understood presentation is usually the most commonly occurring situations as the major rules with the rarer situations as exceptions. CLASSIC in its current definition supports rules but not default reasoning, and there are known problems to simply incorporating exceptions as rules triggered by concepts lower in the subsumption hierarchy [18].

KRS fully supports default reasoning in that rules are checked by tracing the subsumption hierarchy and giving precedence to inferences made at lower levels. Thus, in the knowledge structure defined in Figures 3 and 4, the top level default rule is that an individual recognized as a 'patient' has 'lens recommended' set to 'none.' This is superseded by 'soft' if the patient is recognized by the subsumed concept, 'tear production = normal & astigmatism = no.' This, in its turn, is superseded by 'none' again if the patient is recognized by the subsumed concept, 'tear production = normal & astigmatism = no & prescription = myope & age = presbyopic' (see the bottom left part of Figure 3).

This reasoning is logically correct if all the facts about the patient are known. It only becomes default reasoning if the exception rule cannot be tested because some facts are unknown, for example, the patient's 'prescription' is not known. KRS can indicate whether an inference based on partial information is plausible or exact because it computes a three-valued subsumption relation which allows the system to recognize when no further assertions could make an exception rule trigger. This is termed a 'release' and may be viewed as a rule which triggers when it is impossible for an individual to satisfy a concept except through retraction. Releases may be used to manage the frame problem by discriminating between plausible reasoning based on defaults and exact reasoning based on adequate information. Thus a release attached to the concept 'tear production = normal & astigmatism = no & prescription = myope & age = presbyopic' will fire when either the 'prescription' is not 'myope' or the 'age' is not 'presbyopic' regardless of whether the other information is known or not.

The reasoning algorithm in KRS is efficiently implemented and operates rapidly even with large knowledge bases; for example, the Garvan Institute of Medical Research thyroid diagnosis knowledge base, GARVAN-ES1, consists of 650 rules [8]. The 'ripple-down rules' version of this was moved to KRS and checked on the cornerstone case database of some 650 cases, taking less than one second a case on an Apple Mac IIci.

9 KNOWLEDGE ACQUISITION METHODOLOGIES BASED ON THE SERVER

KSSn is a knowledge support system based on the knowledge representation server which uses the modules shown in Figure 1 to provide an integrated set of knowledge acquisition tools. It is designed to offer the text analysis, graphic elicitation, clustering and induction tools of KSS0 [12, 15], together with the wider range of data types and hierarchical data structures of AQUINAS [1], and the graphic knowledge editing of tools such as ONTOS [34] and CODE [44]. Figure 5 shows a knowledge acquisition methodology based on the appropriate sequence of use of the tools in KSSn.

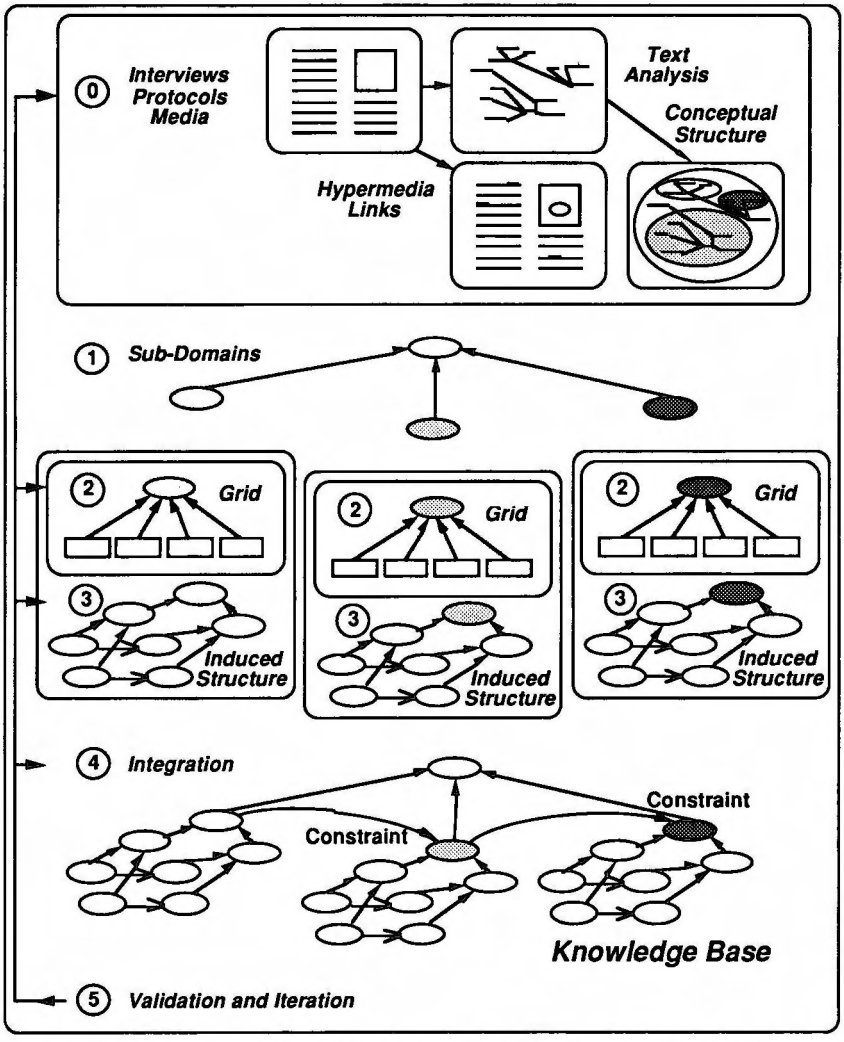


Fig.5 Integration of knowledge acquisition techniques based on KRS

- Stage ⓪ consists of gathering informal knowledge in the hypermedium system.
- Stage ① consists of using informal knowledge to elicit the major coherent sub-domains that together encompass the significant phenomena in the overall domain.
- Stage ② consists of using the repertory grid methodology in each sub-domain to elicit relevant attributes and critical cases.
- Stage ③ consists of using the conceptual induction methodology to derive concepts and rules from the grids.
- Stage ④ consists of linking the sub-domains together, generally by specifying as a constraint in one domain that some role in that domain has as value an individual in another.
- Stage ⑤ consists of testing the overall knowledge base as a performance system and iterating back through any of the earlier stages in order to develop and refine it further.

The architecture of Figure 5 is intended to be illustrative of the way the various knowledge acquisition techniques described naturally combine than a rigid framework for an integrated tool. For example, some sub-domain structures may be developed by direct editing, others by induction from databases, others by text analysis, and others by tools not yet defined or developed. The point is that techniques and tools already available come together naturally in this framework, and it also provides an open architecture for incorporating further techniques and tools. In the longer term, the architecture shown may need major extension, but currently it seems to encompass what is available to us in knowledge acquisition.

The knowledge representation server holds the knowledge structures involved at every stage of this process in a uniform fashion with full access to a range of editing, presentation and documentation tools. The associated modules for text analysis, repertory grid elicitation, conceptual induction, and so on, are relatively small and independent modules that are readily developed, changed and maintained. The use of a common underlying class library for data structures and the user interface supports the provision of a consistent data model and user interface throughout.

10 CONCLUSIONS

The main objective of this paper has been to demonstrate that open architecture knowledge representation servers are straightforward to implement now. The algebraic semantics and computational object structures described should help others undertaking such implementations. The C++ implementation of the server with the functionality and optimizations described is fast and storage efficient, and was easily implemented.

We have run it with knowledge bases of several thousand concepts and individuals from a variety of sources including the GARVAN-ES1 described in Section 7, our sports knowledge bases [17], and a variety of artificial datasets designed to stress the code. Nebel's [38] 'worst-case' example of the combinatorial explosion of anonymous concepts runs without problems—for example, his 20 concept example generates a total of 1,701 concepts and a total data structure of 460 Kbytes.

The space requirements for 100 concepts, 1,000 individuals, with an average of 20 roles of which 5 are multi-valued having an average of 3 slot fillers is 120 KB and scales up nearly linearly with the number of concepts and individuals. The small knowledge bases typical in the early stages of knowledge acquisition take only a few KB.

Keeping subsumption up-to-date appears to be a negligible overhead even with very large numbers of concept. Recognition has not been a problem with the knowledge acquisition datasets that tend to consist of numbers of self-contained knowledge islands. However, it is possible to define knowledge structures that involve substantial propagation over large sets of individuals each time a new assertion is made and this would cause interactivity problems if it was a significant application requirement. This is a domain problem rather than one of server implementation and it would be reasonable to expect it if the system were used to investigate, for example, the mathematics of algebraic structures. Similar problems would arise if configurational problems involving many interacting constraints were represented for solution in the system.

The implications of accepting the restrictions necessary to make subsumption and recognition tractable in the server have still to be investigated in the context of knowledge acquisition. So far, KRS provides such a major increase in representation capabilities compared with previous knowledge acquisition systems that it has not been a major issue. MacGregor [31] and Schmolze [43] have remarked that users want more representation power, not less. Our design philosophy is that the analogy to arithmetic units holds, and it is appropriate to develop clean, complete and fast units with limited capabilities, and then supplement them with other units providing additional functionality ranging from general theorem provers to highly domain-specific modules. This is particularly important in a knowledge acquisition environment where one has to present a clean conceptual model and user interface to users who may never have used a computer before and are attempting to develop overt knowledge structures where none previously existed.

A companion paper [16] has surveyed developments in knowledge acquisition, their foundations, and some of the existing tools. Current third generation research is targeting tools for the 1990's to provide integrated knowledge acquisition environments supporting a wide range of complementary tools and techniques. This paper has described an architecture for such systems based on a general-purpose knowledge representation server. In particular, this architecture attempts to provide the richness of resources required while maintaining the strict software engineering disciplines necessary to the effective development and support of complex systems.

ACKNOWLEDGEMENTS

Financial assistance for this work has been made available by the Natural Sciences and Engineering Research Council of Canada. I am grateful to Mildred Shaw for her critical comments on the first draft of this paper.

REFERENCES

[1] Boose, J.H. & Bradshaw, J.M. (1987) Expertise transfer and complex problems: Using AQUINAS as a knowledge acquisition workbench for knowledge-based systems. **International Journal of Man-Machine Studies, 26,** 3-28.

[2] Borgida, A., Brachman, R.J., McGuiness, D.L. & Resnick, L.A. (1989) CLASSIC: a structural data model for objects. Clifford, J., Lindsay, B. & Maier, D., Eds. **Proceedings of 1989 ACM SIGMOD International Conference on the Management of Data**. pp.58-67. New York: ACM Press.

[3] Brachman, R. J., Borgida, A., McGuinness, D. L., and Resnick, L. A. (1989) The CLASSIC knowledge representation system, or, KL-ONE: The next generation. **Workshop on Formal Aspects of Semantic Networks**. Santa Catalina Island, California (February).

[4] Brachman, R. J., Gilbert, V.P. & Levesque, H. J. (1985) An essential hybrid reasoning system: knowledge and symbol level accounts of KRYPTON. **Proceedings of IJCAI85**. pp.547-551. Los Altos: Morgan Kaufmann.

[5] Brachman, R. J., & Levesque, H. J. (1984) The tractability of subsumption in frame-based description languages. **Proceedings of AAAI-84**. pp.34-37. San Mateo, California: Morgan Kaufmann.

[6] Brachman, R.J. & Schmolze, J. (1985) An overview of the KL-ONE knowledge representation system. **Cognitive Science, 9**(2) 171-216 (April-June).

[7] Cendrowska, J. (1987) An algorithm for inducing modular rules. **International Journal of Man-Machine Studies 27** (4), 349-370 (October).

[8] Compton, P. & Jansen, R. (1989) A philosophical basis for knowledge acquisition. **Proceedings of the Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems (EKAW'89)** pp.75-89. Paris (July).

[9] De Troyer, O., Keustermans, J. & Meersman, R. (1986) How helpful is an object-oriented language for an object-oriented database model? **Proceedings of 1986 International Workshop on Object-Oriented Database Systems**. pp.124-132. Washington: IEEE Computer Society Press.

[10] Fine, K. (1985) **Reasoning with Arbitrary Objects**. Oxford: Basil Blackwell.

[11] Frixione, M., Gaglio, S. & Spinelli, G. (1989) Are there individual concepts? Proper names and individual concepts in SI-Nets. **International Journal of Man-Machine Studies, 30** 489-503.

[12] Gaines, B.R. (1988) Knowledge acquisition systems for rapid prototyping of expert systems. **INFOR, 26**(4), 256-285 (November).

[13] Gaines, B.R. (1989) An ounce of knowledge is worth a ton of data: quantitative studies of the trade-off between expertise and data based on statistically well-founded empirical induction. **Proceedings of the Sixth International Workshop on Machine Learning.** pp.156-159. San Mateo, California: Morgan Kaufmann (June).

[14] Gaines, B.R. (1989) Design requirements for knowledge support systems. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Fourth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop**. pp.12-1-12-20. Banff (October).

[15] Gaines, B.R. & Shaw, M.L.G. (1989) Comparing the conceptual systems of experts. **Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.** pp.633-638. Los Angeles, California: Morgan Kaufmann (August).

[16] Gaines, B.R. & Shaw, M.L.G. (1990) Foundations of knowledge acquisition. **Proceedings of the First Japanese Workshop on Knowledge Acquisition for Knowledge-Based Systems (JKAW'90)**. Kyoto, Japan (October).

[17] Gaines, B.R. & Vickers, J.N. (1988) Design considerations for hypermedia systems. **Microcomputers for Information Management**, 5(1), 1-27 (March).

[18] Ginsberg, M.L. (1987) **Readings in Non-Monotonic Reasoning.** San Mateo, California: Morgan Kaufmann.

[19] Goguen, J.A. (1973) Realization is universal. **Mathematical Systems Theory 6** 359-374.

[20] Grossman, R. (1983) **The Categorial Structure of the World.** Bloomington: Indiana University Press.

[21] Jackson, P., Reichgelt, H. & van Harmelen, F. (1989) **Logic-Based Knowledge Representation.** Cambridge, Massachusetts: MIT Press.

[22] Kelly, G.A. (1955) **The Psychology of Personal Constructs.** Norton: New York.

[23] Kroy, M. (1976) **Mentalism and Modal Logic.** Wiesbaden: Athenaion.

[24] Lambert, K. (1983) **Meinong and the Principle of Independence.** Cambridge: University Press.

[25] Leblanc, H. (1976) **Truth-Value Semantics.** Amsterdam: North-Holland.

[26] Lenat, D.B. & Guha, R.V. (1990) **Building Large Knowledge-Based Systems.** Reading, Massachusetts: Addison-Wesley.

[27] Levesque, H. J. (1984) A logic of implicit and explicit belief. **Proceedings of AAAI-84.** pp.198-202. San Mateo, California: Morgan Kaufmann.

[28] Maida, A.S. & Shapiro, S.C. (1982) Intensional concepts in propositional semantic networks. **Cognitive Science, 6**(4) 291-330.

[29] Manago, M. (1989) Knowledge intensive induction. **Proceedings of the Sixth International Workshop on Machine Learning.** pp.151-155. San Mateo, California: Morgan Kaufmann (June).

[30] MacGregor, R.M. (1988) A deductive pattern matcher. **Proceedings of AAAI88.** pp.403-408. San Mateo: Morgan Kaufmann.

[31] MacGregor, R.M. (1989) The evolving technology of the KL-ONE family of knowledge representation systems. **Workshop on Formal Aspects of Semantic Networks.** Santa Catalina Island, California (February).

[32] Mendelsohn, R.L. (1989) Objects and existence: reflections on free logic. **Notre Dame Journal of Formal Logic, 30**(4) 604-623.

[33] Meinong, A. (1983) **On Assumptions.** Berkely: University of California Press.

[34] Monarch, I. & Nirenburg, S. (1987) The role of ontology in concept acquisition for knowledge-based systems. **Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems (EKAW'87).** Reading University, UK (September).

[35] Morik, K. Ed. (1985) **Knowledge Representation and Organization in Machine Learning.** Berlin: Springer-Verlag.

[36] Murray, K. & Porter, B.W. (1988) Developing a tool for knowledge integration: initial results. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Third AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** pp.22-1-22-12. Banff (November).

[37] Nebel, B. (1988) Computational complexity of terminological reasoning in BACK. **Artificial Intelligence, 34** 371-383.

[38] Nebel, B. (1990) **Reasoning and Revision in Hybrid Representation Systems.** Berlin: Springer-Verlag.

[39] Parsons, T. (1980) **Non-Existent Objects.** New Haven: Yale University Press.

[40] Patel-Schneider, P.F. (1989) A four-valued semantics for terminological logics. **Artificial Intelligence, 38** 319-351.

[41] Reiter, R. (1978) On reasoning by default. **Proceedings of TINLAP-2, Theoretical Issues in Natural Language Processing.** pp.210-218. Champaign, Illinois: University of Illinois.

[42] Schmidt-Schauss, M. (1989) Subsumption in KL-ONE is undecidable **Proceedings of KR'89: First International Conference on Principles of Knowledge Representation and Reasoning.** pp.421-431. San Mateo, California: Morgan Kaufmann (May).

[43] Schmolze, J. (1989) Terminological knowledge representation systems supporting n-ary terms. **Proceedings of KR'89: First International Conference on Principles of Knowledge Representation and Reasoning.** pp.432-443. San Mateo, California: Morgan Kaufmann (May).

[44] Skuce, D., Shenkang, W. & Beauville, Y. (1989) A generic Knowledge Acquisition Environment for conceptual and ontological analysis. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Fourth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop**. pp.31-1-31-20. Banff (October).

[45] Watanabe, H. (1989) Heuristic graph displayer for G-BASE. **International Journal of Man-Machine Studies, 30**(3) 287-302 (March).

[46] Woods, J. (1974) **The Logic of Fiction**. The Hague: Mouton.

[47] Woods, W.A. (1975) What's in a link: Foundations for semantic networks. Bobrow, D.G. & Collins, A.M. (Eds) **Representation and Understanding: Studies in Cognitive Science**. pp.35-82. New York: Academic Press.

[48] Zalta, E.N. (1988) **Intensional Logic and the Metaphysics of Intentionality**. Cambridge, Massachusetts: MIT Press.

Acquiring text editing programs from examples

Dan Hua Mo*
Department of Computer Science
University of Calgary
2500 University Drive, N.W.
Calgary, Alberta, Canada T2N 1N4

Feb. 20, 1990

Abstract

It is frustrating to perform highly repetitive editing tasks manually and their execution should be automated for the benefit of computer users. This paper describes how ideas for *programming by example* can be applied to the *repetitive editing task* problem. Rather than having users program a task, a programming-by-example system requires them to specify examples of how to perform it and synthesizes from these a program that performs the task. A text editing learning system, TELS, is designed and implemented. It is shown to be able to learn procedures that include variables and control structures from traces. Interactive debugging by protocol is proposed to combat the unreliability of a procedure taught by example.

*The author is currently with Research and Development Dept., Willowglen Systems Ltd., Manning Close N.E., Calgary, Alberta, Canada T2E 7N5.

1 Introduction

Let us introduce the problem through an example. Consider the plight of a user who over a period of years has painstakingly accumulated a file of bibliographic references to thousands of books and papers. Each entry contains information such as the names of authors, title, publication date, page numbers, journal name, etc; and has the form shown in Figure 1.

Suppose that the user decides to transform the data from its original format into that of Figure 2. This conversion process poses a formidable problem, for transforming the file involves a considerable amount of tedious work. This work is productive, in the sense that every step makes perceptible progress towards the solution of the problem, but is repetitive nonetheless.

Now, the problem and the approach are defined formally. A *repetitive editing task* is the one that possesses the following

```
Line 1:  %A Ackley, D.H.
Line 2:  %A Hinton, G.E.
Line 3:  %A Sejnowski, T.J.
Line 4:  %D 1985
Line 5:  %T A learning algorithm for Boltzmann machines
Line 6:  %I Research Report, Department of Electrical and Computer
Line 7:  Engineering, Ben-Gurion University
Line 8:  %C Beer-Sheva 84105, Israel
```

Figure 1: The original format of the entry

properties:

1. The input text of the task can be decomposed into consecutive blocks among which a number of fixed strings are frequently present.

2. When a user performs the task manually, he has to constantly repeat the editing actions for the fixed strings, along with some novel actions for the various parts of different blocks.

Such tasks cause considerable frustration, and it is worthwhile to consider whether they can be automated.

Programming by example (Halbert, 1984, Andreae, 1985, Biermann *et al*, 1984, Waterman *et al*, 1986, Bauer, 1979, Nix, 1983) is a way of programming a software system whereby a user performs manually one or more executions of a task and the system unites and generalizes these executions into a program which can then be executed on demand. Generalization may also need to be performed when the program is executed, so that it can match novel situations that are presented. One cardinal advantage of programming by example is that users can record procedures by direct manipulation within the system's interface, without having to specify procedural constructs explicitly.

Previous work has exemplified different approaches to programming by example. The TEMPO (Affinity, 1986), *SmallStar* (Halbert, 1984) and *Clerk* (MacDonald & Witten, 1987) systems concentrate on employing interface facilities, such as dialogue boxes, property sheets and clerk icons, for transcribing programs. These facilities can assist users to specify examples unambiguously, but the systems impose many requirements on the users's knowledge and programming skills. Other systems employ various induction methods to generalize examples into a program. These techniques are either data-driven, such as the NODDY (Andreae, 1985), EBE (Nix, 1983) and MetaMouse (Maulsby & Witten, 1988) systems, or model-directed, *e.g.* EP3 (Waterman *et al*, 1986). The

```
Ackley, D.H., Hinton, G.E., and Sejnowski, T.J. (1985) ''A
learning algorithm for Boltzmann machines'', Research
Report, Department of Electrical and Computer Engineering,
Ben-Gurion University, Beer-Sheva 84105, Israel
```

Figure 2: The resulting format

data-driven inductive systems attempt to induce syntactic program constructs such as variables, data structures and control structures, whereas the model-directed one induces the high level algorithms underlying programs. A teacher plays a significant role in guiding the learning of interactive learning systems such as Meta-Mouse and Clerk, but not in the case of the non-interactive NODDY and EBE. The TELS system described in this paper is an inductive system which is able to learn from traces text editing procedures that include variables and control structures. Its main feature is that it provides facilities to permit interactive debugging of programs taught by example. A *trace* is defined in this paper as a sequence of editing commands, along with values of selected attributes which help to characterize the context in which the commands are executed.

The next section describes the design of the TELS system. Section 3 presents the system's learning algorithm. Section 4 gives the system's implementation status and results. The final section is an acknowledgement.

2 Design

2.1 System overview

The top-level architecture of the TELS system is shown in Figure 3. It consists of a *recorder*, *generalizer*, *predictor* and *debugger*. The system is embedded in a modified version of the MiniEdit editor (Chernicoff, 1984) on a Macintosh computer. MiniEdit implements a set of primitive operations. It is not shown in the diagram.

The recorder records a complete editing trace of a task from a user. The trace comprises values of neighboring attributes of the cursor position where an editing action occurs. The recorder ignores extraneous mouse-clicks or mouse-selects that are not followed by any editing commands or typing actions. The generalizer includes various modules that generalize a trace into a procedure. The predictor utilizes the procedure to predict the user's editing actions. The user can accept, reject or skip the predictions, or supply new examples. Such user-supplied information is employed by the debugger to perform runtime editing of a procedure.

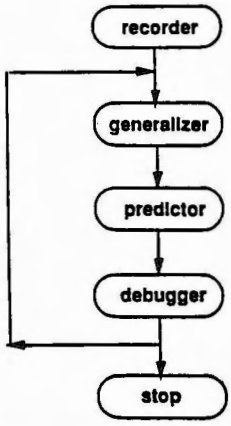


Figure 3: Overview of the TELS system

2.2 Representation of programs

The system represents programs by flow diagrams resembling NODDY's procedure representation (Andreae, 1985). Figure 5 illustrates an initial program for performing the task in Figure 4. It consists of circles connected by arrows representing the flow of control. The circles are known as *nodes*. Except for the *start* and *stop* nodes, each node is an *action step* tuple (*action*, *condition*). An *action* is an editing action. Then comes the condition, or the postcondition of the action, which is a description that governs when control enters the next node. The condition is a conjunction of descriptors for textual and positional attributes, representing the state of the world after the action is performed. Each node except the stop nodes has one or more arrows, or *sequence links*, emanating from it. Nodes with more than one sequence link correspond to branch points. Figure 5 shows actions and their conditions in the initial program of the task.

2.3 Generalization

This section develops methods for generalizing positional and textual descriptors. In the next section, these will be brought together to form a complete learning algorithm. Generalization is performed by the system to make a program applicable to various situations. The system performs several kinds: (1) generalization of positional and textual descriptors, (2) generalization of the condition of a program step which is a conjunction of those descriptors, and (3) generalization of sequential patterns of procedures to include branches and loops.

Task:

Transform the following input text into the mailing label format.

Input:

```
John Blow, 2416 22 St., N.W., Calgary, T2M 3Y7.  #284-4983
Tom Brown, Suite 1, 2741 Banff Blvd., S.E., Calgary, T2L 1J4.  #284-4567
```

Output:

```
John Blow,
2416 22 St.  N.W.,
Calgary,
T2M 3Y7.

Tom Brown,
Suite 1,
2741 Banff Blvd., S.E.,
Calgary,
T2M 1N4.
```

Figure 4: Task transforming addresses to mailing labels

2.3.1 Generalizing descriptors for positional attributes

A cursor position on the screen can be represented by using positions relative to a word, line, paragraph and file. For example, the beginning of a file can be represented as "Posn(beginning-of-word, beginning-of-line, beginning-of-paragraph, beginning-of-file)". Of course, the representation scheme cannot distinguish all cursor positions, but it does differentiate some classes of position. To make up for the shortcomings of positional attributes, the system uses other textual attributes.

The system generalizes a descriptor that has a generalization hierarchy by applying the *climbing generalization tree* rule (Michalski, 1983): for a set of descriptor values, D1, D2, ..., Dn, their least common generalization is taken as their generalized description. For instance, two values "`beg-of-line`" and "`end-of-line`" for the descriptor "`LinePosn`" can be generalized into "`beg-or-end-of-line`" by climbing the generalization tree in Figure 6.



S0: start

S1: click;
Prec_Word('Blow,␣'),Succ_Word('2416');
Posn(beg-of-word, mid-of-line, mid-of-paragraph, mid-of-file);
Num_of_Words(+2), Num_of_Lines(0), Num_of_Chars(+11);

S2: type(carriage-return);
Prec_Word('Blow,␣'),Succ_Word('2416');
Posn(beg-of-word, beg-of-line, mid-of-paragraph, mid-of-file);
Num_of_Words(0), Num_of_Lines(+1), Num_of_Chars(0);

S3: click;
Prec_Word('N.W.,␣'),Succ_Word('Calgary');
Posn(beg-of-word, mid-of-line, mid-of-paragraph, mid-of-file);
Num_of_Words(+4), Num_of_Lines(0), Num_of_Chars(+19);

S4: type(carriage-return);
Prec_Word('N.W.,␣'),Succ_Word('Calgary');
Posn(beg-of-word, beg-of-line, mid-of-paragraph, mid-of-file);
Num_of_Words(0), Num_of_Lines(+1), Num_of_Chars(0);

S5: click;
Prec_Word('Calgary,␣'),Succ_Word('T2M');
Posn(beg-of-word, mid-of-line, mid-of-paragraph, mid-of-file);
Num_of_Words(+1), Num_of_Lines(0), Num_of_Chars(+9);

S6: type(carriage-return);
Prec_Word('Calgary␣'),Succ_Word('T2M');
Posn(beg-of-word, beg-of-line, mid-of-paragraph, mid-of-file);
Num_of_Words(0), Num_of_Lines(+1), Num_of_Chars(0);

S7: Select('#284-4983');
Selected_Text('#284-4983');
Prec_Word('3Y7.'),Succ_Word(null);
Posn(beg-of-word, mid-of-line, mid-of-paragraph, mid-of-file);
Num_of_Words(+2), Num_of_Lines(0), Num_of_Chars(+9);

S8: Delete(#284-4983);
Prec_Word('3Y7. '),Succ_Word(null);
Posn(beg-of-word, end-of-line, mid-of-paragraph, mid-of-file);
Num_of_Words(0), Num_of_Lines(0), Num_of_Chars(0);

S9: stop

Figure 5: Trace for task in Figure 4

Values for numeric positional descriptors are generalized by finding the smallest number interval that covers the values in example descriptors. For instance, "Num_of_Words (1)" and "Num_of_Words (3)" are generalized into "Num_of_Words (<= 3)".

2.3.2 Generalizing descriptors for textual attributes

First, let us define some terms. *Character classes* are regular expressions that use a restricted form of alternation to define a pattern matching a single character drawn from a particular set (Aho *et al*, 1986). A character class is frequently denoted as an expression, [S], that matches any character in S, where S is a string. A notational shorthand "-" is also used to specify character subranges. For example, "[0-9]" means the same as "[0123456789]". In this paper, a character class is denoted by its name as well. For example, "[lower case]" is the same as "[a-z]". Figure 7 gives a generalization tree for the character classes used. Given a string, its character class is defined as the least common generalization of all classes for the constituent characters. For example, the character class for "aaa222" is "[a-zA-Z0-9]" which is the least common generalization of a's class "[a-z]" and 2's class "[0-9]".

A *gap pattern* is a sequence of alternating strings and gaps (Nix, 1983). A *gap* is a variable symbol which matches a string (possibly empty). In this paper, a gap pattern is annotated by a character class and called an *annotated gap pattern*. The notation for such a gap is "[*class*]" where *class* is a character class shown in the generalization tree of Figure 7. For instance, "a␣[a-z]␣gap␣pattern" will match strings that begin with "a␣" followed by some lower-case letters and end in "␣gap␣pattern", where "[a-z]" denotes the gap in the pattern and "␣" the space character. The substrings excluding the gaps in a gap pattern is called its *constant subsequence*. A *common subsequence* of two strings is a string that is concatenated from their common substrings in a way that preserves their sequential order. For example, "AABB", "AA" and "AB" are some common subsequences of "AAXXXXBB" and "AAYYYYBB", whereas "BBAA" is not since the sequential order between "AA" and "BB" is altered.

In practice, many tasks require the system to identify a common pattern for selected strings, preceding words or succeeding words. For instance, a user may want to place the cursor after some words such as "foo(x)" and "bar(y)". A pattern matching these strings must be induced so that the user's intention may be captured. The problem is that there are many patterns that may be generalized from the strings. A learning system must use certain preference criteria to limit the number of generalizations. A string pattern generalization algorithm described in (Mo, 1989) utilizes the common subsequences of strings from the maximum number of character classes, called the *mixed common subsequence*, and the character class hierarchy. It is biased to-

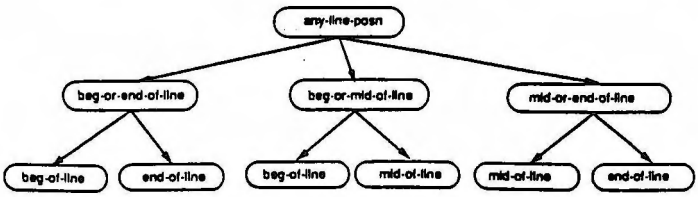
wards discovering maximally specific gap patterns that characterize common subsequences of the character classes in given example strings. We describe the algorithm briefly here. The detailed algorithm can be found in (Mo, 1989). First, the algorithm computes the constant subsequence of the required gap pattern by selecting a mixed common subsequence containing characters from the maximum number of character classes. Then, it inserts gaps into the constant subsequence to make it match the input strings.

An example of the algorithm is shown as follows. Let us illustrate how "`Blow,␣`" and "`N.W.,␣`" are generalized. The above algorithm first finds the mixed common subsequence of their values which is "`,␣`". Then, it constructs the gap pattern "`[letter],␣`" from the mixed common subsequence that matches "`Blow,␣`". The annotation of the gap has a character class "`letter`" since the substring "`Blow`" matching the gap belongs to that character class. The algorithm continues by matching the gap pattern with "`N.W.,␣`". This time, the gap matches the substring "`N.W.`", but the gap's character class does not cover the character class "`character`" of the substring. Therefore, the algorithm generalizes the character classes of the gap and substring into their least common generalization "`character`" by climbing the character class generalization hierarchy. If the second string were "`N.W.,␣XXXX`", then the algorithm would output a new gap pattern "`[character],␣[upper case]`".

2.3.3 Generalizing conditions

A condition of a program step is a conjunctively-linked expression. Each descriptor is also referred to as a *constraint.* To execute an editing action correctly, the system needs to search for the position in a user's text that satisfies the constraints of the action (discussed in the next section). However, these constraints may contain redundant descriptors which do not relate to the user's intention. Therefore, to capture the user's intention, important constraints in a condition must be isolated. The system uses the *dropping condition* rule to generalize a condition by removing some conjunctively-linked descriptors (Michalski, 1983). The problem is to determine which descriptors should be dropped. Some preference criteria are established to bias the system to drop undesired descriptors. It accomplishes this by preserving *significant* constraints determined by the heuristic rules in Table 1.

The first rule retains those matched descriptors that have identical values, because these reflect common constraints among the matching nodes. The other rules preserve the significant constraints of a node by examining the type of editing actions in the node and its successors to select important constraints. Since key editing actions (i.e. cut, paste and type) always start with either mouse-clicks or mouse-selects, an editing procedure contains pairs of nodes starting with either of these two actions. The second and third rules employ this fact. If a mouse-click action is followed by some key editing ac-



Generalization hierarchy for line position attributes

Figure 6: Generalization hierarchies

1. Keep those matched descriptors that have identical values.
2. **IF** a pair of actions starts with a mouse click, such as click-then-type and click-then-paste **THEN** retain descriptors for neighboring words, i.e. `Prec_Word(W1)` and `Succ_Word(W2)`.
3. **IF** a pair of actions starts with a mouse select, such as select-then-type, select-then-cut, select-then-paste **THEN** retain the descriptor for the selected text fragment, i.e. `Selected_Text(T)`.

Table 1: Rules for determining significant constraints

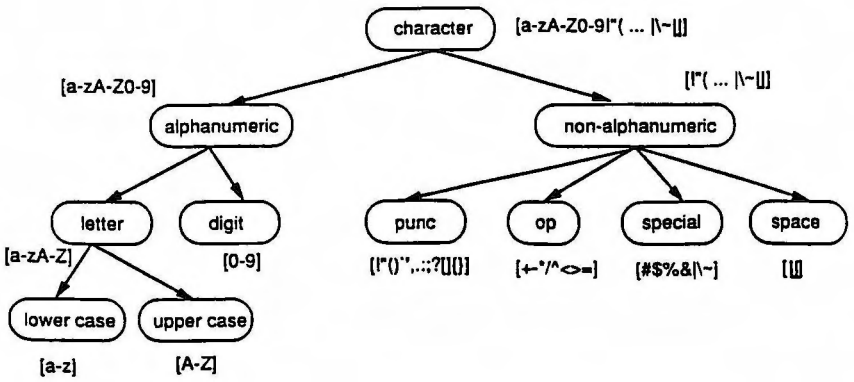


Figure 7: Generalization hierarchy for textual attributes

tions, the neighboring words may be the user's search target for editing and hence these words are significant. In case of a mouse-select, the selected text may be the user's search target and therefore significant.

Constraints that are not retained by the above steps are dropped by marking them as ignored. They are remembered since they may be needed for later specialization in case a condition is overly generalized (Maulsby & Witten, 1988).

3 Learning algorithm

Figure 10 depicts the learning algorithm in its most general form. It consists of two parts: the non-interactive and interactive learners. The former constructs from a single trace a procedure that includes variables, loops and generalized conditions. The procedure is then employed by the latter to predict a user's editing actions in new situations. It is further refined by taking into account the user's responses to predictions and novel inputs. Incorrect generalizations made by the non-interactive learner are detected and modified. Branches and new nodes are added to reflect the novel inputs.

Generalizations for positional and textual descriptors (see Sections 2.3.1 and 2.3.2) are performed by the loop inducer and debugger. The conditional generalizer generalizes conditions of nodes according to the method described in Section 2.3.3. Due to the complexity of the non-interactive learner, its algorithm is

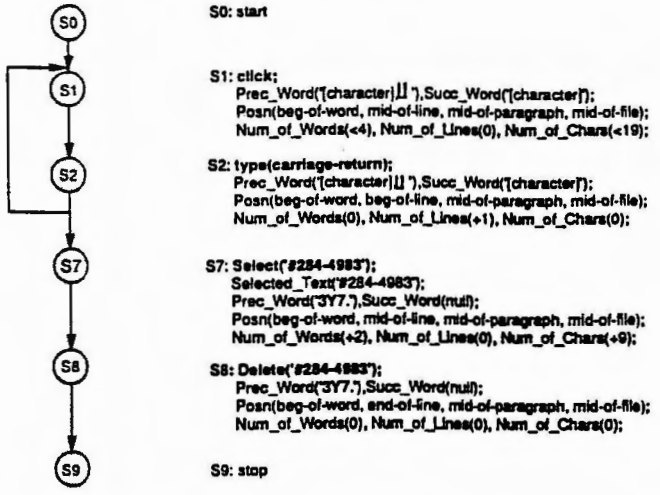


Figure 8: Intermediate program generalized from trace in Figure 5

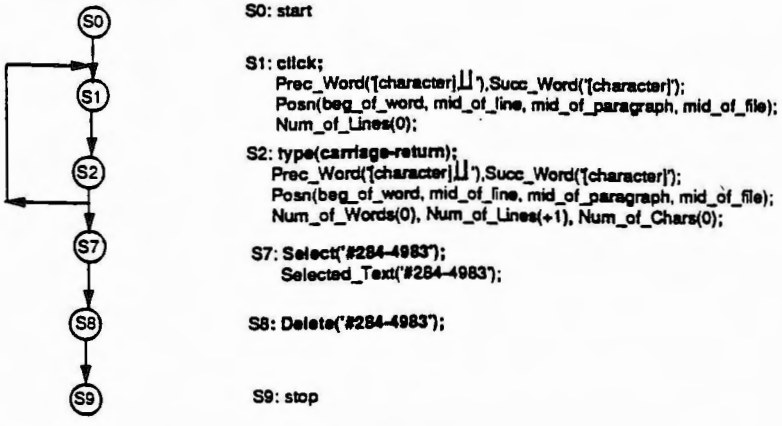


Figure 9: Final program generalized from trace in Figure 5

not described here, but can be found in (Mo, 1989).

3.1 The interactive learner

The purpose of the interactive learner is to make the procedure learned by the non-interactive one reliable. Overgeneralization resulting from the previous learning stages is discovered at this stage through interaction between the system and a user. In addition, the over-specific parts of the procedure are generalized to cover new situations in the same task. Novel inputs from the user are included in the procedure as well. The user's cooperation plays a significant role in the system's learning at this stage. The interactive learner consists of a *predictor*, *executor*, and *debugger*. The first two are discussed in this subsection, while the debugger merits a subsection of its own because of its complexity.

The **predictor** follows the flow of control in the learned procedure and predicts editing actions in program steps. Beginning at the *start* node, it searches a position in the user's text that satisfies the *significant* textual and positional constraints of the condition of a program step associated with a *click* or *select* action. The significant constraints are those preserved after generalization of constraints. For instance, before step 1 of the procedure in Figure 9 is executed, the system will do the following.

1. Search forward on the same line in the text, because the values of "Num_of_Lines (0)" is zero and "Num_of_Words (<4)" marked dropped is greater than zero. If they were negative, then it would search backward.

2. Search for the position where the preceding and succeeding words match the corresponding gap patterns in the node. If a match is found, then check other positional constraints that have constant value, such as "`beg-of-word`". If they are also satisfied, execute the click action by placing the cursor at the position and predicting the editing action "`type(carriage-return)`" on the prediction window in Figure 11. If the user accepts the prediction, the *executor* is invoked to carry out the actual typing. Otherwise, it asks the user to intervene and supply debugging information.

The present system uses only click or select nodes to initiate a search for an editing position. The condition in the node of a *key editing action* should be used to check whether the action is executable before a prediction. However, the system does not at present do this. A user can intervene to correct a wrong prediction and thereby step through a procedure.

At a fork branch, such as S7 in the procedure of Figure 9, a step is selected for execution if its condition can be satisfied. Therefore the loop in the figure will be terminated if S1's condition no longer holds.

The **executor** checks the action type of a procedure step and invokes the module managing the editing action to per-

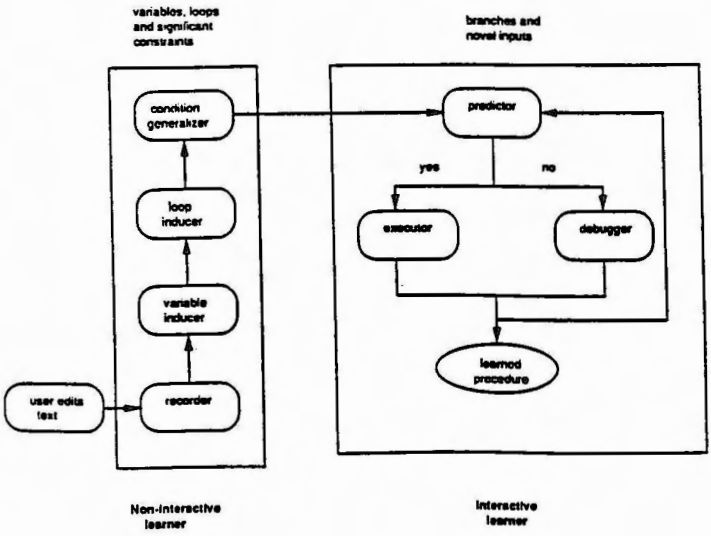


Figure 10: Learning algorithm

form it at the current cursor position. For instance, in Figure 9, when S2 is executed, "`type(carriage-return)`" will be carried out by calling the module for processing character typing. A carriage return will be placed at the current cursor position on the screen.

3.2 The debugger

The debugger allows the user to edit a learned program at run-time through teaching. A procedure learned from examples is not robust in the sense that it may not work correctly in various examples of the same task. There are two reasons. First, it may contain bugs that result from over-generalization or over-specialization in the learning process. Second, the system may encounter novel situations.

The chief problem in debugging a procedure is to locate the place where a bug resides. The MARVIN learning system employs a backtracking method to detect bugs (Sammut & Banerji, 1986). It traces back through the generalization steps that it took to reach an incorrect generaliza-

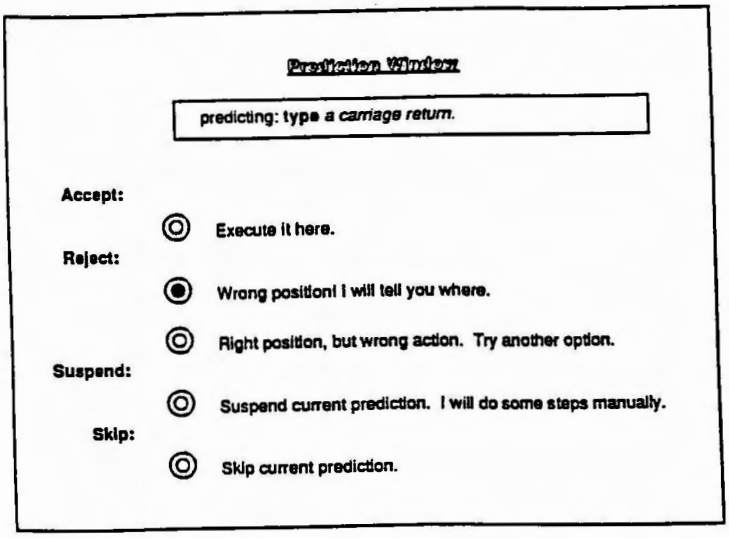


Figure 11: Prediction window

tion. At each step, it first constructs an hypothesis for the step and then asks the user to confirm if the hypothesis is valid. Suppose that an incorrect generalization rule "has-wings(X)→ fly(X)" was built in and the system reaches an incorrect description "chicken(x) → fly(x)" by using the above rule together with "chicken(X) → has-wings(X)". By backtracking, MARVIN would generate a question for "has-wings(x) → fly(x)": for example, "does chicken x fly?" A bug is identified if the user answers no. MARVIN is powerful at constructing testing hypotheses, but the backtracking method requires a user to answer many questions before a bug can be identified.

The system described in this paper uses a different debugging method. It does not require any backtracking. Rather, it debugs a step by asking the user to give correct examples in the interface and then comparing the correct inputs with information in the step to make internal adjustments. The system utilizes the user's rejections of its predictions. Since a prediction may be rejected for many reasons, the learning system should be sensitive to different kinds of response to the prediction and permit the user to debug or extend the learned procedure accordingly. Choices are provided in the prediction window of Figure 11.

For instance, some over-generalization bugs can be fixed using the debugger. These occur when a node is executed in a *place* (i.e. a cursor position or selected text fragment) where it should not have

been. For instance, the procedure in Figure 9 is actually overgeneralized. When it is executed over the example itself, a new line will be created after the comma in "2416 22 St., N.W.". The user can correct the bug by clicking the first radio button under "Reject" (Case 1) and placing the cursor at the desired position where the predicted action should be performed. The system then treats the misplaced action as a negative example and stores it in the node. The neighboring constraints for the correct position are recorded and merged with those in the node. Having done this, the *executor* is invoked to perform the predicted action in the correct position.

4 Implementation status and results

The TELS system is written in Turbo Pascal 1.1 (Borland, 1986) on a Macintosh computer. It has about 7380 lines of PASCAL code (367 K) and its compiled version takes up 81 K. It has been evaluated analytically and empirically with three example tasks (Mo, 1989). We present only the results from the evaluation here while the details of the evaluation are described in (Mo, 1989).

It was demonstrated that TELS is capable of learning from traces text editing procedures that include variables and control structure. The disadvantage that user traces may vary even for the same task are lessened by embedding TELS in a simple editor that provides only a set of primitive operations, and permitting TELS to predict actions eagerly to a user to control free variation in accomplishing the task. Interactive debugging of procedures learned by example was shown to be effective. By comparing the number of mouse-clicks and keystrokes with and without TELS, it was shown that the system reduces effort in repetitive editing tasks. A comparison with Nix's EBE (Nix, 1983) showed that TELS was better able to handle variation in input text. However, for tasks with fixed-format input text, EBE was more effective than TELS.

TELS has several shortcomings. First, the condition generalizer retains significant constraints using some *ad hoc* preference criteria. It should be investigated how to resolve conflict between constraints represented by different attributes and come up with a generalized condition. Second, positional attributes in TELS are based upon a classification of text into characters, words, lines and paragraphs. The definition of these units should be elaborated according to the *mode* of text being edited such as a TROFF file or C program.

5 Acknowledgement

I would like to thank Ian Witten and Bruce MacDonald for helping me develop these ideas. This research has been supported by a research assistantship from the Dept. of Computer Science at the University of Calgary and by the Natural Sciences and Engineering Research Council

of Canada.

References

Affinity, (1986) *Tempo.* Affinity Microsystem Ltd., Boulder, CO.

Aho, A., Sethi, R., and Ullman, J., (1986) *Compilers: Principles, Techniques and Tools.* Addison-Wesley.

Andreae, P., (1985) *Justified generalization: acquiring procedures from examples.* PhD thesis, Dept. of Electrical Engineering and Computer Science, MIT, January.

Bauer, M. A., (1979) Programming by example. *Artificial Intelligence*, (12):1–21, May.

Biermann, A., Guiho, G., and Kodratoff, Y., (1984) An overview of automatic program construction techniques. In Biermann, A., Guiho, G., and Kodratoff, Y., editors, *Automatic Program Construction Techniques*, pages 3–30. Macmillan.

Borland, (1986) *Turbo Pascal for the Mac.* Borland International, Inc., 4585 Scotts Valley Drive, Scotts Valley, CA 95066, USA.

Chernicoff, S., (1984). Macintosh revealed: Volume 2, programming with the toolbox.

Halbert, D. C., (1984) Programming by example. Research Report OSD-T8402, Xerox PARC. Palo Alto CA, December.

MacDonald, B. and Witten, I., (1987) Programming computer controlled systems by non-experts. Research Report 87/271/19, Dept. of Computer Science, University of Calgary, July.

Maulsby, D. and Witten, I., (1988) Teaching a mouse how to draw. Research Report 88/294/06, Dept. of Computer Science, University of Calgary, January.

Michalski, R., (1983) A theory and methodology of inductive learning. In Michalski, R., Carbonell, J., and Mitchell, T., editors, *Machine Learning*, volume 1, pages 83–134. Morgan Kaufmann.

Mo, D., (1989) Learning text editing procedures from examples. Master's thesis, Dept. of Computer Science, University of Calgary.

Nix, R., (1983) *Editing from examples.* PhD thesis, Yale University.

Sammut, C. and Banerji, R. B., (1986) Learning concepts by asking questions. In Michalski, R., Carbonell, J., and Mitchell, T., editors, *Machine Learning*, volume 2, pages 167–191. Morgan Kaufmann.

Waterman, D., Faught, W., Klahr, P., Rosenschein, S., and Wesson, R., (1986) Exemplary programming: applications and design considerations. In Klahr, P. and Waterman, D., editors, *Expert systems: techniques,*

tools and applications, pages 273–309.
Addison-Wesley.