Knowledge Acquisition from Data Bases

Xindong Wu

Ph.D.

University of Edinburgh

1993

Declaration

I hereby declare that this thesis has been composed by myself and that the work described in it is my own.

____

Xindong Wu

Acknowledgements

The work described in this thesis was supported in part by the National Natural Science Foundation of China under Grant No. 68975025 when I was the grant holder and project head of the 68975025 project and is supported in part by the ORS Award of the United Kingdom and the University of Edinburgh Research Scholarship when I am in receipt of them to study for my Ph.D.

Thanks are owed to the following:

- First, I express deep appreciation for help and supervision provided by Dave Robertson. Time and again, technically and socially, he always comes through when it is required, in a way that no Ph.D. student should expect. He has proofread this thesis and most of my papers produced in Edinburgh. Without his support, it would not be possible for me to produce my first thesis draft within a year of arrival and to submit at the University's minimum time point of 22 months of resident Ph.D. study.

- I extend sincere thanks to Robert Rae for his supervision. Although he is really busy in both administrative and technical work (as he is deputy director of the Artificial Intelligence Applications Institute), he has done his best to help me in proofreading this thesis and several of my papers. Also, his advice on how to do research has been helpful to me.

- I am most grateful to Jim Howe who has been a source of powerful support and constant encouragement from the very beginning of my application for Ph.D. study at Edinburgh, and to Alex Gammerman (Royal Holloway, University of London), Chris Mellish, Peter Ross, Derek Sleeman (Aberdeen

University), Tim Niblett (Turing Institute), and Robin Boswell (Turing Institute) for additional comments and advice on my work and/or the earlier thesis drafts.

- Also, I am indebted to many Chinese teachers and colleagues for their advice and support on my work relevant to this thesis when I was in China and my fellow students in Edinburgh for their help in English, LaTeX, and SICStus Prolog. Among them are especially Diancheng Zhang, Shenkang Deng, Fanlun Xiong, Jingao Li, Yong Chen; Ian Frank, Alan Black, Wamberto Vasconcelos, Sheila Glasbey, Alistair Knott, Peter Nowell, Kathleen King, Flavio Soares Correa Da Silva, Khee-Yin How and Ian Gent.

- All the equipment and computing facilities for this thesis are provided by the Department of Artificial Intelligence. I thank collectively all those people who have helped me in different ways: Jean Parker, Carole Douglas, Craig Strachan, Ann Ralls, Michael Keightley, Jean Bunten, Joanne Clarke, Emma Lawson, Janet Lee, Margaret Pithie, Karen Konstroffer, Eric Forbes, Olga Franks, Millie Tupman, Brain Cant and Jim Donachie.

- Last and not least, especial thanks are due to my wife, Zou Yan, for her kind love, support and encouragement throughout. I would like to dedicate this thesis to her and to my cute daughter, Lanlan, who can already write her name in both English and Chinese and do simple sums on her fingers and by heart although she is only four.

Abstract

Knowledge acquisition from data bases is a research frontier for both data base technology and machine learning (ML) techniques, and has seen sustained research over recent years. It also acts as a link between the two fields, thus offering a dual benefit. Firstly, since data base technology has already found wide application in many fields, ML research obviously stands to gain from this greater exposure and established technological foundation. Secondly, ML techniques can augment the ability of existing data base systems to represent, acquire, and process a collection of expertise such as those which form part of the semantics of many advanced applications (*e.g.* CAD/CAM). The major contribution of this thesis is the introduction of an efficient induction algorithm to facilitate the acquisition of such knowledge from data bases.

There are three typical families of inductive algorithms: the generalisation-specialisation based AQ11-like family, the decision tree based ID3-like family, and the extension matrix based family. A heuristic induction algorithm, HCV, based on the newly-developed extension matrix approach is described in this thesis. By dividing the positive examples (PE) of a specific class in a given example set into intersecting groups and adopting a set of strategies to find a heuristic conjunctive rule in each group which covers all the group's positive examples and none of the negative examples (NE), HCV can find rules in the form of variable-valued logic for PE against NE in low-order polynomial time. The rules generated in HCV are shown empirically to be more compact than the rules produced by AQ11-like algorithms and the decision trees produced by the ID3-like algorithms.

KEshell2, an *intelligent learning data base system*, which makes use of the HCV algorithm and couples ML techniques with data base and knowledge base technology, is also described.

Table of Contents

List of Figures

List of Tables

Glossary of Notation

$(), [], \{\}$	the conventional parentheses/brackets/braces
$\#N$	the ordinal N
$<, >, \leq, =, \geq, \neq, \approx$	the conventional arithmetic relations
$\sum$	the arithmetic sum
$\leftarrow$	the conventional (arithmetic or symbolic) assignment
$\log_2$, sin, cos, max, min	the conventional arithmetic functions
ϕ	the empty set
$\cup, \cap, \in$	the conventional set relations
$\mid S \mid$	base: the number of elements in set S
$\{, <Term>\}^*$	zero, one or more $<Term>$s here
$\{<Term>\}_0^1$	with or without $<Term>$ here
$\vee, \mid$	the conventional logical inclusive *or*
$\&, \wedge$	the conventional logical *and*
$\exists$	the conventional existential quantifier
$:-$	the logical if
$:=$	"is defined by"
$\rightarrow$	the conventional logical implies
_	the anonymous variable

$*$	the dead element in extension matrixes
$\#$	a *Don't Care* value
M^T	the transpose of matrix M
$\times$	the cross product
O	the order of time complexity
$\oplus$	the fuzzy plus
$\otimes$	the fuzzy times

Chapter 1

Introduction

1.1 Problem and Domain

Artificial intelligence (AI) is a subject concerned with the problem of how to make machines perform tasks, like vision, planning and diagnosis, that usually need human intelligence. Machine learning (ML) research in AI is concerned with the problem of how machines can acquire the knowledge that might enable them to perform those tasks. Along with the recognition of the so called knowledge bottleneck problem [Feigenbaum 81] in transforming knowledge from human experts to knowledge-based systems, ML research has been expanding rapidly in recent years.

Research on ML has concentrated in the main on inducing rules from unordered sets of examples, especially attribute-based induction, an inductive formalism where examples are described in terms of a fixed collection of attributes. Learning from examples has been seen [Michie 87, Quinlan 88a] as a feasible way of avoiding the knowledge bottleneck problem. While it is often difficult for an expert to articulate his expertise explicitly and clearly, it is usually relatively easy to document case studies of his skill at work. Learning from examples is also a basis of other learning strategies, such as learning by observation and learning by discovery. The learning systems in commercial use today are almost exclusively inductive ones. The two most widespread families of learning algorithms to date are ID3-like family and AQ11-like family, both being attribute-based. In contrast to the credit assignment and the generate-and-test process in genetic algorithms [Carbonell 90] and numerical activity vectors in connectionist methods

[Dayhoff 90], attribute-based learning has concentrated on symbolic and heuristic computations. These relate to models that operate at the levels of symbols and operations that manipulate symbolic expressions with an emphasis on heuristic rather than computationally explosive optimization strategies. No explicit credit assignment strategies are necessary in the attribute-based induction paradigm.

However, despite some commercial success with existing learning systems, there are limitations to both ID3-like and AQ11-like algorithms for both research and applications. Although a lot of work has been done on the basic ID3 and AQ11 algorithms, as we will analyse in detail in Chapter 2, each of the improvements has also caused new problems. The motivation of this thesis is to take a little-known and inadequate approach (the newly-developed extension matrix approach [Hong 85]), improve it to be competitive with ID3-like and AQ11-like algorithms and do an empirical study of its properties. The principal contribution of this thesis is to describe a heuristic induction algorithm, HCV, based on the extension matrix approach. By dividing the positive examples (PE) of a specific concept or class in a given example set into intersecting groups and adopting a set of strategies to find a heuristic conjunctive formula in each group which covers all the group's positive examples and none of the negative examples (NE), HCV can find a covering formula in the form of a variable-valued logic for PE against NE in low-order polynomial time. A comparison of HCV with the ID3-like algorithms and the AQ11-like algorithms is also made with examples.

Based on the HCV algorithm, an *intelligent learning data base system*, *KEshell2* [Wu 92c], which couples ML techniques with data base and knowledge base technology to implement automatic knowledge acquisition from data bases, is also developed.

1.2 Outline of the Thesis

This thesis is organised as follows.

Chapter 2 provides a review of the current inductive learning research [Wu 93b]. It first summarizes the three typical families of inductive algorithms, the AQ11-like, ID3-like and the extension matrix approach based algorithms, with their main features and recent development being analysed; and then reviews the main theme of this thesis, knowledge acquisition from data bases.

Chapter 3 and Chapter 4 describe the major research contribution of this thesis. In Chapter 3, the HCV algorithm [Wu 92a, Wu 93a] based on the extension matrix approach is designed in detail. It starts with a simple example of attribute-based learning. The strategies adopted in HCV are described and analysed afterwards and a comparison between the algorithm with ID3 is made finally. Chapter 4 provides a performance comparison of HCV with other algorithms, such as ID3, ID5R, ASSISTANT, AQR and CN2, in terms of rule compactness and accuracy on the three famous classification problems, the *MONK's problems*.

Chapter 5 presents an intelligent learning data base system, *KEshell2*, which is an improved version of *KEshell* [Wu 90, Wu 91] with the HCV algorithm above being coupled with data base and knowledge base technology to implement automatic knowledge acquisition from relational data bases. This chapter first gives an introduction to the techniques developed in *KEshell* and then presents *KEshell2* with examples.

1.3 Digest of Conclusions

1. The development of the HCV algorithm provides a reasonable solution to the NP-hard problem in the extension matrix approach for inductive learning. As its time is low-order polynomial, HCV can be seen as one of the fastest learning algorithms to date.

2. The rules in variable-valued logic generated in HCV are shown empirically to be more compact in the term of the number of conjunctive rules than both the decision trees or their equivalent decision rules produced by the ID3-like algorithms and the rules produced by the AQ11-like algorithms although we have not yet theoretical proof.

3. An important result which is clearly highlighted by this thesis is that the goal of creating practical intelligent learning data base systems to implement automatic knowledge acquisition from data bases is no longer difficult or elusive.

Chapter 2

Review of Inductive Learning

2.1 Introduction

AQ11 and ID3 are the two most widespread algorithms in machine learning. They are respectively representatives of the generalisation-specialisation strategy based and the decision tree method based families of inductive algorithms. However, a new family of inductive algorithms (AE1 *et al.*) based on the extension matrix approach has been proposed recently. Although this approach is little known to the machine learning community due to its own inadequacy, we will find in Chapter 3 and Chapter 4 that it can be improved to compete with other learning methods.

This chapter first summarizes AQ11, ID3 and the extension matrix approach and then reviews the main theme of this thesis, knowledge acquisition from data bases.

2.2 AQ11

2.2.1 Developers and background

AQ11 [Michalski *et al.* 78] is a "bottom-up" algorithm designed by Michalski *et al.* It shares the basic generalisation-specialisation strategy with Winston's ARCH program [Winston 70] and Mitchell's candidate elimination algorithm [Mitchell 77, Mitchell 78].

Winston's work on concept learning was performed on his ARCH program. He described a mechanism which learned concepts by looking for relationships between semantic network representations of block world configurations. Two processes were particularly important in his formulation: 1) finding and exploiting commonalities among structural descriptions for the same type of configuration; and 2) finding significant differences between positive examples and negative examples. The two processes correspond to the present terminology *generalisation* and *specialisation*. His ARCH program effectively generalised the representation so as to cover all the positive examples and specialised it so as to exclude all the negative ones. If examples were presented incrementally, then a new positive example triggered generalisation and a new negative example triggered specialisation.

In 1977, Mitchell described a method, called *"candidate elimination"*, which is similar to Winston's method in the sense that it is based on generalisation and specialisation but different in the way in which it explores the solution (or hypothesis) space. In Winston's ARCH program, hypotheses were generated and tested one by one, while in Mitchell's method, hypotheses are systematically deleted from a representation of the entire hypothesis space as they are found to be unsatisfactory. The novel feature of Mitchell's method was the way in which it allowed the hypothesis space to be efficiently represented as a version space.

The aim of Mitchell's method is to ensure that, at all times, the version space contains the complete set of satisfactory representations. A simplified description of the candidate elimination algorithm is as follows. A generalised notion, called *the description identification*, of the version space method is described in [Mellish 91].

Procedure `Candidate Elimination`

```
S1:  Initialise the version space.

S2:  Set G to be the set of most general representations.

S3:  Set S to be the set of most specific representations.

S4:  For each new training example do
```

```
S41:  If it is positive then update S so as to ensure
      that it still contains the set of maximally
      specific, satisfactory representations.
S42:  If it is negative then update G so as to ensure
      that it still contains the complete set of maxi-
      mally general, satisfactory representations.
S43:  If G=S then exit.
```

Return.

AQ11 was a more sophisticated variant of the basic generalisation-specialisation method which could be used to generate representations for a classification function, i.e., a target mapping with more than one output and more than two input groups.

2.2.2 Algorithm description

The input of AQ11 took the form of a set of $(attribute, value)$ pairs, which is why it was claimed to be the earliest attribute-based algorithm. The target output was the classification for the input. The representations produced for the target mapping were rules written in an extended logic notation, called *variable-valued logic*[1] [Michalski 75], one rule for each distinct concept[2].

Suppose the training set is subdivided into n subsets with each subset containing the inputs which should evoke a particular output, AQ11 works as follows.

```
For each subset do

    S1:  Convert all the examples in the subset into positive
         examples.

    S2:  Convert all the other examples into negative examples
         and use previously generated rules as dummy negative
         examples.

    S3:  Apply generalisation-specialisation and a set of
         heuristic strategies [Hong 89a] to the training examples
         so produced.
```

[1]The variable-valued logic developed by Michalski is a calculus for representing decision problems where decision variables can take on some range of values. Its principal syntactic entity is a *selector* with the general form

$$[X \# R] \tag{2.1}$$

where X is a variable or attribute, $\#$ is a relational operator (such as $=, \neq, <, >, \leq$, and $\geq$), and R, called a *reference*, is a list of one or more values that X could take on. A well-formed rule in the logic is similar to a production rule but with selectors as the basic components of both its left-hand and right-hand sides.

[2]In PLANT/ds [Michalski *et al.* 80] where AQ11 has been first successfully applied, for example, a concept is one of the fifteen soybean diseases.

2.2.3 Application

AQ11 is known for succeeding in automatic construction of the knowledge base in the expert system PLANT/ds [Michalski *et al.* 80]. It was the successful application of AQ11 on soybean diagnosis that provided the first clear demonstration of the application potential of ML techniques.

In the soybean experiment, the inputs were descriptions of diseased soybean plants and the target outputs were the names of diseases. A striking result of the experiment was the fact that the rules derived by AQ11 actually out-performed human experts[3]. The inductively derived rules achieved 100% correct diagnoses while rules derived from experts achieved only 96.2% correctness.

The experiment showed that the basic generalisation-specialisation method could be used to generate representations for a classification function. This clear demonstration of the power of ML techniques helped to establish ML as a major subfield of AI with serious, practical applications. It demonstrated the way in which ML could be used to solve realistic problems.

2.2.4 Advantages and disadvantages

The main advantage of AQ11 is that it will make sure that the maximally general representations do not cover any negative representations. In other words, it will make sure that the general boundary does not overlap any preciously constructed general boundaries. However, there are various problems with AQ11.

First, as we can see from its Step 2, it is only possible when rules are of the same syntactic form as examples.

Second, the rules produced by AQ11 will vary depending on the order in which the training subsets are presented. In general, the first rule induced will be more general than the ones produced later.

---

[3]Although AQ11 produced rather large and unwieldy rules while the rules derived from the experts tended to be much shorter.

Third, the AQ11 algorithm is computationally more expensive in both the cost of rule production and the complexity of rules produced than ID3, although the rules produced in AQ11 in the form of variable-valued logic are said to be more comprehensible [O'Rorke 82].

Finally, another drawback of the AQ11 algorithm is that being bottom-up in nature it is prone to disruption by noisy data where an example turns up in more than one subset.

2.3 ID3

2.3.1 Developer and background

ID3 [Quinlan 79] is a "top-down" algorithm developed by Quinlan out of the Concept Learning System (CLS) by Hunt [Hunt *et al.* 66].

CLS is a learning mechanism which accepts a set of training pairs and constructs a representation in the form of a decision tree, which is equivalent to a disjunctive rule. The decision tree is structured so that each leaf node has a target output associated with it. An arbitrary input is processed by simply applying the tree to the input (i.e. propagating the input down through the tree). This produces a leaf node which in turn yields the target output.

Main steps in the CLS algorithm can be described as follows.

```
S1:  T←the whole training set.

   Create a T node.

S2:  If all examples in T are positive, create a 'yes' node

   with T as its parent and stop.

S3:  If all examples in T are negative, create a 'no' node

   with T as its parent and stop.
```

```
S4:  Select an attribute X with values v_1, ..., v_N and partition
     T into subsets T_1, ..., T_N according to their values on X.
     Create N T_i nodes (i = 1, ..., N) with T as their parent
     and X = v_i as the label of the branch from T to T_i.
S5:  For each T_i do:  T <- T_i and goto S2.
```

2.3.2 Algorithm description

Quinlan modified the CLS algorithm in two ways.

First, he added a process known as *windowing*. This was designed to enable the algorithm to cope with very large training sets.

If the training set is very large then, rather than process the entire set in one, it may be more efficient to process a small sample first. If the sample is a representative of the complete set, the decision tree produced will be similar to the one which we would get by processing the entire training set. Once we have produced a tentative tree we can then gradually perfect it. To do this we simply search through the training set looking for any *(input, output)* pairs which are not properly represented and each time we find such an exception we modify the tree appropriately. However, in recent work, windowing does not feature very strongly. Some evidence [Wirth *et al.* 88] suggests that windowing typically provides very little benefit.

Second, and more importantly, Quinlan devised an information theoretic heuristic (the *entropy* measure) which decided how to split the inputs at each stage of the tree growing process thus enabling smaller and therefore more efficient decision trees to be constructed.

ID3 works as follows.

Suppose $T = PE \cup NE$ where PE is the set of positive examples and NE is the set of negative examples, $p = |PE|$ and $n = |NE|$. An example e will be determined to belong to PE with probability $p/(p+n)$ and NE with probability

$n/(p+n)$. By employing the information theoretic heuristic, a decision tree is considered as a source of message, "PE" or "NE", with the expected information needed to generate this message, given by

$$I(p,n) = \begin{cases} -\frac{p}{p+n} log_2 \frac{p}{p+n} - \frac{n}{p+n} log_2 \frac{n}{p+n} & when\ p \neq 0\ \&\ n \neq 0 \\ 0 & otherwise. \end{cases} \tag{2.2}$$

If attribute X with value domain $\{v_1, ..., v_N\}$ is used for the root of the decision tree, it will partition T into $\{T_1, ..., T_N\}$ where T_i contains those examples in T that have value v_i of X. Let T_i contain p_i examples of PE and n_i of NE. The expected information required for the subtree for T_i is $I(p_i, n_i)$. The expected information required for the tree with X as root, $EI(X)$, is then obtained as weighted average.

$$EI(X) = \sum_{i=1}^{N} \frac{p_i + n_i}{p+n} I(p_i, n_i) \tag{2.3}$$

where the weight for the i-th branch is the proportion of the examples in T that belong to T_i. The information gained by branching on X, $G(X)$, is therefore

$$G(X) = I(p,n) - EI(X). \tag{2.4}$$

ID3 examines all candidate attributes, chooses X to maximize G(X), constructs the tree, and then uses the same process recursively to construct decision trees for residual subsets $T_1, ..., T_N$. For each T_i ($i = 1, ..., N$): if all the examples in T_i are positive, create a 'yes' node and halt; if all the examples in T_i are negative, create a 'no' node and halt; otherwise select another attribute in the same way as above.

2.3.3 Application

ID3 has been shown [O'Rorke 82] to be computationally cheaper in both the cost of rule production and the complexity of rules produced than the AQ11 algorithm although the rules produced by AQ11 are said to be more comprehensible than the decision tree representation used in the ID3-like algorithms. The decision tree method based family of algorithms, also called *divide-and-conquer* methods [Quinlan 91], have been incorporated into a number of commercial systems including *1st Class, ACLS, Expert-Ease, ExTran* and *RuleMaster*.

2.3.4 Advantages and disadvantages

One of the great advantages of the ID3 method is the fact that it does not require users to specify background knowledge in the form of, say, generalisation hierarchies[4]. This means that ID3 can be applied to any syntactically well-formed training set. This, together with the high performance of the algorithm, has enabled ID3 to form the central component in several commercial packages.

Like AQ11, ID3 has some limitations:

First, its decision tree representation is less convenient for manipulations than the variable-valued logic in AQ11 and production rules when a single decision tree is not sufficient to represent all the expertise of a domain. In this case, a number of decision trees of the domain need to be converted into decision rules before they can be used by a rule-based system. Although the conversion of decision trees to rules is not very difficult if we do not try to simplify the rules produced (see Section 2.4.6), the rules transformed from decision trees are still too simple to express things like memberships. Of course, for those domains where a decision tree is sufficient to express the expertise, we can use the decision tree directly by designing a non-rule-based problem solver.

Second, once an attribute is selected, all arcs labeled by values that attribute takes must be expanded. This can make resulting paths longer than those actually needed because, by the time specific concepts (leaves on the decision tree) are developed, irrelevant variables may have been introduced.

Third, the number of branches (paths) might still be large since at each arc, only one value can be labeled.

---

[4] Núñez argued that for this reason, most of the time the ID3 family of algorithms are neither logical nor understandable to experts and he made some improvements (i.e., executing different types of generalisation and reducing the classification cost) on ID3 in his algorithm by means of background knowledge in [Núñez 91].

Finally, although the information theoretic heuristic can usually generate efficient decision trees, ID3 is still heuristic, which means it is not guaranteed to find the simplest decision tree that characterizes the given training instances because the information theoretic heuristic is by no means complete, and suffers from excessive complexity [Utgoff 89] and is therefore usually incomprehensible to experts since it needs to examine all candidate attributes to choose one at each non-leaf node.

2.4 Recent Development of AQ11 and ID3

2.4.1 Introduction

The original AQ11 and ID3 algorithms have been extended in several ways to improve their various capacities such as noise handling and incremental induction in their successors such as AQ15, CN2, ID5R and C4. This section gives an account of these developments.

2.4.2 Noise handling

The Achilles heel of AQ11 is its inability to handle noise. Two approaches have been tried to overcome this difficulty. The first approach [Michalski *et al.* 86] uses a partial match procedure, called TRUNC, to execute rules in AQ15, while the second approach, CN2 [Clark *et al.* 89], couples the entropy measure adopted in ID3 to produce rules in domains where problems of poor description language and/or noise may be present.

ID3 can be easily adapted to handle noise by virtue of its top-down approach to decision tree construction. During induction, all possible attribute tests are considered when growing a leaf in a decision tree and the entropy measure is used to choose one to place at each node. In noisy environments, we can halt tree growth when no more significant information gain can be found. ID3's capacity to handle noisy data has been studied in [Quinlan 86a] and [Quinlan *et al.* 86]. Noise handling in decision tree method based induction algorithms has been studied independently as a statistical technique [Breiman *et al.* 84] and shows a convergence between ML research in AI and statistics [Gams *et al.* 91, Gammerman *et al.* 91].

However, the results produced by noise-tolerant algorithms such as AQ15 and CN2 are usually not completely consistent with the given training examples, which means those algorithms cannot guarantee to generate exact rules or decision trees in noise-free problems. This is not acceptable in cases where we need consistent rules to correctly classify all known examples.

2.4.3 Incremental learning

There are several common problems in all kinds of inductive learning algorithms:

1. First, when an example set is very large, how can they speed up their learning processes?
2. Second, when an example set is not a static repository of data, e.g. examples may be added, deleted, or changed, the induction on the example set cannot be a one-time process, so how can induction algorithms deal with the changing examples?
3. Finally, when some inconsistency (e.g. noise) is found in an example set or a knowledge base just produced, how can they remove it?

One possible way to solve those problems is incremental learning, which means dividing a large example set into a number of subsets and treating each subset each time. Although no existing algorithm has found a complete solution to those problems, a lot of work has been done in this direction. For instance, AQ11 has taken advantage of different subsets. The AQ11 based AQ15 [Hong *et al.* 86, Michalski *et al.* 86], ID4 [Schlimmer *et al.* 86], ID5R [Utgoff 89] and the windowing technique in ID3 can be viewed as good examples of research on incremental learning.

Generally speaking, incremental induction usually takes more time because it needs to restructure decision trees or rules when some new examples do not fit the decision trees or rules developed so far. This is a common trade-off between time and space in Computer Science.

2.4.4 Constructive learning

Neither AQ11-like nor ID3-like algorithms need explicit, built-in background knowledge. That is why they are sometimes called empirical learning methods, which are different in nature from the knowledge-rich learning methods, such as AM [Lenat 79] and EURISKO [Lenat 83] developed by Lenat, learning by analogy (or case-based reasoning [Kolodner 92]), explanation-based learning [DeJong *et al.* 86, Mitchell *et al.* 86], and inductive logic programming [Muggleton 90].

However, there is always implicit background knowledge embedded in the formulation of solution spaces and in the representation of examples. When a solution space turns out to be inadequate, which we often call the imperfect-knowledge problem, representation modification is needed and the modification process typically involves searching for useful new descriptive features (constructive induction) in terms of existing features or attributes. AQ17 [Bloedorn *et al.* 92, Thrun *et al.* 91] of the AQ11-like family has been developed to implement iterative construction of new attributes based on existing ones.

Constructive learning [Michalski 86, Muggleton 88, Matheus 89, Mehra *et al.* 89, Matheus *et al.* 89, Bala *et al.* 92] has become a strong theme in inductive learning research. One of the difficulties in constructive learning is that the complexity in some cases (such as iterative feature construction) is extreme but there are situations in which it is a necessary part of learning [Thornton 91].

2.4.5 Decision tree pruning

The basic ID3 algorithm tends to construct exact decision trees. However, in many real-world problems such as medical diagnosis and image recognition, the classification cannot be exact due to noise and/or uncertainty in data. As a result, a constructed tree by ID3 may not be able to capture the proper relations in data. Decision tree pruning mechanisms have been designed in many systems such as ASSISTANT [Cestnik *et al.* 87] and C4 [Quinlan *et al.* 87] to prevent this phenomenon. Once a non-leaf subtree meets a specific criterion (e.g. with an equal or smaller number of misclassifications), it is replaced by a leaf.

Pruning can usually simplify decision trees. The simplified trees can sometimes classify more accurately unseen cases in noisy environments [Quinlan 89a]. However, pruning decision trees is something very similar to the TRUNC technique adopted in AQ15 and noise handling in other algorithms mentioned in Section 2.4.2. It can simplify decision trees in noisy environments, but not improve the induction algorithms used to construct the trees. Also, we do not expect it to work properly in noise-free environments.

2.4.6 Decompiling decision trees into production rules

Decompiling decision trees has been studied in [Corlett 83, Quinlan 87, Quinlan 89a] and implemented in C4 [Quinlan *et al.* 87] and C4.5 [Quinlan 92]. It contains three basic steps:

1. Traverse a decision tree to obtain a number of conjunctive rules. Each path from the root to a leaf in the tree corresponds to a conjunctive rule with the leaf as its conclusion.

2. Check each condition in each conjunctive rule to see if it can be dropped without more misclassification than expected on the original training examples or new test examples.

3. If some conjunctive rules are the same after Step 2, then keep only one of them.

Transformation of decision trees to production rules provides a way of combining different trees into the same knowledge base for more complicated domain.

The final production rules produced are sometimes both simpler than the original decision trees and more accurate when classifying new examples in noisy environments [Quinlan 87].

However, dropping conditions from the decision-tree-traversal rules in Step 2 is something like a new induction algorithm which can work on the original example sets but in a way totally different from the ID3-like algorithms. Therefore, the time complexity for the transformation is expensive. Also, Appendix A shows an example set where no conditions can be dropped from the decision-tree-traversal rules. In this case, Step 2 is redundant.

2.4.7 Binarization of decision trees

Binarization in CART [Breiman *et al.* 84], ASSISTANT [Cestnik *et al.* 87] and NewID [Boswell 90] groups the attribute values into two subsets. It can usually produce smaller decision trees.

However, as indicated in [Quinlan 88b], there are two major problems in those systems. Firstly, binarization could lead to decision trees that are even more unintelligible to human experts than the ordinary case due to unrelated attribute values being grouped together and multiple tests on the same attributes in the binary decision trees. Secondly, binarization requires a large increase in computation to properly split the attribute values.

2.4.8 A new selection criterion for decision tree construction

As ID3 has been found to operate unsatisfactorily when there are attributes with varying numbers of discrete possible values, [Quinlan 88b] proposes a new heuristic, called *the gain ratio criterion*, instead of the entropy measure adopted in ID3 for selecting tests in decision tree generation.

Here again, in some cases as shown in [Quinlan 88b], the new gain ratio criterion can outperform the entropy measure. However, in many other cases, even when there are attributes with varying numbers of discrete possible values, the new criterion cannot improve the decision trees produced by ID3 at all. Appendix B shows two decision trees produced by the new gain ratio heuristic and the entropy measure respectively on the same example set in Table B-1. The decision tree produced by the new criterion (Figure B-2) is a little more complicated than the decision tree produced by the original ID3 algorithm (Figure B-1): both need seven conjunctive rules (paths from the root to terminals in each decision tree) but the new criterion needs one more conjunction. Appendix B is also empirical proof of ID3's heuristic rather than complete property: a decision tree made by hand (Figure B-3) is clearly smaller than the tree produced by ID3.

2.4.9 Structured induction

ID3's simplicity is largely attributable to the following two restrictions placed on its application domains [Quinlan 88a]:

- The induction is a kind of classification, i.e., the knowledge we are trying to capture is that of assigning a case to one of a set of mutually exclusive classes.

- Each case is described in terms of a fixed collection of properties or attributes. An attribute may have a small set of discrete possible values or might be a real-valued numerical variable.

This limitation of ID3 exists also in both the AQ11-like family of algorithms mentioned above and the extension matrix approach based family of algorithms below for complex applications.

Although induction offers a considerable short cut in comparison to those methods of rule generation such as explanation-based learning and inductive logic programming which couple both deduction and induction, decision tree method based algorithms provide large decision trees that are opaque to the user in large problem domains. Shapiro [1987] has developed the technique of structured induction.

The basic idea is to split the whole complex problem which might be very large in size into a number of subproblems by using domain knowledge and apply the ID3 algorithm to each of the subproblems. Shapiro's work concerned solutions to the chess endgames King and Pawn vs. King (black-to-move) and King and Pawn vs. King and Rock (white-to-move, white pawn on a 7) as trial problems of measurable complexity. The resultant systems contained humanly understandable decision trees which were synthesised from expert supplied examples.

Shapiro (personal communication, May 23, 1992) has recently also developed an industrial-strength code generator (which can be used to remove the need for any inference engine to run the resulting decision trees or rules generated by ID3) with target languages being C, COBOL (85 and VSII), and REXX and is delivering applications into large US corporations with greater than 80% automatically generated code. The code generator is said to be able to code standard transaction-based processing by the provision of examples and code-fragments.

The structured induction technique can also be coupled with both the AQ11-like family of algorithms mentioned above and the extension matrix approach based family of algorithms below.

2.4.10 Conclusions

In addition to the development mentioned above, other features such as handling real-valued attributes [Paterson *et al.* 82, Fayyad *et al.* 92] have also been studied. However, as we have analysed above, although each of them is useful in some specific cases, none of the extensions have generally improved AQ11 or ID3 in noiseless environments. AQ11's time complexity and rule compactness in noise-free domains have not been improved at all. The core of the decision tree method based family of algorithms is still the entropy measure to select an attribute by examining all candidate attributes during the splitting of examples.

2.5 The Extension Matrix Approach

2.5.1 Developers and background

The new family of inductive algorithms based on the extension matrix approach was first developed in the University of Illinois by Hong *et al.* [Hong 85, Hong *et al.* 87] and then redesigned by the author [Wu 92a]. In contrast to the generalisation-specialisation strategy in AQ11-like algorithms and the decision tree method in ID3-like algorithms, the algorithms of the extension matrix approach based family take a new kind of matrix, called *an extension matrix*, as their mathematical basis.

2.5.2 Terminology and notation

Let a be the number of attributes $\{X_1, ..., X_a\}$ in an example space, n be $|NE| = | \{e_1^-, ..., e_n^-\} |$ where e_i^- $(i = 1, ..., n)$ is the i-th negative example and p be $|PE| = | \{e_1^+, ..., e_p^+\} |$ where e_i^+ $(i = 1, ..., p)$ is the i-th positive example. Let NE be expressed by

$$NEM = \{e_1^-, ..., e_n^-\}^T = (r_{ij})_{n*a} \tag{2.5}$$

with the i-th negative example $e_i^-(i = 1, ..., n)$ being expressed on the i-th row of matrix NEM and $NEM(i, j) = r_{ij}$ indicating the value of e_i^- on attribute X_j is r_{ij}.

Definition 2.5.1. Let the k-th $(k = 1, ..., p)$ positive example be expressed as $e_k^+ = (v_{1_k}^+, ..., v_{a_k}^+)$, the matrix below is the *extension matrix* of e_k^+ against NE

$$EM_k = (r_{ij_k})_{n*a} \tag{2.6}$$

where

$$r_{ij_k} = \begin{cases} * & when\ v_{j_k}^+ = NEM_{ij} \\ NEM_{ij} & when\ v_{j_k}^+ \neq NEM_{ij} \end{cases}$$

and '$*$' denotes a dead element which cannot be used to distinguish the positive example from negative examples.

Definition 2.5.2. In an EM_k, a set of n nondead elements r_{ij_i} $(i = 1, ..., n,$ $j_i \in \{1, ..., a\})$ that come from the n different i rows is called a *path* in the extension matrix.

Lemma [Hong 89a]. A path $\{r_{1j_1}, ..., r_{nj_n}\}$ in an EM_k corresponds to a conjunctive formula

$$L = \bigwedge_{i=1}^{n} [X_{j_i} \neq r_{ij_i}] \tag{2.7}$$

which covers e_k^+ against NE and *vice versa.*

Each $[X_{j_i} \neq r_{ij_i}]$ here is a *selector* in variable-valued logic. If r_{ij_i} appears on m $(m \in \{0, ..., n\})$ rows in the same column j_i of an EM_k, we say it or $[X_{j_i} \neq r_{ij_i}]$ covers m rows of the EM_k.

Definition 2.5.3. Matrix $EMD = (r_{ij})_{n*a}$ with

$$r_{ij} = \begin{cases} * & when\ \exists k_1 \in \{i_1, ..., i_k\} : EM_{k_1}(i,j) = * \\ \vee_{k_2=1}^{k} EM_{i_{k_2}}(i,j) = NEM(i,j) & otherwise \end{cases} \tag{2.8}$$

is called the *disjunction matrix* of the positive example set $\{e_{i_1}^+, ..., e_{i_k}^+\}$ against NE or the disjunction matrix of $EM_{i_1}, ..., EM_{i_k}$.

Definition 2.5.4. In the EMD of a positive example set $\{e_{i_1}^+, ..., e_{i_k}^+\}$ against NE, a set of n nondead elements r_{ij_i} $(i = 1, ..., n, j_i \in \{1, ..., a\})$ that come from the n different i rows is also called a *path.*

Theorem 2.5.1. A path $\{r_{1j_1}, ..., r_{nj_n}\}$ in the EMD of $\{e_{i_1}^+, ..., e_{i_k}^+\}$ against NE corresponds to a conjunctive *formula* or *cover*

$$L = \bigwedge_{i=1}^{n} [X_{j_i} \neq r_{ij_i}] \tag{2.9}$$

which covers all of $\{e_{i_1}^+, ..., e_{i_k}^+\}$ against NE and *vice versa.*

Proof. If $EMD(i,j) = *$, there must be a $k_2 \in \{i_1, ..., i_k\}$ and $EM_{k_2}(i,j) = *$, which means there is no common nondead element on the (i,j)-position of all the extension matrixes $EM_{i_1}, ..., EM_{i_k}$. If there exists no dead element on the (i,j)-position of any $EM_{k_2}(k_2 \in \{i_1, ..., i_k\})$, it is certain that $EM_{k_2}(i,j) = NEM(i,j)$ according to Definition 2.5.1 and that the *NEM(i,j)* is a common nondead element in $EM_{i_1}, ..., EM_{i_k}$ according to Definition 2.5.3. Therefore, the common nondead elements in all the extension matrixes $EM_{i_1}, ..., EM_{i_k}$ and the nondead elements in their disjunction matrix EMD correspond to each other and each common path in $EM_{i_1}, ..., EM_{i_k}$ formed by the common nondead elements in every $EM_{i_{k_2}} (k_2 = 1, ..., k)$ corresponds to a path in EMD and *vice versa*. According to the above Lemma, the formula which corresponds to a path in EMD must be a common formula for all of the $\{e^+_{i_1}, ..., e^+_{i_k}\}$ against NE.

If there is no path which covers all the n rows in EMD, there is no common path and therefore no conjunctive formula cover in all the extension matrixes $EM_{i_1}, ..., EM_{i_k}$.

Definition 2.5.5. If there exists at least one path in the EMD of a positive example set $\{e^+_{i_1}, ..., e^+_{i_k}\}$ against NE, all the positive examples in the set intersect and the positive example set is called an *intersecting group.*

Theorem 2.5.2. For a given set of examples, if PE and NE are persistent, which means they contain no common examples, there always exists at least one conjunctive formula covering any positive example $e^+_k \in PE$ against NE.

Proof. As PE and NE are persistent, we can always find at least one nondead element on each row i in the EM_k of e^+_k against NE which discriminates e^+_k and the i-th negative example e^-_i in NEM. Therefore, we can always find at least one path which corresponds to a conjunctive formula cover in EM_k.

From Definition 2.5.5 and the proof process of Theorem 2.5.2, we can easily get the following corollary.

Corollary. For a given set of examples, if PE and NE are persistent, there always exists at least one conjunctive formula cover for each intersecting example group.

2.5.3 Optimization problems

There are two striking optimization problems in the extension matrix approach:

- The minimum formula (MFL) problem: Generating a conjunctive formula that covers a positive example or an intersecting group of positive examples against NE and has the minimum number of different conjunctive selectors.

- The minimum cover (MCV) problem: Seeking a cover which covers all positive examples in PE against NE and has the minimum number of conjunctive formulae with each conjunctive formula being as short as possible.

As the extension matrix EM_k of each positive example e_k^+ against NE contains all such paths that each correspond to a conjunctive formula of e_k^+ against NE and an optimal cover of PE against NE is such a minimum set of formulae that is a logical combination of all the formulae from every EM_k ($k = 1, ..., p$), both MFL and MCV problems have been proved to be NP-hard [Hong 85].

Two complete algorithms, MFL and MCV, are designed to solve the optimization problems MFL and MCV in paper [Wu 92a] with $O(na2^a)$ and $O(n2^a2^{2^a} + pa^2 2^{2a})$ time respectively when each attribute domain $D_i (i = 1, ..., a)$ satisfies $|D_i|= 2$. When there exists $|D_j|>2$ or D_j is a real-valued interval ($j \in \{1, ..., a\}$), a decomposition method that decomposes D_j into several sub-domains whose bases are each two is also designed there.

2.5.4 Heuristic strategies in AE1

As the nature of the MFL and MCV problems is NP-hard, when an example set or an attribute space is large the induction process based on the complete algorithms will become computationally intractable. Two strategies are adopted in AE1 to find approximate rather than optimal solutions for both MFL and MCV problems [Hong 85]:

1. Starting search from the columns with the most nondead elements, and

2. Simplifying redundance by deductive inference rules in mathematical logic.

There are two problems in AE1. First, its first strategy can easily lose optimal solutions in some cases. Taking the simple extension matrix below as an example, the first heuristic strategy in AE1 cannot produce the optimal formula ($[X_1 \neq 1] \wedge [X_3 \neq 1]$) since it will choose the selector $[X_2 \neq 0]$ at first. Second, simplifying redundance for MFL and MCV problems is NP-hard. No heuristic strategy for this process has been reported.

$$\begin{pmatrix} 1 & * & * \\ * & 0 & 1 \\ 1 & 0 & * \\ * & 0 & 1 \\ 1 & 0 & * \\ * & * & 1 \end{pmatrix}$$

2.5.5 Advantages and disadvantages

Generally speaking, the extension matrix approach is still little known due to its own inadequacy described above. The developer of AE1 has recently developed an AE5 system [Hong 89b] based on AE1 but the basic algorithm remains to be the same. The only difference between AE5 and AE1 is that some facilities such as constructive and incremental induction have been added to the AE5 system. So, the two major problems of AE1 described above also apply to AE5. However, a detailed comparison between the improved extension matrix approach with ID3 and AQ11 will be given in Chapter 3 after the HCV algorithm of this family has been introduced.

2.6 Learning from Data Bases

2.6.1 Introduction

Although some commercial successes have been found in existing learning systems, there are limitations on current ML programs. Existing knowledge acquisition tools (such as [Mowforth 86, Boose *et al.* 88, Marcus 88, Piatetsky-Shapiro *et al.* 91, Frawley *et al.* 92]) have concentrated on building knowledge bases for expert systems and designing various learning algorithms. As data base technology has already found wide application in many fields, ML research obviously stands to gain from this greater exposure and established technological foundation. Two fundamental questions arise here. Does data base technology need ML techniques? If so, how can they be coupled successfully?

2.6.2 Does data base technology need ML?

On one hand, as data bases grow in both number and size, the prospect of mining them for new, useful knowledge becomes yet more enticing [Quinlan 89b]. On the other hand, despite its commercial success, conventional data base technology has limitations in many advanced applications (see Appendix D). That is why integrating AI technology into data base technology, called IDB (intelligent data base) research, has been identified [Brodie 88] as one of the research frontiers of data base technology and has become a popular research topic all over the world. There are five directions in the current IDB research: object-oriented data base systems; deductive data base systems; expert data base systems; intelligent man-machine interfaces which include the design of meaningful operation interfaces and of friendly natural-language interfaces; and recursive query optimization. The knowledge bases (which contain deductive rules and/or semantic information such as the conceptual hierarchy among data) in existing IDB systems can only be built up by hand with known technology. Knowledge acquisition in IDB systems has become a central and difficult problem in IDB research. Research in this area is expected to lead to significant progress in the whole data base field.

2.6.3 How can ML be well coupled with data bases?

Broadly speaking, all kinds of attribute-based learning algorithms can be adapted to extract knowledge from data bases. It is not difficult to add an induction engine to an existing data base system in an *ad hoc* way (such as [Cai *et al.* 91] and [Ke *et al.* 91]) to implement rule induction from data bases or design some specific engines to learn from domain-specific data sets (e.g. [Blum 82]). However, when we integrate ML techniques into data base systems, we must face many problems [Quinlan 89b] such as:

- The knowledge learned needs to be tested and/or used back in the integrated IDB systems. This implies more expressive representations for both data (e.g. tuples in relational data bases, which represent instances of a problem domain) and knowledge (e.g. rules in a rule-based system, which can be used to solve users' problems in the domain) and deduction/inference mechanisms are needed.

- More efficient induction algorithms are needed. The algorithms should be capable of being applied to realistic data bases, e.g. $\geq 10^6$ relational tuples. This needs the algorithms to be more efficient than existing ones. Exponential or even medium-order polynomial complexity will not be of practical use.

- Another problem is how to balance ML facilities and other functions in the IDB systems, particularly when is the proper time to trigger the ML facilities.

The first and the third problems both concern how to couple ML facilities with data base and knowledge base technology in IDB systems. This is the main difficulty in developing practical IDB systems. However, the second problem concerning low-order polynomial time induction algorithms is the crucial requirement for knowledge acquisition from data bases. Although a lot of work (e.g. [Cai *et al.* 91], [Ke *et al.* 91] and various induction algorithms [Bundy *et al.* 85, McDonald 89, Muggleton 86, Langley 89, Wu 93b]) has been done, the requirements above for knowledge acquisition from realistic data bases are still far away for existing systems to reach and no existing systems have been reported to be able to integrate effectively ML techniques with both data base and knowledge base technology.

In this thesis, we define an integrated learning system, which couples ML techniques with both data base and knowledge base technology, as *an intelligent learning data base system* (ILDB) if it provides mechanisms for

1. translating standard (e.g. relational) data base information into a form suitable for use by its induction engines,

2. using induction techniques to extract rules from data bases, and

3. interpreting the rules produced to solve users' problems.

With an ILDB system, one can, for example, produce a small number of conjunctive rules for some diseases from a large medical cases of these diseases. The ILDB system can then use the rules in two different ways: keeping these rules instead of the original cases because the original cases might take up a large space; and using these rules to diagnose new cases.

Chapter 3

HCV: A Heuristic Covering Algorithm

3.1 Introduction

Time complexity and description compactness[1] are two important criteria for all induction algorithms. In the extension matrix approach, there are two extreme strategies, each of which places special emphasis on only one of the two criteria. The first is finding all possible formulae from each positive example's extension matrix first and then taking an exhaustive search among all the formulae to find the shortest combination which covers all the positive examples. This strategy can give the shortest description in the form of variable-valued logic but works in exponential time. The second is simply separating one positive example from NE by "memorizing" the positive example or all positive examples in PE from NE by "memorizing" each of the positive examples into a Boolean OR formula. This trivial heuristic can work quickly but generates an extremely large description. An OR formula of this kind cannot be used directly to classify new examples which have not been presented in the training example set while simplifying it into the shortest form also needs NP-hard time. Therefore, a good learning algorithm should be able to either avoid the NP-hard time or produce a briefer description which is at least able to correctly classify the PE and NE in a given training

---

[1]As we have defined in Section 2.5.3, the measures for description compactness adopted in this thesis are 1) the number of conjunctive formulae or rules, and 2) the number of all conjuncts or selectors in all the conjunctive rules.

example set. In this chapter, we will show that the HCV algorithm has made progress on both the time and the description compactness.

For the MFL problem introduced in Section 2.5.3, a heuristic algorithm, HFL, is specifically designed to find a conjunctive formula for an extension matrix or a disjunction matrix of an intersecting group of positive examples. Three of the four strategies adopted in HFL are complete and the fourth one is a reasonable heuristic. The HCV algorithm is a heuristic induction algorithm designed for the MCV problem also mentioned in Section 2.5.3. By partitioning the positive examples (PE) in a given example set into intersecting groups and calling HFL to find a heuristic conjunctive formula in each group which covers all the positive examples in the group and none of the negative examples (NE), it can find a covering formula in the form of variable-valued logic for PE against NE in low-order polynomial time.

This chapter presents the algorithm in detail. As the AQ11 algorithm has been shown [O'Rorke 82] to be more expensive in both the cost of rule production and the complexity of rules produced than the ID3 algorithm and its heuristic strategies are much more complicated [Hong 89a], we will concentrate the comparison of our HCV algorithm on the ID3 algorithm in terms of time complexity and rule compactness in this chapter. More algorithms will be compared with HCV in Chapter 4.

3.2 An Example of Attribute-based Learning

Given a discrete finite attribute space of a dimensions, $E = D_1 \times ... \times D_a$, where each D_j ($j = 1, ..., a$) is a finite set of symbolic values or a numerical interval, an example, or a case, e=($V_1, ..., V_a$) is an element of E means $V_j \in D_j$. A positive example is such an example that belongs to a known class which, say, has a specific name in E. All the other examples which do not belong to the known class can be called negative examples (NE) at the moment we are considering the known class. The induction task is to generate a description, say production rules or a decision

Table 3–1: Cases of *Pneumonia* and *Tuberculosis*

ORDER	*FEVER*	*COUGH*	*X-RAY*	*ESR*	*AUSCULTATION*	**DISEASE**
1	high	heavy	flack	normal	bubble-like	
2	medium	heavy	flack	normal	bubble-like	
3	low	slight	spot	normal	dry-peep	*Pneumonia*
4	high	medium	flack	normal	bubble-like	
5	medium	slight	flack	normal	bubble-like	
6	absent	slight	strip	normal	normal	
7	high	heavy	hole	fast	dry-peep	
8	low	slight	strip	normal	normal	*Tuberculosis*
9	absent	slight	spot	fast	dry-peep	
10	low	medium	flack	fast	normal	

tree, that covers all of the positive examples (PE) against[2] NE or classifies them correctly.

Example 3-1 (from [Hong 89a]). Given sets PE and NE of cases of *Pneumonia* and *Tuberculosis* in Table 3-1. Each case is described by five attributes: *fever* {absent, low, medium, high}, *cough* {slight, medium, heavy}, shape of focus shown by *X-ray* {spot, strip, flack, hole}, erythrocyte sedimentation rate *(ESR)* {normal, fast}, and sound of heart by *auscultation* {normal, dry-peep, bubble-like}.

Running ID3, we get a decision tree in Figure 3-1[3] which is equivalent to the

[2]'against' is used to mean that the description should cover none of the negative examples.

[3]Notice that the information gains for *FEVER* and *ESR* are the same on the 3rd, 7th, and 9th examples after the *AUSCULTATION* attribute has been chosen. ID3 has no specific strategy to deal with this situation, so our implementation simply selects the attribute which is presented first in the table. Other implementations may use different strategies (e.g. always choosing the last attribute presented) and therefore produce slightly different results. Although in our current example set, choosing the last attribute leads to a slightly better decision tree, we can easily give further examples where the reverse is the case.

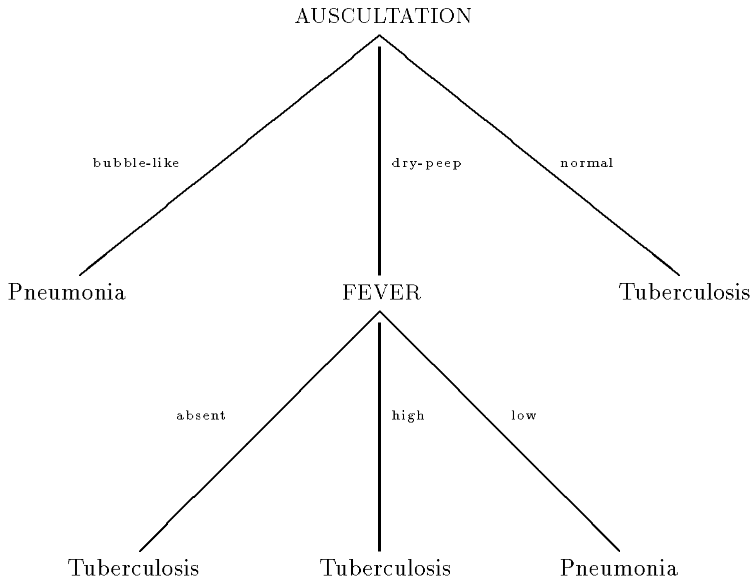

Figure 3–1: A decision tree (by ID3) for Example 3-1

conjunctive decision rules below (the conditions in the **bold** type style can be dropped):

if *AUSCULTATION=bubble-like* then *Pneumonia*;

if **AUSCULTATION=dry-peep** & *FEVER=absent*
then *Tuberculosis*;

if *AUSCULTATION=dry-peep* & *FEVER=high* then *Tuberculosis*;

if *AUSCULTATION=dry-peep* & *FEVER=low* then *Pneumonia*; and

if *AUSCULTATION=normal* then *Tuberculosis*.

Running the HCV algorithm in Section 3-4, we can get the following rule in variable-valued logic for *Pneumonia* against examples of *Tuberculosis*:

[ESR=normal]

[AUSCULTATION $\in$ *{bubble-like, dry-peep}]*

$\rightarrow$

[DISEASE=Pneumonia].

A comparison in more detail between ID3 and HCV will be given in Section 3.6.

3.3 The HFL algorithm

The HFL algorithm is designed to find a heuristic conjunctive formula which corresponds to a path in an extension matrix or a disjunction matrix when there is at least one path in the disjunction matrix. As a disjunction matrix can be processed in the same way as an extension matrix to find its conjunctive formulae, we will only refer to the extension matrixes below.

3.3.1 Four strategies in HFL

Four strategies are adopted in the HFL algorithm:

1. The *fast* strategy. In an extension matrix $EM_k = (r_{ij})_{n*a}$, if there is no dead element in a (say j) column, then $[X_j \neq r_j]$ where $r_j = \vee_{i=1}^{a} r_{ij}$ is chosen as the one selector cover for EM_k.

   For example, selector $[AUSCULTATION \neq \{normal, dry - peep\}]$ below can cover all the five rows in the extension matrix.

$$\begin{pmatrix} absent & slight & strip & * & normal \\ * & * & hole & fast & dry - peep \\ low & slight & strip & * & normal \\ absent & slight & spot & fast & dry - peep \\ low & medium & * & fast & normal \end{pmatrix}$$

2. The *precedence* strategy. When a r_{ij} in column j is the only nondead element of a row i in an extension matrix $EM_k = (r_{ij})_{n*a}$, the selector $[X_j \neq r_j]$ where $r_j = \vee_{i=1}^{a} r_{ij}$ is called an inevitable selector and thus is chosen with top precedence.

For example, $[X_1 \neq 1]$ and $[X_3 \neq 1]$ are two inevitable selectors in the extension matrix below which we have mentioned in Section 2.5.4.

$$\begin{pmatrix} 1 & * & * \\ * & 0 & 1 \\ 1 & 0 & * \\ * & 0 & 1 \\ 1 & 0 & * \\ * & * & 1 \end{pmatrix}$$

3. The *elimination* strategy. When each appearance of some nondead element in the j_1-th column of some row is always coupled with another nondead element in the j_2-th column of the same row in an extension matrix $EM_k = (r_{ij})_{n*a}$, $[X_{j_1} \neq r_{j_1}]$ where $r_{j_1} = \vee_{i=1}^{a} r_{ij1}$ is called an eliminable selector and thus eliminated by selector $[X_{j_2} \neq r_{j_2}]$ where $r_{j2} = \vee_{i=1}^{a} r_{ij_2}$.

   For example, attribute X_2 can be eliminated by attribute X_3 below.

$$\begin{pmatrix} 1 & * & * \\ * & 0 & 1 \\ 1 & 0 & 1 \\ * & 0 & 1 \\ 1 & 0 & 1 \\ * & * & 1 \end{pmatrix}$$

4. The *least-frequency* strategy. When all inevitable selectors have been chosen and all eliminable selectors have been excluded but all the selectors chosen have not yet covered all the rows in an extension matrix, exclude a least-frequency selector which has least nondead elements in its corresponding column in the extension matrix.

   For example, attribute X_1 in the following extension matrix can be eliminated and there still exists a path.

$$\begin{pmatrix} 1 & * & 1 \\ * & 0 & 1 \\ 1 & 0 & * \\ * & 0 & 1 \\ 1 & 0 & * \\ * & 0 & 1 \end{pmatrix}$$

Theorem 3.1. All of the *fast, precedence* and *elimination* strategies are complete, which means if there exists one or more shortest conjunctive formulae in an extension matrix, they will not lose it.

Proof. a) For an extension matrix $EM_k = (r_{ij})_{n*a}$, the largest and the possible least numbers of different selectors in a path are n and 1 respectively. When n selectors, whether the same or different, from n different rows but the same column j form a path, they can be integrated into one, $[X_j \neq \vee_{i=1}^{a} r_{ij}]$. So the selector found by the *fast* strategy must be an optimal formula of the extension matrix.

b) When r_{ij} in column j is the only nondead element of some row i in an extension matrix, the selector $[X_j \neq r_{ij}]$ needs to appear or to be integrated into a complex selector like $[X_j \neq \{..., r_{ij}, ...\}]$ in any path of the extension matrix. Picking up such kinds of selectors with top precedence will not violate the correctness of any path which is built but will speed up the construction process.

c) For any path $\{..., r_{j1}, ...\}$ in an extension matrix, we can simply replace r_{j_1} with r_{j_2} and $\{..., r_{j2}, ...\}$ is also a path in the same extension matrix when $r_{j_1} = \vee_{i=1}^{a} r_{ij_1}$, $r_{j_2} = \vee_{i=1}^{a} r_{ij_2}$, and each appearance of some nondead element in the j_1-th column of some row is always coupled with another nondead element in the j_2-th column of the same row. If there is another r_{j_3} in the path whose corresponding selector $[X_{j_3} \neq r_{j_3}]$ is also eliminable by $[X_{j_2} \neq r_{j_2}]$, we can simplify the path by replacing r_{j_1} and r_{j_3} with r_{j_2}. Therefore, the elimination strategy is complete for constructing optimal covers.

Theorem 3.2. If there exists at least one shorter path which has less than n different conjunctive selectors in an extension matrix, the solution generated by the *least-frequency* strategy must be the shorter one.

Proof. When all inevitable selectors have been chosen and all eliminable selectors have been excluded but all the selectors chosen have not yet covered all the rows in an extension matrix, there must exist at least one redundant selector in the extension matrix. For example, suppose there are k ($k \leq n$) rows in the extension matrix which have not been covered and all the nondead elements on those rows are $r_{i_1 j_{i_1 1}}, ..., r_{i_1 j_{i_1 j_1}}, ..., r_{i_k j_{i_k 1}}, ..., r_{i_k j_{i_k j_k}}$, as no more inevitable selector can be found at this moment, each row must contains at least two nondead elements. Therefore, crossing out any column can guarantee that each of those rows will still contain at least one nondead element, which means there is still at least one path left in the extension matrix. We can thus get the correctness proof of the fourth strategy above. As different selectors from the same column in an extension matrix can be integrated into one, excluding a column means there are at most $n-1$ selectors in the paths left. So the paths after this strategy must be shorter than the trivial ones.

Although the column with least nondead elements is not necessarily removed from all the optimal paths, the removal looks reasonable as choosing a column with fewer nondead elements means more columns thus more selectors may be involved in connecting a path. So the fourth strategy is a sensible heuristic. However, it is still heuristic. Firstly, this strategy is sensitive to the order of attributes in given examples. When we have two attributes with the same least nondead elements at some stage, different implementations of this strategy could produce different results. For the extension matrix below, excluding the second or the third attribute will produce different formulae ($X_3 \neq 1 \wedge X_4 \neq 0 \wedge X_5 \neq 0$ and $X_1 \neq 0 \wedge X_2 \neq 1$ respectively) to cover the same extension matrix, although the numbers of nondead elements in the second and third columns are the same: three. However, recall that the order of attributes also matters in the ID3 algorithm (see Section 3.2, footnote 3).

$$\begin{pmatrix} 0 & 1 & * & 0 & 0 \\ * & 1 & 1 & * & * \\ 0 & * & 1 & * & * \\ * & 1 & * & 0 & * \\ 0 & * & 1 & 0 & 0 \\ 0 & * & * & 0 & 0 \\ 0 & * & * & * & 0 \end{pmatrix}$$

Secondly, removing the least-frequency selector could also lose optimal paths. Taking the following extension matrix as an example, $X_1 \neq 0 \wedge X_3 \neq 0$ is its optimal formula. Removing the third column will cause the final formula to be $X_1 \neq 0 \wedge X_2 \neq 1 \wedge X_4 \neq 1$.

$$\begin{pmatrix} 0 & 1 & 0 & * \\ 0 & 1 & * & 1 \\ * & * & 0 & 1 \\ * & 1 & 0 & * \\ 0 & * & * & 1 \\ 0 & 1 & * & 1 \end{pmatrix}$$

These two problems also apply in a similar way to the first strategy of AE1 described in Section 2.5.4. However, as compared to AE1, we have adopted three complete strategies in HFL.

3.3.2 Algorithm description

In the HFL algorithm below, Function RESET1 is designed to find the selector $[X_j \neq r_j]$ where $r_j = \vee_{i=1}^{a} r_{ij}$ when the j-th column has been chosen by either the *fast* strategy or the *precedence* strategy, Function RESET2 is designed to cross out the nondead elements on uncovered rows in column j when the j-th column has been found eliminable by either the *elimination* strategy or the *least-frequency* strategy, and words between /* and */ are explanatory notes. An EM in the algorithm can be either an extension matrix (EM_k) or a disjunction matrix (EMD) in which there is at least one path.

```
Procedure HFL(EM; Hfl)
   integer n, a
   matrix EM(n, a), D(a), CH(a), C(n)
   set Hfl
   S0:  D←0 /* D marks the elimination status of each column. */
        CH←0 /* CH marks the chosing status of each column. */
        C←0 /* C marks the covering status of each row. */
        Flag←'F' /* Flag='T' indicates that EM has been
            fully covered. */
        Hfl← φ /* initialisation */
   S1:  /* the fast strategy */
         i←1
         while i≤a & Flag='F' do
             { j←1, Flag1←'F'
               /* During the test of each column i in EM, Flag1
              ='T' means some * element in the column has been
              found and thus the fast strategy fails there. */
               while j≤n & Flag1='F' do
                   { if C(j)=1 then goto L1
                       if EM(i,j)=* then Flag1←'T'
                       L1:  j←j+1
                   }
```

```
            if j=n+1 & Flag1='F' then
                { RESET1(i,Set)
                  Hfl←Hfl∧[X_i≠Set]
                }
          }
S2:  /* the precedence strategy */
      if Flag='F' then
        { Flag2←'F'
        /* Flag2='T' indicates that at least one inevitable
       selector has been chosen during this step. */
          for i=1 to n do
              if C(i)=0 and only EM(i,j) is a nondead
                    element on the i-th row
                  then { RESET1(j,Set), Flag2←'T'
                          Hfl←Hfl∧ [X_j ≠Set] }
          next i
        }
      if Flag2='T' then goto S1
S3:  /* the elimination strategy */
      if Flag='F' then
        { Flag3←'F'
        /* Flag3='T' indicates that at least one eliminable
       selector has been excluded during this step. */
          for i=1 to a do
```

```
if CH(i)=0 & D(i)=0 then
    { j←1, Flag32←'F'
      /* Flag32='T' means column i has been
    eliminated. */
      while j≤a & Flag32='F' do
        if CH(j)=0 & D(j)=0 & i≠j then
            { k=1, Flag33←'F'
              /* Flag33='T' means column i can
            not be eliminated by column j with
            the elimination strategy. */
                while k≤n & Flag33='F' do
                  if EM(i,k)≠ * & EM(j,k)=*
                        then Flag33←'T'
                        else k←k+1
                if k=n+1 & Flag33='F'
                    then { D(i)←1,RESET2(i)
                           Flag32←'T'
                           Flag3←'T' }
            }
            else j←j+1
    }
L2:  next i
}
if Flag3='T' then goto S2
```

```
S4:  /* the least-frequency strategy */

      if Flag='F' then

          { for i=1 to a do

                t(i)←0

                /* t(i) counts the nondead elements on
               uncovered rows in column i. */

                if CH(i)=0 & D(i)=0 then

                      for j=1 to n do

                          if C(j)=0 & EM(j,i)≠*

                              then t(i)←t(i)+1

                      next j

              next i

              i←1

              while i≤a & t(i)=0 do i←i+1

              for j=i+1 to a do

                  if t(j)<t(i) & t(j)≠0 then i←j

              next j

              D(i)←1,RESET2(i)

              goto S2

            }

Function  RESET1(j,Set)

  Set← φ

  for i=1 to n do

      if C(i)=0 & EM(i,j)≠*
```

```
        then { Set←Set∪{EM(i,j)}
                  EM(i,j)←*, C(i)←1 }
  next i
  CH(j)←1
  if all of C(k)=1 (k=1,...,n) then Flag←'T'
  /* This test is for those cases where after one selector
   has been chosen, EM has been fully covered. */
Return(Set)
Function  RESET2(j)
  for i=1 to n do
      if EM(i,j)≠* then EM(i,j)←*
  next i
Return
Return(Hfl)
```

Steps S1, S2, S3 and S4 implement the *fast, precedence, elimination* and *least-frequency* strategies introduced in Section 3.3.1. Once the *fast* strategy finds a column which has nondead elements on all the uncovered rows in an EM, the EM can be fully covered and thus the Hfl is ready. After one or more inevitable selectors have been chosen in Step S2, HFL will come back to Step S1 to test the *fast* strategy on uncovered rows. Every time a selector has been chosen by either the *fast* strategy or the *precedence* strategy, there are two possible cases: all the selectors which have been chosen up till now either have or have not yet covered all the rows in the EM. Those two cases are tested in Function RESET1. After one or more columns have been crossed out by the *elimination* strategy in Step S3, the *precedence* strategy and the *fast* strategy will be tested again. Only in those cases when all inevitable selectors have been chosen and all eliminable selectors have been excluded but all the selectors chosen have not yet covered all the rows in an extension matrix, the *least-frequency* strategy is used and it can always cross out a column which has not been crossed out before. After Step S4, HFL comes back to test the *precedence* strategy. Since excluding a column by either the *elimination* strategy in Step S3 or the *least-frequency* strategy in Step 4 does not cover any uncovered rows, the *fast* strategy cannot be applicable immediately after these two strategies. This is why the control in HFL comes back to Step 2 instead of Step 1 at the end of both Step 3 and Step 4. Each time the control comes back to Step S2 or Step S1, there is at least one column or selector has been processed, either chosen or crossed out. There are a columns in an EM in total, therefore at most a loops in HFL are needed.

Step S0 requires $n+2a+2$ operations. The time complexity for Steps S1, S2, S3 and S4 is $O(na)$, $O(na)$, $O(na^2)$ and $O(na)$ respectively. In the worst case, a loops are needed among Steps S1, S2, S3 and S4 to complete the assignment of $Flag \leftarrow$ 'T'. The time complexity for the whole algorithm is thus

$$O(n+2a+2+a(na+na+na^2+na)) \approx O(na^3).$$

When the EM in the HFL algorithm is the extension matrix EM_k of a positive example $e_k^+ = (v_{1_k}^+, ..., v_{a_k}^+)$ against NE, a selector $[X_j \neq r_j]$ in the Hfl is equivalent to $[X_j = v_{j_k}^+]$ with existing examples in a given example set. Meanwhile, if the EM is the disjunction matrix EMD of an intersecting group of positive examples $e_{i_1}^+, ..., e_{i_k}^+$ against NE, a selector $[X_j \neq r_j]$ is equivalent to $[X_j \in \vee_{k_2=1}^{k} v_{j_{i_{k_2}}}^+]$ in the context of existing examples. When X_j is a numerical attribute, $\vee_{k_2=1}^{k} v_{j_{i_{k_2}}}^+$ can be further grouped into a number of intervals none of which will contain any NEM_{ij} $(i=1,...,n)$ (see Section 3.5.5).

From Theorems 3.1 and 3.2, we can easily get the theorem below (Theorem 3.3).

Theorem 3.3. If there exists at least one path in an EM, the HFL algorithm can produce a conjunctive formula which corresponds to a path in the EM. The number of selectors in Hfl produced by HFL is always smaller than n so long as there is at least one path with less than n different elements in the EM.

3.3.3 An example run of HFL

For the example set given in Table 3-1, there are five positive examples for *Pneumonia*. $PE = \{e_1^+, e_2^+, e_3^+, e_4^+, e_5^+\}$, $NE = \{e_6^-, e_7^-, e_8^-, e_9^-, e_{10}^-\}$, and

$$NEM = \begin{pmatrix} absent & slight & strip & normal & normal \\ high & heavy & hole & fast & dry-peep \\ low & slight & strip & normal & normal \\ absent & slight & spot & fast & dry-peep \\ low & medium & flack & fast & normal \end{pmatrix}.$$

The disjunction matrix EMD of $\{e_1^+, e_2^+, e_3^+, e_4^+, e_5^+\}$ against NE is

$$EMD = \begin{pmatrix} absent & * & strip & * & normal \\ * & * & hole & fast & * \\ * & * & strip & * & normal \\ absent & * & * & fast & * \\ * & * & * & fast & normal \end{pmatrix}$$

by Definition 2.5.3.

Running HFL on EMD, attribute $FEVER$ is eliminated by the *least-frequency* strategy during the first loop of Steps S1, S2, S3 and S4 and EMD becomes

$$\begin{pmatrix} * & * & strip & * & normal \\ * & * & hole & fast & * \\ * & * & strip & * & normal \\ * & * & * & fast & * \\ * & * & * & fast & normal \end{pmatrix}.$$

In the second loop, $[ESR \neq fast]$ which is equivalent to $[ESR = normal]$ is chosen as an inevitable selector on the fourth row by the *precedence* strategy and it covers rows 2, 4 and 5. EMD now becomes

$$\begin{pmatrix} * & * & strip & * & normal \\ * & * & * & * & * \\ * & * & strip & * & normal \\ * & * & * & * & * \\ * & * & * & * & * \end{pmatrix}.$$

In the third loop, attribute $X{-}ray$ is eliminated by attribute $AUSCULTATION$ by the *elimination* strategy and EMD becomes

$$\begin{pmatrix} * & * & * & * & normal \\ * & * & * & * & * \\ * & * & * & * & normal \\ * & * & * & * & * \\ * & * & * & * & * \end{pmatrix}.$$

Selector $[AUSCULTATION \neq normal]$ which is equivalent to $[AUSCULTATION \in \{bubble - like, dry - peep\}]$ is finally chosen by the *fast* strategy to cover the remaining rows 1 and 3.

Therefore, the Hfl for EMD generated by the HFL algorithm is

$$[ESR = normal] \wedge [AUSCULTATION \in \{bubble - like, dry - peep\}]$$

which covers all the five examples of *Pneumonia* against examples of *Tuberculosis*.

3.4 The HCV Algorithm

3.4.1 Algorithm description

The basic idea for the HCV algorithm is to partition PE of a specific class into p' ($p' \leq p$) intersecting groups first; call the heuristic Algorithm HFL to find a Hfl for each intersecting group; then give the covering formula by logically ORing all the Hfl's finally.

The GEM algorithm is designed to generate the disjunction matrix EMD of $e^+_{i_1}, ..., e^+_{i_k}$ against NE from $EM_{i_1}, ..., EM_{i_k}$ according to Definition 2.5.3. When there exists a dead element on the (i, j)-position of any of $EM_{i_1}, ..., EM_{i_k}$, $EMD(i, j) = *$. Otherwise, $EMD(i, j) = NEM(i, j) = EM_{i_{k_2}}$ ($k_2 \in \{1, ..., k\}$).

```
Procedure GEM({EM_{i_1}, ..., EM_{i_k}}; EMD)
  integer n, a, k
  matrix EM_{i_1}(n,a), ..., EM_{i_k}(n,a), EMD(n,a)
  for j_1=1 to n do
      for j_2=1 to a do
          if ∃k_2∈{1, ..., k}:  EM_{i_{k_2}}(j_1, j_2)=*
              then EM(j_1, j_2)←*
              else EM(j_1, j_2)←EM_{i_1}(j_1, j_2)
      next j_2
  next j_1
Return(EM)
```

The time complexity for Algorithm GEM is $O(nak)$ with k being the number of positive examples.

Algorithm IDEN below is designed to test whether there is a path in a disjunction matrix EMD with the result being returned by logical variable $Flag$. It tests each row of EMD to ascertain whether there is at least one nondead element on the row. If each row has at least one nondead element, then there exists at least one path in EMD and thus $Flag$ is assigned to 'T'.

```
Procedure IDEN(EMD; Flag)
  integer n, a
  matrix EMD(n, a)
  logical Flag
  i←1, Flag2←'F'
  while i≤n & Flag2='F' do
    { j←1, Flag3←'F'
      while j≤a & Flag3='F' do
          if EMD(i, j)≠ * then Flag3←'T'
              else j←j+1
      if Flag3←'F' then Flag2←'T'
          else i←i+1
    }
  if Flag2='F' then Flag←'T'
      else Flag←'F'
Return(Flag)
```

In the worst case, Algorithm IDEN needs to test each element in EMD. The time complexity for Algorithm IDEN is $O(na)$.

Based on the GEM and IDEN algorithms above and the HFL algorithm in Section 3.3, the HCV algorithm is designed as follows where GEM and IDEN are used to partition PE into intersecting groups and HFL is used to find a conjunctive formula for each intersecting group.

```
Procedure HCV(EM_1, ..., EM_p; Hcv)
   integer n, a, p
   matrix EM_1(n,a), ..., EM_p(n,a), D(p)
   set Hcv
   S1:  D←0 /* D(j)=1 (j=1, ..., p) indicates that
                 EM_j has been put into an intersecting group. */
        Hcv←φ /* initialisation */
   S2:  for i=1 to p do
              if D(i)=0 then
                 { EM←EM_i
                   for j=i+1 to p do
                        if D(j)=0 then
                           { call GEM({EM, EM_j}; EM2)
                             call IDEN(EM2; Flag)
                             if Flag='T' then
                               { EM←EM2, D(j)←1 }
                           }
                   next j
                   call HFL(EM; Hfl)
                   Hcv ←Hcv∨Hfl
                 }
        next i
 Return(Hcv)
```

Step S1 in the algorithm above requires $p+1$ operations. The worst case operation for Step S2 is

$$O(\sum_{i=1}^{p}(na+\sum_{j=i+1}^{p}(2na+na+na+1)+(na^3)+1))$$
$$\approx O(pna^3+p^2na).$$

So the time complexity for Algorithm HCV is $O(pna^3+p^2na)$.

Theorem 3.4. The formula Hcv generated by Algorithm HCV covers all the positive examples against negative examples in a given example set.

Proof. Each Hfl in the Hcv produced by Algorithm HCV covers a group of positive examples against NE. So no negative example in NE will be covered by any Hfl. Neither will the Hcv cover any of the negative examples in NE (because it is an OR combination of all the Hfls). As all positive examples have been included in the intersecting groups in Step S2, each positive example is covered by a Hfl in the Hcv.

Algorithm HCV is a bidirectional algorithm. It first groups the positive example set in a top–down way and then calls algorithm HFL, which works in a bottom–up way. Its time is low-order polynomial as opposed to exponential in the first strategy mentioned in the introduction of this chapter. From Theorems 3.1 and 3.2 and the Corollary in Section 2.5.2, both Algorithm HFL and Algorithm HCV usually produce shorter formulae than the trivial strategy (the second strategy) also mentioned in the introduction of this chapter so long as the shorter formulae exist.

Theorem 3.5. If there exists at least one conjunctive cover in a given training example set, the formula produced by HCV must be a conjunctive one.

Proof. If there exists at least one conjunctive cover in a given training example set, there must exist at least one path in the disjunction matrix of all the positive examples against the negative examples in the given example set according to Theorem 2.5.1. Therefore, all the positive examples will be put into an intersecting group in Step 2 of HCV and a conjunctive Hfl will be produced by calling HFL as the solution.

However, the intersecting groups partitioned by HCV and therefore the results returned by the HFL algorithm (described in Section 3.3) on each intersecting group or partition are sensitive to the order of examples in a given example set. For a given partition, the order of positive examples does not affect their disjunction matrix and therefore does not change the result of HFL. When changing the order of two negative examples, just two rows are swaped in the disjunction matrix of the positive examples in the partition. As all the four strategies in HFL only relate to the number of nondead elements in each column, the order of negative examples cannot affect the results of HFL or, therefore of HCV. So, only when the order of positive examples changes can the result produced by HCV become different. We will demonstrate this in Section 3.4.2.

3.4.2 Two example runs of HCV

Example 3-2. Table 3-2 (which is the same as Table A-1 in Appendix A) shows a set of training examples for deciding whether to play golf on a Saturday afternoon.

Considering PE (of *Play*) and NE (of *Don't Play*) in Table 3-2, let us observe the results generated by the HCV algorithm.

For the given example set, $NE = \{e_1^-, e_2^-, e_7^-, e_8^-, e_{12}^-, e_{13}^-, e_{14}^-\}$, $PE = \{e_3^+, e_4^+, e_5^+, e_6^+, e_9^+, e_{10}^+, e_{11}^+\}$ and

$$NEM = \begin{pmatrix} rain & hot & high & true \\ rain & cool & normal & true \\ sunny & hot & normal & true \\ sunny & mild & high & true \\ sunny & hot & high & false \\ sunny & cool & normal & false \\ rain & mild & normal & true \end{pmatrix}.$$

The first intersecting group found in Step S2 by starting with the first positive example (e_3^+) and calling the GEM and IDEN algorithms is $\{e_3^+, e_4^+, e_6^+\}$ and the

Table 3–2: Cases of *Play* and *Don't Play* (adapted from [Quinlan 86b])

ORDER	*OUTLOOK*	*TEMPERATURE*	*HUMIDITY*	*WINDY*	**DECISION**
1	rain	hot	high	true	*Don't Play*
2	rain	cool	normal	true	*Don't Play*
3	overcast	mild	high	true	*Play*
4	overcast	mild	normal	false	*Play*
5	rain	hot	high	false	*Play*
6	overcast	cool	normal	true	*Play*
7	sunny	hot	normal	true	*Don't Play*
8	sunny	mild	high	true	*Don't Play*
9	sunny	mild	normal	false	*Play*
10	rain	cool	normal	false	*Play*
11	rain	hot	high	false	*Play*
12	sunny	hot	high	false	*Don't Play*
13	sunny	cool	normal	false	*Don't Play*
14	rain	mild	normal	true	*Don't Play*

disjunction matrix EMD_1 against NE is

$$EMD_1 = \begin{pmatrix} rain & hot & * & * \\ rain & * & * & * \\ sunny & hot & * & * \\ sunny & * & * & * \\ sunny & hot & * & * \\ sunny & * & * & * \\ rain & * & * & * \end{pmatrix}.$$

Calling HFL, $[OUTLOOK \neq \{rain, sunny\}]$ which is equivalent to $[OUTLOOK = overcast]$ is chosen by the *fast* strategy and the first Hfl is thus

$$[OUTLOOK = overcast].$$

The second intersecting group found in Step S2 by starting with the third positive example (e_5^+) and calling the GEM and IDEN algorithms is $\{e_5^+, e_{10}^+, e_{11}^+\}$ and the disjunction matrix EMD_2 is

$$EMD_2 = \begin{pmatrix} * & * & * & true \\ * & * & * & true \\ sunny & * & * & true \\ sunny & mild & * & true \\ sunny & * & * & * \\ sunny & * & * & * \\ * & mild & * & true \end{pmatrix}.$$

Running HFL, $[WINDY \neq true]$ and $[OUTLOOK \neq sunny]$ which are equivalent to $[WINDY = false]$ and $[OUTLOOK = rain]$ respectively are both chosen as inevitable selectors and they cover all of the five rows in EMD_2. Therefore, the second Hfl is

$$[WINDY = false] \wedge [OUTLOOK = rain].$$

The third intersecting group is $\{e_9^+\}$ and the disjunction matrix EMD_3 is

$$EMD_3 = \begin{pmatrix} rain & hot & high & true \\ rain & cool & * & true \\ * & hot & * & true \\ * & * & high & true \\ * & hot & high & * \\ * & cool & * & * \\ rain & * & * & true \end{pmatrix}.$$

Running HFL, $[TEMPERATURE \neq \{hot, cool\}]$ is first chosen as an inevitable selector and it covers rows 1, 2, 3, 5 and 6, attributes $OUTLOOK$ and $HUMIDITY$ are then excluded by attribute $WINDY$ and $[WINDY \neq true]$ is finally chosen as an inevitable selector on the fourth row after $OUTLOOK$ and $HUMIDITY$ have been crossed out. The equivalent Hfl for this intersecting group is

$$[TEMPERATURE = mild] \wedge [WINDY = false].$$

Therefore,

$$Hcv = [OUTLOOK = overcast] \quad \vee$$

$$[WINDY = false] \wedge [OUTLOOK = rain] \quad \vee$$

$$[TEMPERATURE = mild] \wedge [WINDY = false]$$

whose equivalent rule in variable-valued logic is:

[*OUTLOOK=overcast*]

$\vee$

[*WINDY=false*]

[*OUTLOOK=rain*]

$\vee$

[*TEMPERATURE=mild*]

[*WINDY=false*]

$\rightarrow$

[*DECISION=Play*] .

Meanwhile, the decision tree generated by ID3 and its equivalent rules are given in Appendix A.

For the example set shown in Table 3-2, there are 14!=87,178,290,000 possible orders for the 14 examples. When we consider the intersecting groups of positive examples of *Play* against examples of *Don't Play*, the order of negative examples does not change the partitioning of positive examples. So there are 7!=5040 possible partitionings for the 7 positive examples. Theoretically, once the order of positive examples changes, the partitions and therefore the result produced by HCV could be different. However, over 500 different orders of the 7 positive examples have been tested and 2 different partitionings have been found, but the results produced by HCV are the same.

Example 3-3. Table 3-3 shows two different orders of the same eight examples in an artificial example set, with which HCV produces different results.

The results produced by HCV for the T class are given below:

For Table 3-3 (a):

[B=[1]]
[A=[0]]
$\vee$
[A=[1]]
[C=[1]]
$\rightarrow$
The T class.

For Table 3-3 (b):

[B=[1]]
[C=[1]]
$\vee$
[A=[1]]
[C=[1]]
$\vee$
[A=[0]]
[C=[0]]
$\rightarrow$
The F class.

However, the order of examples does not effect the decision trees generated by ID3. Figure 3-2 shows the decision tree generated by ID3 for both Table 3-3 (a) and (b). The decision-tree-traversal rules are as follows:

B=1 & A=0
$\vee$ B=1 & A=1 & C=1
$\vee$ B=0 & A=1 & C=1
$\rightarrow$
The T class.

B=0 & A=0
$\vee$ B=0 & A=1 & C=o
$\vee$ B=1 & A=1 & C=0
$\rightarrow$
The F class.

3.4.3 A comparison between HCV and AE1

As we mentioned in Section 2.5.4, the extension matrix approach was first introduced in AE1 in 1985. To the best of the author's knowledge, the approach itself has not yet been improved at all in anywhere since then except in the author's HCV algorithm. The developer of AE1 has recently developed an AE5 system [Hong 89b] based on AE1 but the basic algorithm remains to be the same. The only difference between AE5 and AE1 is that some facilities such as constructive and incremental induction have been added to the AE5 system. So, the two major problems of AE1 described in Section 2.5.4 also apply to AE5.

Table 3–3: Cases in different orders

(a)

ORDER	*A*	*B*	*C*	*CLASS*
1	0	0	1	*F*
2	0	1	0	*T*
3	0	1	1	*T*
4	1	0	0	*F*
5	1	0	1	*T*
6	1	1	0	*F*
7	1	1	1	*T*

(b)

ORDER	*A*	*B*	*C*	*CLASS*
1	0	0	1	*F*
2	1	1	1	*T*
3	0	1	1	*T*
4	1	0	0	*F*
5	1	0	1	*T*
6	1	1	0	*F*
7	0	1	0	*T*

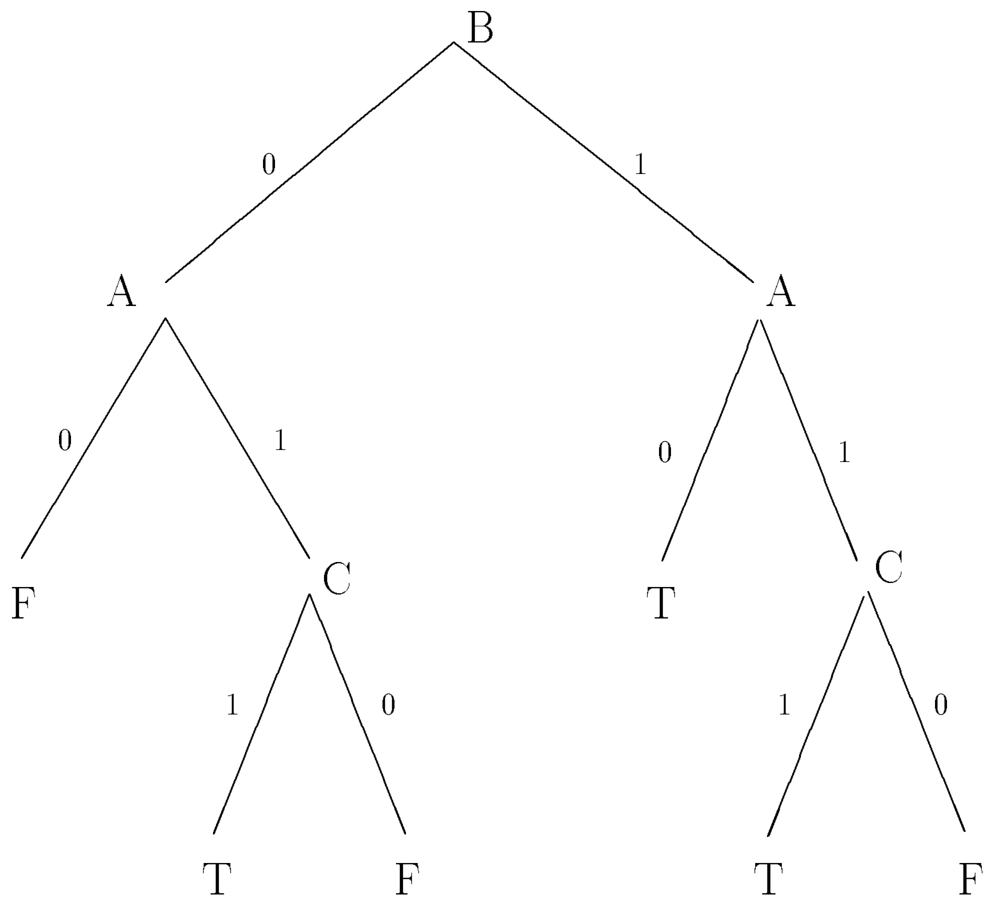

Figure 3–2: A decision tree (by ID3) for Table 3-3

Although HCV is based on the extension matrix approach developed in AE1, there are two radical improvements on the approach itself in HCV as well as those implementation aspects described in Section 3.5.

- The use of disjunction matrixes.

Although disjunction matrixes are also defined in [Hong 85], AE1 does not produce and use them in the partitioning of positive examples. AE1 still needs to produce all extension matrixes of positive examples against negative examples (by remembering only dead elements in each extension matrix [Hong 89b]), while in HCV, as we will see in Section 3.5.6, we need at most two extension matrixes at each time stage. By using disjunction matrixes, HCV can both save a lot of data space and provide a natural way to reduce its rules' complexity. As we have seen from Theorem 3.5, if there exists at least one conjunctive cover in a given training example set, the formula produced by HCV must be a conjunctive one. This is by no means guaranteed in AE1.

- Three complete strategies in HFL.

HCV has provided a reasonable solution to both the MFL and the MCV problems described in Section 2.5, which are NP-hard in nature. So the second disadvantage of AE1 has disappeared in HCV. Although the first disadvantage of AE1 still exists in HCV's *least-frequency* strategy, we have provided three complete strategies at the same time. For those example sets where the three strategies are enough to produce the final results, we can guarantee that the results are optimal. According to all the experiments the author has carried out including those mentioned in this thesis, the three strategies are always useful even when they are not enough to produce an optimal result.

3.5 Some Related Problems in Implementation

3.5.1 Introduction

The HCV algorithm has been implemented on both PC machines (see Chapter 5) and Sun workstations. The term "HCV (Version 1.0)" in the following account indicates the current implementation (Version 1.0, [Wu 92d]) of the HCV algorithm in SICStus Prolog which runs on SUN3 and SPARC workstations. In this implementation, HCV (Version 1.0) can classify more than two classes of examples and produce rules for each class of pre-classified examples by using part of the AQ11 technique outlined in Section 2.2 (*i.e.* assuming that examples not classified as positive are negative). For the example set in Appendix B, the rules produced by HCV (Version 1.0) are shown below:

[X2=[b]]
$\vee$
[X1=[0]]
[X2=[a]]
$\vee$
[X1=[0]]
[X4=[0]]
$\rightarrow$
The T class.

[X2=[c,a]]
[X1=[1]]
$\vee$
[X2=[c]]
[X4=[1]]
$\rightarrow$
The F class.

The HCV (Version 1.0) program also allows the user to evaluate the rules' accuracy in terms of a set of pre-classified *test* examples (see Chapter 4).

3.5.2 *Don't Cares* in HCV

The **#** symbol in HCV (Version 1.0), like in many other induction programs, has a specific meaning when representing attribute values: *Don't Cares*. For instance, the example set in Appendix B can be equivalently expressed as Table 3-4 in HCV (Version 1.0).

Table 3–4: Cases of T and F

ORDER	*X1*	*X2*	*X3*	*X4*	**CLASS**
1	1	a	#	1	F
2	1	a	a	0	F
3	1	b	c	1	T
4	0	b	b	0	T
5	0	a	c	1	T
6	1	b	a	#	T
7	1	c	c	0	F
8	1	c	b	1	F
9	0	c	b	0	T
10	0	a	a	0	T
11	0	c	c	1	F
12	0	c	a	0	T
13	1	a	b	0	F
14	0	a	a	1	T
15	0	b	a	1	T

A *Don't Care* value of an attribute in an example is used to indicate that the value of the attribute is irrelevant to the classification of the example. An example with *Don't Care* values can always be converted into a number of equivalent examples which have no *Don't Care*s. Therefore, a *Don't Care* attribute is universally quantified and *Don't Care* values can in practice be used to compress data spaces.

In the extension matrix approach, a dead element ($*$) in an extension matrix indicates that a positive example and a negative example have the same attribute value and therefore the attribute value cannot be used to distinguish the positive example from the negative example. When a negative example (say the i-th) has a *Don't Care* value on an attribute (say the j-th), the r_{ij} in every positive example's extension matrix must be $*$ because the negative example can take every positive example's j-th attribute value. When a positive example has a *Don't Care* value on its j-th attribute, all the values on the j-th column of the positive example's extension matrix must be $*$ according to the meanings of *Don't Care* values and $*$.

3.5.3 Noise handling in HCV

Strictly speaking, the HCV algorithm as it stands does not process noisy data. *Unknown*s, which are different from the *Don't Care* values described above and usually used to represent uncertainty or missing attribute values, are not permitted in either the HCV algorithm or the HCV (Version 1.0) program. A `?` value is just a common value like `a, 'X0.5'` or `0.24` in HCV (Version 1.0). It has no specific meaning here. However, it is not difficult to add a screening engine to remove noise from an example set before induction takes place.

When an example turns up in more than one class in an example set, we say that there is a contradiction in the example set. The HCV (Version 1.0) program checks for contradictions when a training example set is input. Once a contradiction is found, the corresponding example will be omitted from both induction and test. For instance, even if we put two more examples as below in Table 3-4, the results produced by HCV (Version 1.0) will not change at all.

ORDER	*X1*	*X2*	*X3*	*X4*	**CLASS**
16	0	c	a	1	F
17	0	c	a	1	T

When an example turns up more than once in the same class in an example set, we say that there is a repetition in the example set. Repetitions are silently processed in HCV (Version 1.0): each example is counted in only once during induction even if it appears many times.

3.5.4 The default rule

In the HCV (Version 1.0) program, we have adopted the specific engine in CN2 [Clark *et al.* 89] for defining default rules which have no reference to attribute values. The most commonly occurring class in a training example set is assigned to all those new examples which cannot be classified by the rules produced by the HCV algorithm. Examples can be found in Appendix C.

3.5.5 Numerical attributes

Basically, numerical attributes in the HCV algorithm are processed in the same way as symbolic ones during induction. When generating selectors, HCV (Version 1.0) adopts the following procedure to group values of each numerical attribute (say X_m) once selected by the *fast* or *precedence* strategy into a number of intervals:

Step 1: Collect all values, $Vlist$, of the numerical attribute which the current intersecting group of positive examples take.

Step 2: Sort $Vlist$ into an ascendant order.

Step 3: Partition $Vlist$ into a number of intervals, $[V_{1_1}, V_{1_{i_1}}]$, $[V_{2_1}, V_{2_{i_2}}]$, ..., $[V_{k_1}, V_{1_{i_k}}]$, such that

1. there is not a V such that $V_{j_1} < V < V_{j_{i_j}}$ $(j = \{1, ..., k\})$ and V appeared on X_m of any negative examples;

2. there must be some V' that $V_{j_{i_j}} < V' < V_{(j+1)_1}$ for each $j \in \{1, ..., k-1\}$ and V' appeared on X_m of some negative example(s).

Step 4: Write the selector as $[X_m = [[V_{1_1}, V_{1_{i_1}}], [V_{2_1}, V_{2_{i_2}}], ..., [V_{k_1}, V_{1_{i_k}}]]$ where "=" indicates membership ($\in$).

3.5.6 Size of extension matrixes

In each extension matrix or disjunction matrix, all nondead elements are the same as those in the negative example matrix NEM according to Definitions 2.5.1 and 2.5.3. We only need to remember the dead elements in execution. The number of dead elements in the extension matrix or disjunction matrix must be less than na where n and a are the numbers of negative examples and attributes in a given example set.

All the positive examples are processed one by one in the HCV algorithm. At each stage, we need at most an extension matrix and a disjunction matrix. The space needed for an implementation of the HCV algorithm is less than $2na$. Therefore, the HCV algorithm is low-order polynomial learnable in both time and space. This is an important feature for good learning algorithms [Valiant 84].

3.6 A Comparison with ID3 and HCV

One difference between HCV and ID3 is that the HCV algorithm only produces rules for positive examples while ID3 generates decision trees to classify both positive and negative examples. However, this is not an advantage of ID3 over HCV. For instance, if all the examples in an example set are people from different countries in the world, when we are told that some of them are British and the task is to find characteristics of British, we will only be interested in the description produced for British because all other examples which cannot be satisfied by the description will automatically belong to other countries. Although the ID3 algorithm will automatically produce a description for negative examples at the same time as it produces the description for positive examples, we do not think that is useful in many cases. A description for negative examples belonging to all other countries except Britain will not help anything because 1) it can be inferred from the description for positive examples, and 2) if we want to know which specific country a negative example belongs to, we need to run ID3 once again. The HCV (Version 1.0) program mentioned in the last section has already been able to produce rules to classify more than two classes of examples. The entropy measure in ID3 can also be easily extended [Clark 90] to chunk examples into more than two classes.

The following is a comparison between ID3 and HCV.

The reason for using decision trees rather than rules, such as the variable-valued logic rules adopted in AQ11 and HCV, is said by [Jackson 90] to be that the ID3-like algorithms are comparatively simpler than other learning algorithms. From the fourth disadvantage of ID3 (see Section 2.3.4) and the time complexity of HCV, we can say that the argument is now no longer convincing. Although the information theoretic heuristic is by no means complete, ID3 needs to examine all possible candidate attributes and their values to choose one attribute at each non-leaf node of its decision trees and thus its time complexity is still expensive [Utgoff 89]. In HCV, although all of the *fast, precedence* and *elimination* strategies are complete (see Theorem 3.1), which means if there exists one or more shortest conjunctive formulae in an extension matrix they will not lose it, the *fast* strategy can choose an optimal attribute as soon as it finds the attribute without any attention to other attributes and the *precedence* strategy can choose an inevitable attribute by examining only the values of one row in an extension matrix. High efficiency has been seen as an important requirement for knowledge discovery and exponential or even medium-order polynomial complexity will not be of practical use [Quinlan 89b] in realistic data bases. We have not provided the comparison of HCV and ID3 on time performance because there are different results for ID3's time complexity and it is difficult to say which is correct or wrong. Therefore, we can not say in general that HCV outperforms ID3 in time. However, we have given detailed analysis on HCV's time complexity and have shown that it is low-order polynomial and therefore computationally acceptable. The first significant advantage of the HCV algorithm is that it supports a reasonable solution to the NP-hard problem in the extension matrix approach for inductive learning.

Contrasting to the second and third disadvantages of ID3, different values of the same attribute which take on only positive examples can be easily grouped into a selector in the variable-valued logic. In ID3, once an attribute is selected, all arcs labeled by values that attribute takes must be expanded. This can still make the number of branches (paths) large since at each arc only one value can be labeled, and resulting paths might be longer than those actually needed because, by the time specific concepts (leaves on the decision tree) are developed, irrelevant variables may have been introduced. All of the four strategies adopted in Algorithm HFL and the partitioning technique in HCV are designed to reduce the number of selectors. From Theorem 3.1, for those problems where the *fast, precedence* and *elimination* strategies are enough to produce their final formulae, we can guarantee that the formulae are optimal. From Theorem 3.5, if there exists at least one conjunctive cover in a given training example set for positive examples against negative examples, the formula produced by HCV must be a conjunctive one. However, the information theoretic heuristic in ID3 is not complete, which means it is not guaranteed to find the simplest decision tree that characterizes the given training instances. From the example sets given above, the rules produced by HCV are all more compact [4] in terms of the numbers of conjunctive rules and conjunctions than the decision trees or their equivalent decision rules produced by ID3. So, the compactness of rules in HCV is its second advantage. However, the *least-frequency* strategy is still heuristic. We cannot guarantee the rules produced by HCV must be more compact than the decision trees generated by ID3 in all possible cases. There are still three kinds of possible results for a new example

[4] This is still true when we (1) only count the rules for positive examples and (2) count a membership like $X_j \in [V_1, ..., V_n]$ (or $X_j = [V_1, ..., V_n]$) in HCV rules as n non-membership conjunctions.

set: 1) HCV produces more compact rules as analysed and shown above; 2) HCV and ID3 produce similar rules because ID3 can usually produce efficient decision trees, and 3) ID3 produces more compact rules than HCV when ID3 can produce the shortest decision tree while HCV cannot generate optimal rules. For instance, the order of cases in a given example set can effect the result of HCV but does not change the decision tree generated by ID3. Sometimes, we could possibly change the order of examples to make ID3 outperform HCV.

Also, all of the four strategies adopted in Algorithm HFL and the partitioning technique in HCV are more comprehensible than the information theoretic heuristic for most human experts who are not familiar with information theory.

3.7 Conclusions

The HCV algorithm described in this chapter can be viewed as a representative of the extension matrix approach based family of inductive algorithms. As its time is low-order polynomial, it can be seen as one of the fastest learning algorithms to date. From the analysis of the strategies adopted in HCV and the example sets shown above, the description in variable-valued logic generated in HCV is similar to that adopted in AQ11, which is the advantage of AQ11 over ID3. Also, the rules generated by HCV have been shown empirically to be fairly compact although we cannot yet generally say they are necessarily more compact than the decision trees produced by the ID3-like algorithms. For those problems where the *fast, precedence* and *elimination* strategies are enough to produce their final formulae, we can guarantee that the formulae are optimal but the information theoretic heuristic in ID3 cannot.

Chapter 4

The MONK's Problems: A Performance Comparison of HCV with Other Induction Algorithms

4.1 Introduction

This chapter provides a performance comparison of HCV with other algorithms, such as ID3, ID5R, ASSISTANT, AQR (which can be viewed as an implementation of the AQ11 algorithm to chunk examples into only two classes) and CN2, in terms of rule compactness and accuracy on the three *MONK's problems*. All these algorithms have been mentioned in Sections 2.2 to 2.4. The MONK's problems are concerned with learning concept descriptions from examples.

The data for the three MONK's problems are described in Chapter 1 of [Thrun *et al.* 91]. However, as some examples were missing from the training sets in [Thrun *et al.* 91], I have consulted John Cheng in Carnegie Mellon University and completed the data sets myself. The results from applying the HCV (Version 1.0) program to the three problems are recorded in Appendix C, whereas the results generated by other algorithms are adopted from [Thrun *et al.* 91]. Those results were given by a collection of researchers, each of whom was an advocate or the creator of the algorithm they tested.

4.2 The MONK's Problems

The MONK's problems are derived from an artificial robot domain, in which robots (examples) are described by six multiple-valued attributes, i.e., *head_shape*, *body_shape*, *is_smiling*, *holding*, *jacket_color*, and *has_tie*. The size of the value sets of the six attributes are 3, 3, 2, 3, 4 and 2, respectively as below.

$$\begin{array}{lcl} head\_shape & \in & \{\text{round, square, octagon}\} \\ body\_shape & \in & \{\text{round, square, octagon}\} \\ is\_smiling & \in & \{\text{yes, no}\} \\ holding & \in & \{\text{sword, balloon, flag}\} \\ jacket\_color & \in & \{\text{red, yellow, green, blue}\} \\ has\_tie & \in & \{\text{yes, no}\} \end{array}$$

Consequently, the example space consists of the total of $3*3*2*3*4*2=432$ possible examples. The three MONK's problems, called *M1, M2*, and *M3*, are all binary classifications defined over the same space. They differ in the type of the concept to be learned, and in the amount of noise in the training examples. Each problem is given by a logical description of a concept. Robots belong to either this concept or not, but instead of providing a complete concept description to the learning problem, only a subset of all 432 possible robots with its classification is given. The learning task is then to generalise over these examples and, if the particular learning technique at hand allows this, to derive a simple concept description. After a concept description has been produced by a learning algorithm from the training examples of each of the three problems, the whole 432 examples are used to test the accuracy of the concept description.

The three MONK's problems are specifically designed as below:

- Problem M1: *(head_shape = body_shape) OR (jacket_color = red)*

  From 432 examples (216 positive and 216 negative), 124 (62 positive and 62 negative) were randomly selected for the training set. There were no misclassifications in the training examples.

This problem is in standard disjunctive normal form (DNF) and is supposed to be easily learnable by all symbolic learning algorithms.

- Problem M2: *exactly two of the six attributes have their first value*

  From 432 examples (142 positive and 190 negative), 169 (105 positive and 62 negative) were randomly selected as training examples. Again, there was no noise in the training set.

This problem is said to be among the most difficult to learn using solely logic-based inductive learners (such as AQ11-like algorithms and ID3-like algorithms). It combines different attributes in a way which makes it complicated to describe in DNF or CNF (conjunctive normal form) using the given attributes only.

- Problem M3: *(jacket_color = green) AND (holding = sword) OR (jacket_color is NOT blue) AND (body_shape is NOT octagon)*

  From 432 examples (228 positive and 204 negative), 162 (60 positive and 62 negative) were selected randomly, and among them there were 5% misclassifications, i.e., noise, in the training set.

This problem is again in DNF but serves to evaluate learning algorithms under the presence of noise.

4.3 Results Produced by HCV

Appendix C records in detail the results produced by HCV (Version 1.0) on the three MONK's Problems. A default rule (with its condition being DEFAULT, see Section 3.5.4) there means that if none of the rules before the default rule are successful in matching a given test example, then the test example will be classified to the default class. Tables 4-1 and 4-2 provide a short overview of Appendix C.

As ID3 rules do not contain memberships in their conditions, Table 4-1 provides a third measurement, the *number of equivalent non-membership conjunctions*, to

Table 4–1: Rules (by HCV) from Training Sets

Measurement	Training Set 1	Training Set 2	Training Set 3
number of conjunctive rules	7	39	18
number of conjunctions	16	168	62
number of equivalent non-membership conjunctions	25	241	92

Table 4–2: Accuracy (by HCV) on Test Sets

Test Set 1	Test Set 2	Test Set 3
100%	81.25%	90.28%

convert the memberships in HCV rules to plain conjunctions. In this measurement, a membership like $X_j = [V_1, ..., V_n]$ is counted as n non-membership conjunctions. However, this measurement is only applicable to the comparison of HCV with ID3 and ID5R because all other algorithms such as ASSISTANT, AQR and CN2 have memberships in their rules. Actually, as we discussed in Section 2.3.4, being unable to group attribute values is simply a disadvantage of ID3.

4.4 Results Produced by Other Algorithms on Training Sets

The following tables, which are adopted from Chapters 6 and 3 of [Thrun *et al.* 91], show the results produced by ID3 (with and without windowing), ID5R, ASSISTANT, AQR and CN2 on the training sets of the three MONK's problems. We will not include ASSISTANT's results on the M3 problem in this and next sections because ASSISTANT has been intended (by pruning its decision tree) to detect and eliminate the misclassified examples in Training Set 3 and

Table 4–3: ID3 without Windowing on the MONK's Problems

Measurement	Training Set 1	Training Set 2	Training Set 3
number of non-leaf nodes	32	64	14
number of leaves	62	110	31

Table 4–4: ID3 with Windowing on the MONK's Problems

Measurement	Training Set 1	Training Set 2	Training Set 3
number of non-leaf nodes	13	66	13
number of leaves	28	110	29

therefore its results cannot 100% correctly classify all the training examples and are not comparable to those produced by other algorithms. The measurements for the decision trees produced by the ID3-like algorithms are the numbers of leaves and non-leaf nodes and for variable-valued logic rules produced by the AQ11-like algorithms are the numbers of conjunctive rules (or complexes) and conjunctions (selectors).

Table 4–5: ID5R on the MONK's Problems

Measurement	Training Set 1	Training Set 2	Training Set 3
number of non-leaf nodes	34	64	14
number of leaves	52	99	28

Table 4–6: ASSISTANT on the MONK's Problems

Measurement	Training Set 1	Training Set 2
number of non-leaf nodes	7	56
number of leaves	8	56

Table 4–7: AQR on the MONK's Problems

Measurement	Training Set 1	Training Set 2	Training Set 3
number of conjunctive rules	36	83	36
number of conjunctions	123	334	114

Table 4–8: CN2 on the MONK's Problems

Measurement	Training Set 1	Training Set 2	Training Set 3
number of conjunctive rules	10	58	24
number of conjunctions	13	145	38

Table 4–9: Number of Rules

Algorithm	Training Set 1	Training Set 2	Training Set 3
ID3 without Windowing	62	110	31
ID3 with Windowing	28	110	29
ID5R	52	99	28
ASSISTANT	8	56	
AQR	36	83	36
CN2	10	58	24
HCV	**7**	**39**	**18**

4.5 Performance Comparison

As each leaf in a decision tree corresponds to a conjunctive decision-tree-traversal rule, the number of leaves in a decision tree is equivalent to the number of conjunctive decision-tree-traversal rules. Table 4-9 lists the number of conjunctive rules produced by each of the induction algorithms mentioned so far in this chapter. From Table 4-9, we can clearly see that the rules produced by HCV are the most compact in terms of the number of conjunctive rules. Since non-leaf nodes in a decision tree are shared by different conjunctive decision-tree-traversal rules, the number of non-leaf nodes in a decision tree is not comparable to the number of conjunctions in the equivalent conjunctive decision-tree-traversal rules. With respect to the number of conjunctions generated, CN2 seems to be better than HCV. However, as we will discuss below, the rules produced by CN2 do not match the training examples exactly.

Of course, the comparison on Table 4-9 is not completely 'fair' because different algorithms have different notions of 'conjunctive rule'. For example, HCV, ASSISTANT, AQR and CN2 use memberships while ID3 and ID5R do not or cannot.

Table 4-10 provides numerical evaluation of the accuracy of the rules or decision trees produced by various algorithms based on the percentage of the test examples correctly classified. As we can see from Table 4-10, HCV works perfectly well on the M1 problem. Its accuracy is also among the best on the M2 problem. However, it does not perform that well in noisy environments like the M3 problem.

It is shown [Thrun *et al.* 91] that the rules produced by CN2 on Training Set 2 can only 92.90% correctly classify the original training set. This is a common case for many noise-tolerant AQ11-like algorithms such as CN2 and AQ15 because each of them has a threshold parameter to indicate the minimum percentage of selectors/conjunctions in a rule they generated that must be true for the rule to apply. The rules produced by them are intentionally incomplete and inconsistent with the training set since they were generated with some error tolerance. Sometimes,

Table 4–10: Accuracy

Algorithm	Test Set 1	Test Set 2	Test Set 3
ID3 without Windowing	83.24%	69.12%	95.60%
ID3 with Windowing	98.56%	67.92%	94.44%
ID5R	79.77%	69.23%	95.28%
ASSISTANT	100.00%	81.25% on 351 examples	
AQR	95.88%	79.63%	87.04%
CN2	100.00%	68.98%	89.12%
HCV	**100.00%**	**81.25%**	**90.28%**

this feature of noise-tolerant AQ11-like algorithms can recognize noise in training examples. For example, AQ14-NT, a version of the AQ11 algorithm that employs a noise-filtration technique, can recognise 100% correctly all the test examples in Test Set 3 after some loops of concept-driven filtration of training examples with truncation parameter equal to 10% and repeated induction. However, as we have discussed in Section 2.4, noise filtration or tolerance can only work usefully in noisy environments. It can be harmful in noise-free domains. For example, 100% correct classification of the original training examples should be a basic requirement for all learning algorithms in noiseless environments. The CN2 and AQ15 algorithms can easily violate this requirement.

4.6 Conclusions

In addition to the algorithms mentioned above, many other algorithms such as

- AQ17-DCI (a version of the AQ11 algorithm with data-driven constructive induction),

- AQ17-HCI (a version of the AQ11 algorithm with hypothesis-driven constructive induction)

- AQ15-GA (a version of the AQ11 algorithm combined with a genetic algorithm),
- AQ15-FCLS (a version of the AQ11 algorithm oriented toward learning flexible concepts),
- AQ14-NT (as mentioned in the last section),
- ECOBWEB (a *unsupervised* clustering system: examples are not preclassified as positive or negative examples there),
- Backpropagation (a function approximation algorithm for multilayer feed-forward networks based on gradient descent), and
- Cascade-Correlation (a neural network learning algorithm that builds a near-minimal multi-layer network topology in the course of training)

have also been applied to and tested on the three MONK's problems [Thrun *et al.* 91]. With respect to accuracy of the results generated by various algorithms including those have been mentioned in earlier sections, the neural network methods, Cascade-Correlation and Backpropagation (with weight decay), seem to be the best on the three MONK's problems. Their accuracy on the three test example sets are both 100%, 100% and 92.7% respectively. However, the result compactness and time of all those algorithms are not comparable because they have different knowledge representation and were implemented and tested by different people on different machines.

Chapter 5

KEshell2: An Intelligent Learning Data Base System

5.1 Introduction

KEshell2 is an improved version of *KEshell* [Wu 90, Wu 91] with the facilities of extracting knowledge from relational data bases. This chapter first gives an introduction to *KEshell* and then presents *KEshell2* with examples.

KEshell is an alternative tool to rule-based production systems based on an integration of rule-based and numeric computations. It adopts a 2-level representation language, *rule schema + rule body*, which is derived originally from Xiong *et al.*'s "rule skeleton + rule body" representation [Wu 90]. Rule schemata in the language are used to describe the hierarchy among factors or nodes in domain reasoning networks while rule bodies, which comprise computing rules as well as inference rules, are used to express specific evaluation methods for the factors and/or the certainty factors of the factors in their corresponding rule schemata. By representing explicitly numeric computation and inexact calculations as well as inference rules, the language supports a flexible way to process procedural knowledge and uncertainty. This representation can be easily adapted to rule induction: a rule schema can be used to describe the relationship between concepts or classes and features or attributes, and the corresponding rule body can express all the concrete rules produced by induction algorithms. Induction techniques can normally produce rules within a rule body, not the whole rule set.

KEshell2 is based on *KEshell*, with the $dBASE3$ package and the major contribution of this thesis described in Chapter 3 integrated to permit knowledge acquisition from relational data bases. The HCV algorithm and the ID3 algorithm have been implemented in the knowledge acquisition engine (K.A. Engine) of *KEshell2*. A deduction module which uses the rules produced by HCV to solve users' problems has been designed and implemented in the inference and deduction engine (I/D Engine) of *KEshell2*.

5.2 *KEshell*: A "Rule Schema + Rule Body" Based Knowledge Engineering Shell

5.2.1 Problems in production systems

Rule-based production systems are one of the most widely used models of knowledge representation in AI, in particular expert systems. Rather than expressing a logical calculus about the world as in Prolog-based systems or computing the numeric values defined over data as in conventional programming, production systems normally determine how the symbol structures that represent the current state of the problem should be manipulated to bring the representation closer to a solution. Problems which have been solved in production systems can usually be encoded in LISP or PROLOG, of course; the point is that production systems and rule-based programming languages are specifically designed to do them, and as a result they do them rather well [Jackson 90].

A production system is defined by three basic components:

- *a rule base* which consists of a collection of IF ... THEN statements called productions, production rules, or simply rules;

By convention, the IF part of a rule is called its LHS (left-hand side), and its THEN part is called its RHS (right-hand side). Both LHS and RHS of a rule usually adopt the grammar of either $(object, attribute, value)$ triples (such as in

OPS5 and EMYCIN) or $(attribute, value)$ pairs (such as in M.1, EXPERT and KES). An $(object, attribute, value)$ triple or an $(attribute, value)$ pair is called a condition when it appears in the LHS of a rule and called a conclusion when in the RHS. A rule in the rule base typically says that if its LHS holds then its RHS can be logically drawn and is thus called [Wu 90] *an inference rule* as opposed to *the control rules* (or *meta rules*) which are used, say, to determine how to apply other rules and *the computing rules* which will be introduced below in the rule schema + rule body language.

- *a working memory* which holds facts including the data, goal statement and intermediate results that make up the current state of the problem in solving;

Facts in the working memory can take the form of either $(object, attribute, value)$ triples or $(attribute, value)$ pairs with associated certainty factors giving the strength of belief in the values.

- *an inference engine* which decides when to apply which rules.

The inference engine typically operates according to the following "recognise–act" algorithm:

1. *Match.* Find the rules in the rule base whose LHSs are satisfied from the existing contents of the working memory.

2. *Conflict resolution.* Select one rule with a satisfied LHS by applying one or more conflict resolution strategies; if no rule is available in the rule base, stop.

3. *Act.* Adapt the working memory according to the RHS of the selected rule, perhaps adding a new item or deleting an old one.

4. Goto (1) for further 3-phase "match–conflict resolution–act" cycles.

There are many advantages in production systems [Wu 90, Barr *et al.* 81, Brownston *et al.* 85], such as expressibility, modularity, uniformity and naturalness, but there are also several significant disadvantages inherent in the formalism [Wu 90]. We summarize these below:

5.2.1.1 Low efficiency

Efficiency is an important consideration in production systems since they may be expected to exhibit high performance in interactive domains or real-time domains. However, chaining in production systems is much more complicated than testing the satisfiability of individual propositional formulae [Dowling & Gallier 84]. Non-worst-case subexponential algorithms are not possible for the general case [Tambe & Newell 88, Miranker *et al.* 90] and existing rule-based programming can not abolish combinatorial explosion completely [Jackson 90] although in practical applications, the conflict resolution strategies, such as LEX and MEA [Brownston *et al.* 85] employed in OPS5, tend to choose rules that lead to a reasonable solution.

In the basic forward chaining algorithm, each 3-phase "matching – conflict resolution – action" cycle deals with the problem of matching rules in a rule base with the working memory. However, the successful matching of a rule with the working memory does not always mean its immediate action. A rule may fail to match the working memory in an overall problem-solving process but it often needs to be tested in each 3-phase cycle. Meanwhile, some other rules may be successful in matching the working memory from the very beginning of a problem solving but always fail to get the priority of action in each conflict resolution phase. When there are changes in the working memory, they need to be tested again and again. Those problems cause low efficiency of the basic forward inference. For naive production system algorithms, all but the smallest systems are computationally intractable. Some earlier systems have been observed to spend more than 90% of their total run time performing matching [Forgy 82].

Even in Rete [Forgy 82] of the OPS5 language and TREAT [Miranker 87] of the DADO machine, the two commonly assumed best match algorithms, there are no definite solutions for those problems. Both Rete and TREAT were developed to avoid matching all rules with the working memory to find applicable rules on each cycle. For example, when rules in a rule base are first loaded into the system, OPS5 compiles them into a set of features to be checked. The features are mostly tests of values of attributes which are shared by different rules. They are connected in a tree-structured discrimination network that efficiently performs the matching process. However, the NP-hard problem has by no means been eliminated in OPS5 and DADO.

5.2.1.2 Lack of flexibility in expressing procedural knowledge

Symbolic computation, in which non-numeric symbols and symbol structures can be construed as standing for various concepts and relationships between them, has been characteristic of AI. Knowledge representation in AI is concerned mainly with the way in which symbolic information might be stored and large bodies of knowledge can be formally described for the purpose of symbolic computation. Although the distinctions between AI representations like production systems and procedural representations are clear [Jackson 90, Barr *et al.* 81], procedural knowledge has not been well integrated into production systems up till now. When the LHS and RHS in a rule contain all of 1) an assignment of a symbolic value in a discrete domain, 2) an assignment of a numeric function value and 3) a logical condition or conclusion, it is difficult to use existing production systems or rule-based programming languages – although such things can be represented, it is done in an *ad hoc* way, rather than as an integral part of the notation.

5.2.1.3 Lack of flexibility in inexact inference

There has been broad agreement among AI researchers that inexact inference is important in many expert system applications due to many different sources of uncertainty from imperfect domain knowledge and/or imperfect case data in AI problem solving; however, there is very little agreement concerning the form inexact inference should take [Jackson 90]. Five typical inexact models [Wu *et al.* 88, Jackson 90] which have been adopted in AI systems are probability theory based methods, the belief and disbelief model adopted in MYCIN, fuzzy logic, belief functions (e.g. the Dempster–Shafer theory on evidence) and incidence calculus, each having its inherent advantages and disadvantages.

A standard approach to implement inexact inference in existing AI systems is comprised of three components: a measure (e.g. a probability or a fuzzy degree) to describe imperfect data, a measure (e.g. a conditional probability or a rule strength) to represent imperfect rules and an inexact model which contains a set of computing formulae to evaluate the certainty factor of each conclusion in the RHS of a rule according to the certainty factors of all the conditions in the LHS of the rule. There are two problems in the normal approach:

- The inexact model once chosen is implicitly implemented inside the inference engine rather than being expressed explicitly. In other words, the execution of different rules uses the same set of computing formulae for their conclusions' certainty factors. This rigidity is inconvenient for some applications (see Example 5-3) where different conclusions need different operators to compute their uncertainty factors.

- Each inexact model has its own drawbacks. It would be quite useful in some applications if we can integrate several models together, say, using both probability calculus and fuzzy calculus. However, that is impossible with the standard implementation approach.

With respect to all of the considerations in Sections 5.2.1.1 to 5.2.1.3, we integrate rule-based and numeric computations into a 2-level description in our rule schema + rule body language to improve the production system structure. In contrast to the aforesaid problems, a linear forward chaining algorithm, *LFA* ([Wu 93d], also described in Appendix E), is supported by the language based on sorting the knowledge in a knowledge base into a partial order. Numeric computation and inexact calculus are explicitly expressed in the same way as inference rules in rule bodies in the language so that an expert can use different inexact models or their changed formulae when building a knowledge base.

5.2.2 Rule schema + rule body

5.2.2.1 The syntax

The grammar used in rule schema + rule body is that of *(factor, value)* pairs as opposed to *(object, attribute, value)* triples in OPS5 and EMYCIN.

Definition 5.1. A *factor*, which has a similar meaning to an attribute in M.1, EXPERT and KES, is a name involved in a domain expertise. It can be a logical assertion, a discrete set variable or a continuous numeric variable.

Definition 5.2. A rule schema has a rule-like structure with the general form of

$$\text{IF } E_1, ..., E_n \text{ THEN } A$$

or

$$\text{IF } E_1 \wedge ... \wedge E_n \text{ THEN } A$$

where all of $E_1, ..., E_n$ and A are factors. Each E_i $(i = 1, ..., n)$ is called a premise factor and A is called the conclusion factor in the rule schema.

In rule induction, each E_i is an attribute and each value of A indicates a class to learn.

Definition 5.3. Each rule schema IF $E_1, ..., E_n$ THEN A has a corresponding rule body which contains all the available information to evaluate the value of A (when A is a non-logical variable) and/or the certainty factor of A. All the factors included in the rule body must appear in the rule schema so that when there are data for all of $E_1, ..., E_n$, A can definitely be evaluated by the rule body if the information inside is complete.

Definition 5.4. *A computing rule* is different from an inference rule in that its RHS is an assignment to a non-logical variable or a certainty factor. For example, *IF A>10, CF(B)>0.4 THEN X=sin(C)+max{0.5, cos(D)}*.

In each rule body, there may be one or more inference rules similar to those in production systems and/or computing rules for computation. All the rules in a rule body are used to determine the value of the conclusion factor in its corresponding rule schema and/or the certainty factor (CF) of the value/factor. When the conclusion factor is a logical assertion, the rule body can be used to compute the CF of the assertion. When the conclusion factor is a set variable or a numeric variable, the rule body is used both to evaluate the value of the factor and to compute its CF. Thus, the computation of the CF of a factor can be processed in the same way as the evaluation of non-logical factors, both being explicitly expressed in rule bodies. When all the factors in a domain expertise are logical assertions and all the rule bodies have the same rules for computing CFs, the inexact inference then behaves similarly to the standard implementation approach mentioned in Section 5.2.1.3. When all the factors are numeric variables and no uncertainty calculus is needed, all the rule bodies will be used to express computation models and a rule schema plus its rule body is analogous to a procedure or function in conventional programming.

Definition 5.5. A rule schema with its corresponding rule body is called a *rule set.*

A rule set is an independent knowledge unit in the rule schema + rule body language. It can be described in BNF (*Backus Normal form*) as follows.

<rule set> := Rule Set # <rule set number><rule schema><rule body>

<rule set number> := <integer>

<rule schema> := Schema: IF <premise factors> THEN <conclusion factor>

<premise factors> := <premise factor>{, <premise factors>}

<premise factor> := <factor>

<conclusion factor> := <factor>

<factor> := <variable name>|<logical assertion>

<logical assertion> := <predicate(object)>|<predicate>

<rule body> := Body: (<C-rule>|<I-rule>){<rule body>}

<C-rule> := (<factor>|CF(<factor>))= <assignment expression>

<assignment expression> := <value>|<algebraic expression>

<I-rule> := IF<antecedents>THEN<conclusion>

<antecedents> := <antecedent>{,<antecedents>}

<antecedent> := (<factor>|CF (<factor>))<relation-sym> <assignment expression>

<relation-sym> := >|<|=|<>|>=|<=

<conclusion> := <C-rule>

<value> := <integer>|<real>|<symbolic value> |<probability>| <fuzzy degree>

The terms <variable name>, <predicate(object)>, <predicate>, <algebraic expression> and different kinds of values above have the standard interpretation.

Clearly, the rule schema + rule body representation can be easily adapted to rule induction. A rule schema can be used to describe the relationship between concepts or classes and features or attributes, and the corresponding rule body can express all the concrete rules produced by induction algorithms. Induction techniques have the job of learning a rule body instead of the whole rule set.

5.2.2.2 Advantages of the language

As compared with production systems, the rule schema + rule body language has five main advantages:

1. it can avoid matching all the rules in a knowledge base with the working memory at run time when some piece of data is not available;

Suppose we have a rule schema "If A_1, A_2, A_3, A_4, A_5 then B" and all $A_1, ..., A_5$ have a value domain {1,...,1000}, there are at least 1000^5 possible decision-tree like rules with form of or similar to

$$\text{If } A_1 = N_1, A_2 = N_2, A_3 = N_3, A_4 = N_4, A_5 = N_5 \text{ then ...}$$

in total where $N_i \in \{1, ..., 1000\}$ $(i = 1, ..., 5)$. Now if we have a large quantity of data on A_1, A_2, A_3 and A_4 but no data ab out A_5, only 5 comparisons are needed: Once no data about A_5 is found during matching of the rule schema, all of the rules in the rule body will be neglected without matching.

A linear forward chaining algorithm (*LFA*) will be further described in Appendix E based on this advantage.

2. it gives a naturally structured organisation of knowledge bases in terms of rule sets;

The self-contained format (see Definition 5.3) of rule sets and their communication channel (the working memory in grammar of $(factor, value)$ pairs with associated certainty factors giving the strength of belief in the values, see Section 5.2.3.1) clearly enhance the modularity of the language. One can edit, modify and execute a rule set independently without much attention to other rule sets. Modularity is an important virtue for AI programs (like other software) because it makes programs easier to understand, explain and modify.

3. it gives an expressive representation of computation knowledge;

Example 5-1. The knowledge for solving the general equation $AX^2 + BX + C = 0$ can be expressed in rule schema + rule body as follows.

Rule schema: *IF A, B, C THEN X*

Rule body:

IF A=0, B≠0 THEN X=-C/B

IF A≠0 THEN X=(-B+$\sqrt{B^2 - 4AC}$*)/(2A)*

IF A≠0 THEN X=(-B-$\sqrt{B^2 - 4AC}$*)/(2A)*

4. the flexible expression of uncertainty calculus in rule bodies.

Example 5-2. In fuzzy multi-objective judgement problems [Wu *et al.* 87], we often need different judging operators, such as $M(\wedge, \vee), M(\otimes, \oplus)$[1], $M(\wedge, \oplus)$ and $M(\otimes, \vee)$, for different objectives to evaluate their fuzzy degrees. It is easy to define a specific judging operator for each objective in the rule sets where the objective is the conclusion factor.

Also, a domain expert can define CFs as either probabilities or fuzzy degrees, as required. It is possible to use both probability calculus and fuzzy calculus in a knowledge base for inexact inference in rule schema + rule body. However, this mainly depends on the calculus formulae and possible transformation defined in rule bodies. For each factor in a domain expertise, its CF will be computed according to the rule bodies which have it as their conclusion factor.

5.2.2.3 A working example

Example 5-3 shows a condensed excerpt from a knowledge base in rule schema + rule body for an assessment problem.

Example 5-3. An assessment problem in a given area has three objectives: numeric Object_1, fuzzy Object_2 whose value domain is $\{A,\ B,\ C\}$ and logical Object_3. Object_1 is defined on $X1, X2$ and $X3$, whose value can be evaluated from the values of either $X31$ and $X32$ or $X33, X34$ and $X35$. All of Xi's ($i = 1, 2, 3, 31, 32, 33, 34, 35$) are numeric while all of $Y1, Y2$ and $Y3$, which are used to define Object_2, have the fuzzy domain of $\{A,\ B,\ C\}$. Object_3 can be inferred from $X35$ and two pieces of logical evidence, $Z1$ and $Z2$. The expertise given by an anonymous expert is shown below in rule schema + rule body. Words between /* and */ are explanatory notes.

---

[1]

$$a \oplus b = a + b \tag{5.1}$$

$$a \otimes b = ab \tag{5.2}$$

where a and b are fuzzy degrees.

Rule Set #1

Schema:

IF Object_1, Object_2, Object_3 THEN Assessment

Body:

IF Object_1>85, Object_2=A, CF(Object_3)>0.7 THEN Assessment=A

IF Object_1>85, Object_2=A THEN Assessment=B

/* A further condition, CF(Object_3)<=0.7, is implied in the second rule. */

IF CF(Object_3)>0.5, Object_1>60, Object_2<>C THEN Assessment=B

/* '<>' denotes ≠. */

Assessment=C

/* When none of the above three rules can be satisfied, this rule is unconditionally applicable. */

IF CF(Object_3)>0.5 then CF(Assessment)=CF(Object_2)

CF(Assessment)=min{0.5, CF(Object_2)}

Rule Set #2

Schema:

IF X1, X2, X3 THEN Object_1

Body:

Object_1=(X1+X2+X3)/3

Rule Set #3

Schema:

IF X31, X32 THEN X3

Body:

X3=max{X31, X32}

Rule Set #4

Schema:

IF X33, X34, X35 THEN X3

Body:

X3=(min{X33,X34}+X35)/2

Rule Set #5

Schema:

IF Y1, Y2, Y3 THEN Object_2

Body:

IF Y2<>C THEN Object_2=Y1

IF Y2=C THEN Object_2=Y3

CF(Object_2)=min{1-CF(Y1)CF(Y2)CF(Y3), CF(Y1), CF(Y2), CF(Y3)}

Rule Set #6

Schema:

IF X35, Z1, Z2 THEN Object_3

Body:

IF X35>75 THEN CF(Object_3)=(CF(Z1)+CF(Z2)-CF(Z1)CF(Z2))/2

CF(Object_3)=(CF(Z1)+CF(Z2))/(1-min{CF(Z1),CF(Z2)})

The uncertainty calculus in Rule Sets #5 and #1 is defined on fuzzy logic while in Rule Set #6, CF(Z1), CF(Z2) and CF(Object_3) are defined on probabilities. In Rule Sets #2, #3 and #4, uncertainty calculus is unnecessary because all of Xi's ($i = 1, 2, 3, 31, 32, 33, 34, 35$) are supposed to be real numbers without uncertainty. The different CFs defined on fuzzy degrees and probabilities are unified in Rule Body #1 to give the final fuzzy CF to the goal Assessment.

5.2.3 *KEshell*: a rule schema + rule body based knowledge engineering shell

5.2.3.1 Overview

KEshell is a rule schema + rule body based knowledge engineering shell, with its structure shown in Figure 5-1.

In the diagram, KB denotes knowledge bases and KBMS is a knowledge base management subsystem that detects the repetitions, redundancy and contradictions which will be described in Section 5.2.3.2, finds dead cycles in a knowledge base and sorts it. *KEshell* has been implemented in Turbo-PROLOG on IBM PC computers.

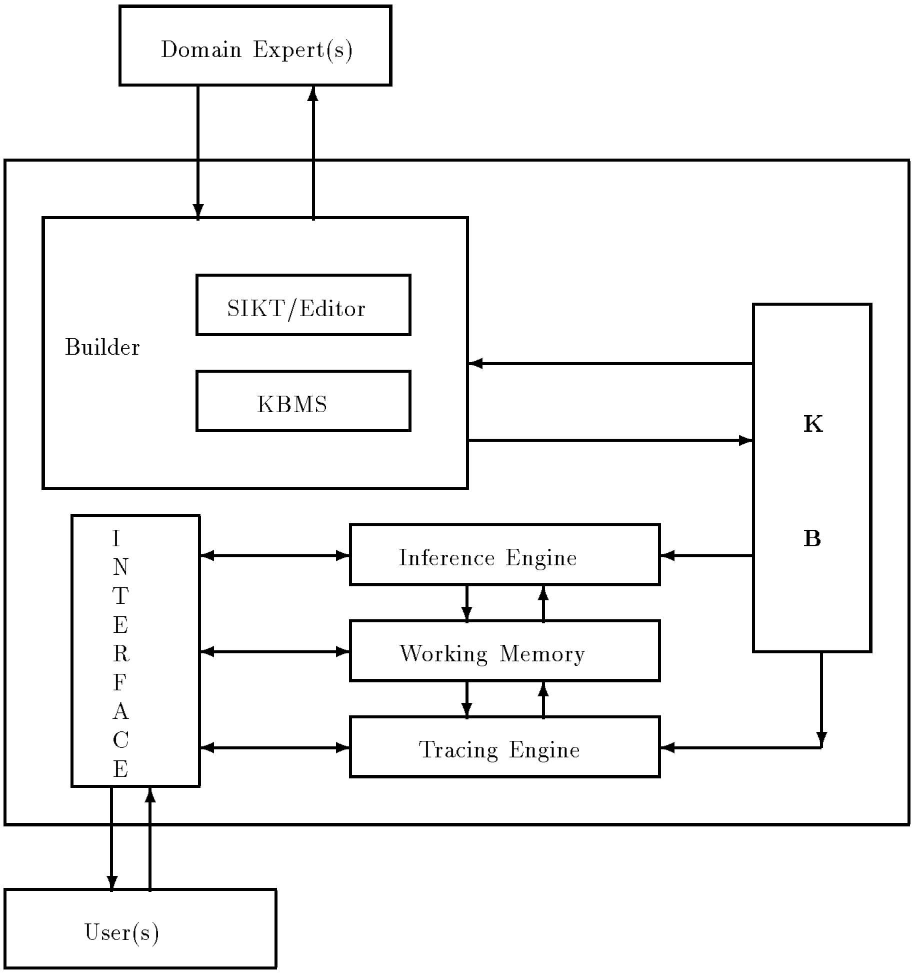

Figure 5–1: The System Structure of *KEshell*

Facts in the working memory are represented as $(factor, value)$ pairs with accompanying certainty factors, which can be either probabilities or fuzzy degrees. Rule schemata and rule bodies in KB represent domain expertise. As the factors in a rule schema are allowed to be variables, it is easy to enter many concrete rules, inference rules and/or computing formulae, in one generic rule schema and then take the rule body as a "look-up" table.

The inference engine in *KEshell* supports both forward and backward chaining with the forward chaining working in linear time (see Appendix E). When there is a dead cycle in a knowledge base, which means there is some error in the knowledge base that cannot be removed, only backward chaining can be adopted. *KEshell* has a special engine in KBMS to detect cycles in knowledge bases.

There are two ways to build knowledge bases. The first is the structured interactive knowledge transfer module (*SIKT*) which will be described in Section 5.2.3.2. As we will show, the communication between experts and *SIKT* is quite friendly. An expert does not need to know much about knowledge engineering or programming to build a knowledge base in this mode, just answering all the questions *SIKT* asked. The other way is to use a text editor, which will be described in Section 5.2.3.3. All other modules will also be described in detail below.

5.2.3.2 Structured interactive knowledge transfer — SIKT

KEshell was intended to be an expert system building tool (not a programming language) with which experts were expected to interact directly to build knowledge bases without the aid of knowledge engineers (or expert systems programmers). The *SIKT* in *KEshell* is a module that guides experts in inputing their expertise in a top-down mode and an interactive manner.

For knowledge transfer, two work buffers are set up: a defined factor set (DFS), where all the factors that have been given evaluation methods are put and an undefined-factor stack (UDF), which is a set of all the factors whose evaluation methods have not been given by expert(s).

The guidance procedure is designed as follows:

Step-1: DFS=ϕ, UDF=ϕ;

Step-2: Ask the expert to give all the top factors in his/her domain reasoning network and put all the factors given by the expert into UDF;

Step-3: If UDF=ϕ then goto Step-7;

Step-4: POP(Factor) from UDF, if Factor appears in DFS then goto Step-3;

Step-5: i) Ask the expert if Factor is a terminal factor in the domain reasoning network, which means that users will provide possible evidence for it for inference. If yes then put Factor into DFS and goto Step-3;

ii) Acquire a new rule schema whose conclusion factor is Factor: ask the expert to give all the factors that can determine the Factor in a new evaluation method and put all the new factors that have not existed in DFS or UDF into UDF and write down a new rule schema in form of *IF all the factors THEN Factor* into the knowledge base;

Step-6: Ask the expert if there is any other new method to determine Factor, if yes then goto Step-5 else put Factor into DFS and goto Step-3;

Step-7: Check for semantic inconsistencies in the schemata given;

There are three types of semantic inconsistencies: 1) dead cycles (see *Section 5.2.3*), 2) repetitions, and 3) knowledge insufficiency. When there are two rule schemata which are the same in a knowledge base, we say there exists a repetition in the knowledge base. The expert will be asked to integrate the same rule schemata into one when a repetition is found. If there exists such a factor in a knowledge base that is neither a top factor (e.g. a disease) nor a terminal factor (e.g. a symptom) in the corresponding domain reasoning network and there is no rule schema which has the factor as its conclusion factor, the knowledge base is incomplete. The expert will be asked to input more knowledge on the factor when the factor is discovered.

Step-8: For each rule schema acquire its rule body: Ask the expert to input each rule in a line and put the rule into the knowledge base. This process is repeated until the expert answers with an empty line;

Step-9: Check for semantic inconsistencies in each rule body;

The semantic inconsistencies in a rule body mean three possible aspects: 1) repetitions, 2) redundancy, and 3) contradictions. When there are two rules in a rule body which are the same, we say there exists a repetition in the rule body and one of them will be automatically deleted. When there are two rules in a rule body which have the same RHS but the LHS of one contains the LHS of the other, we say there exists redundancy in the rule body and the former rule will be automatically deleted. A contradiction in a rule body indicates two such rules in the rule body that have the same LHS but different RHSs. Contradictions will be displayed to the expert so that he may make appropriate alterations.

Step-10: Call Step-7 and Step-9 until no changes and no inconsistencies are found;

Step-11: Let the expert approve the knowledge base that has just been built: If the expert declares that there are still some modifications, then allow expert to modify the existing knowledge base and goto Step-10 to check consistency;

Step-12: Sort the existing knowledge base into a partial order.

The sorting algorithm has been introduced in Section 5.2.3.

The dialogue between an expert and *SIKT* to build the knowledge base in Example 5-3 is recorded in Appendix F.

In contrast to KBEmacs [Waters 85], whose power came principally from the ability to construct a program out of the algorithmic fragments stored in a library, *SIKT* does not act as a general program editor for programmers but is able to automatically build executable knowledge bases in rule schema + rule body out of the dialogue with experts.

5.2.3.3 The editor

The syntax for the editor is the rule schema + rule body language in BNF plus a declaration of factors' types and domains for those factors which are set variables. For instance, the knowledge base file for Example 5-3 can be edited as the following declaration plus the description of rule sets in Section 5.2.2.3.

```
factor_type(Assessment, 2)
factor_type(Object_1, 3)
factor_type(Object_2, 2)
factor_type(Object_3, 1)
factor_type(X1, 3)
factor_type(X2, 3)
factor_type(X3, 3)
factor_type(X31, 3)
factor_type(X32, 3)
factor_type(X33, 3)
factor_type(X34, 3)
factor_type(X35, 3)
factor_type(Y1, 2)
factor_type(Y2, 2)
factor_type(Y3, 2)
factor_type(Z1, 1)
factor_type(Z2, 1)
factor_set(Assessment,[A, B, C])
factor_set(Object_2,[A, B, C])
factor_set(Y1,[A, B, C])
factor_set(Y2,[A, B, C])
factor_set(Y3,[A, B, C])
```

5.2.3.4 The tracing engine

There is a tracing engine in *KEshell* for knowledge base debugging at run time. It first labels all the knowledge that has been used during inference, then deletes the knowledge which is useless for concluding the goal(s) even if it was used during the inference and has been labeled, and finally displays the inference process by

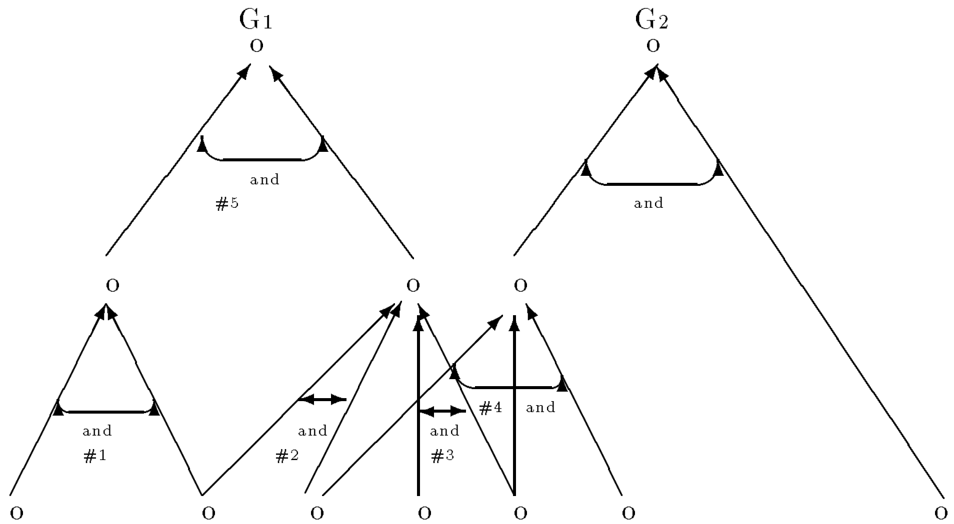

Figure 5–2: An Example for the Trace Engine

using only the knowledge that is still labeled. For example, suppose rule sets #1, #2, #3, #4 and #5 in Figure 5-2 have been executed during a problem solving session, the #4 rule set is useless for reaching the goal G1. The tracing engine can also work in linear time by displaying rule sets according to their renumbered ordinals in a similar way to the linear forward chaining algorithm in Appendix E.

5.2.3.5 The interface

The interface for users is even simpler. Questions concerning evidence are asked, answers are obtained, the chaining process takes place, and the solution(s) are given. The following is a sample dialogue using the knowledge base shown in Example 5-3. The sentences with K: at the beginning are generated by *KEshell* and those with M: are input of a user.

K: Fact acquisition now:

K: Can you provide data for X1? (Y|N|Q:Stop giving more data)

M: Y

K: The value:

M: 79

K: CF (DEFAULT=1):

M:

/* an empty line */

K: Can you provide data for X2? (Y|N|Q:Stop giving more data)

M: Y

K: The value:

M: 79

K: CF (DEFAULT=1):

M:

K: Can you provide data for X33? (Y|N|Q:Stop giving more data)

M: Y

K: The value:

M: 89

K: CF (DEFAULT=1):

M:

K: Can you provide data for X34? (Y|N|Q:Stop giving more data)

M: Y

K: The value:

M: 79

K: CF (DEFAULT=1):

M:

K: Can you provide data for X35? (Y|N|Q:Stop giving more data)

M: Y

K: The value:

M: 79

K: CF (DEFAULT=1):

M:

K: Can you provide data for X31? (Y|N|Q:Stop giving more data)

M: Y

K: The value:

M: 79

K: CF (DEFAULT=1):

M:

K: Can you provide data for X32? (Y|N|Q:Stop giving more data)

M: N

K: Can you provide data for Y1? (Y|N|Q:Stop giving more data)

M: Y

K: The value: (Domain: {A, B, C})

M: D

K: Wrong input! Try again: (Domain: {A, B, C})

M: B

K: CF (DEFAULT=1):

M: 0.6

K: Can you provide data for Y2? (Y|N|Q:Stop giving more data)

M: Y

K: The value: (Domain: {A, B, C})

M: A

K: CF (DEFAULT=1):

M: 0.8

K: Can you provide data for Y3? (Y|N|Q:Stop giving more data)

M: Y

K: The value: (Domain: {A, B, C})

M: B

K: CF (DEFAULT=1):

M: 0.75

K: Can you provide data for Z1? (Y|N|Q:Stop giving more data)

M: Y

K: CF (DEFAULT=0):

M: 0.8

K: Can you provide data for Z2? (Y|N|Q:Stop giving more data)

M: Y

K: CF (DEFAULT=0):

M: 0.6

K: Linear forward chaining now ...

K: The solution for your problem is

K: Assessment=C with CF=0.5

5.2.4 Summary

Rule schema + rule body is a representation language which integrates numeric computation, inexact calculus and logical inference. We have described the representation language and the structured interactive knowledge transfer module (*SIKT*) in *KEshell* in detail in this section. The production system model used in this section, which the author would take as the standard model, is much more like that used in OPS5 and text books [Jackson 90, Barr *et al.* 81] than other systems or languages where efficiency gains are at the expense of representational power.

The major feature of *KEshell* is that the shell has supported all the techniques mentioned in this section and put them to practical use. As compared with large system programming languages [Harmon *et al.* 85], such as OPS5, ART and KEE, inexperienced programmers and subject matter experts can interactively build knowledge bases in *KEshell* by calling the *SIKT* module. The price paid for this is that the representation in *KEshell* is not so powerful as in these programming languages. For example, users can define control in an OPS5-like rule but they cannot do so in *KEshell*.

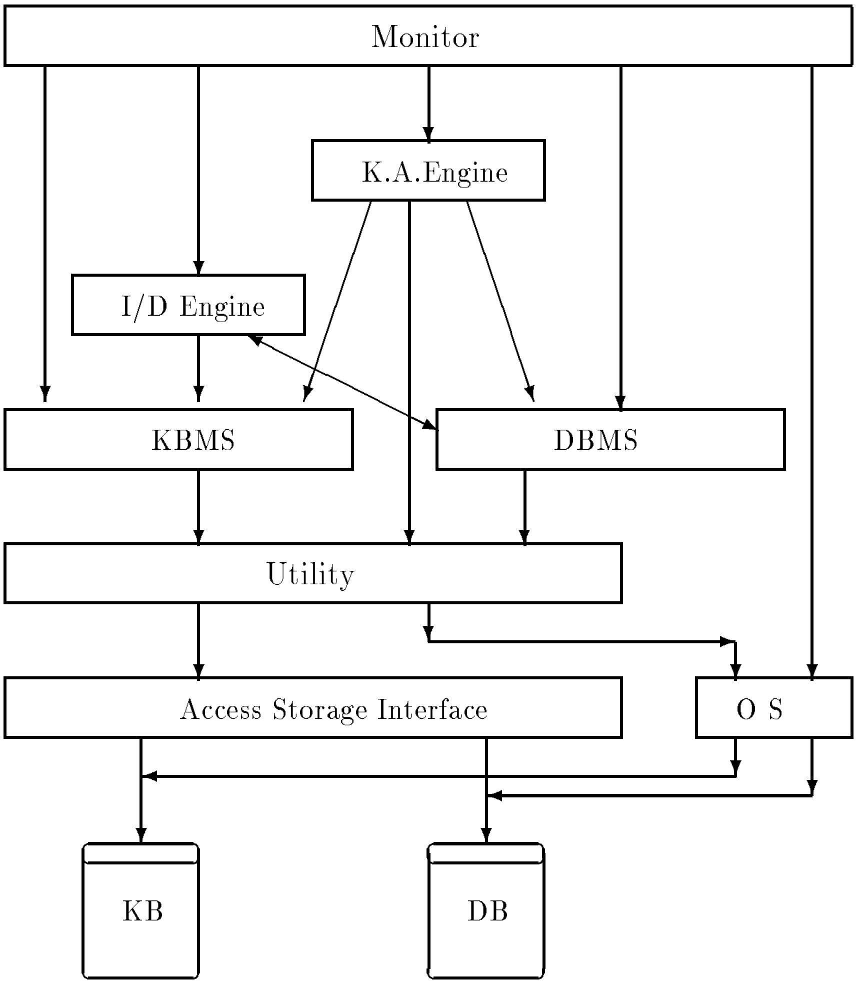

Figure 5–3: The System Structure of *KEshell2*

5.3 *KEshell2*: A Knowledge Engineering Shell which Couples ML Techniques with Data Base and Knowledge Base Technology

5.3.1 System structure

Figure 5-3 shows the system structure of *KEshell2*, the improved version of *KEshell* with the knowledge acquisition engine, K.A. Engine, which implements induction from data bases, and *dBASE3* being integrated.

In the diagram, Monitor is a man-machine interface which exchanges information with users in the form of pull-down menus. I/D Engine is an inference and deduction engine based on the inference engine of *KEshell*. KBMS and DBMS are facilities adopted mainly from *KEshell* and *dBASE3* respectively to support knowledge base and data base management functions. Utility contains a set of common procedures that are shared by K.A. Engine, KBMS and DBMS. Access Storage Interface is composed of the basic knowledge/data operators. DB and KB denote data bases and knowledge bases respectively and OS indicates operating system facilities. For the implementation of $KEshell2$, the operating system used was PC-DOS, referred to as DOS hereafter.

5.3.2 Monitor

The Monitor module in *KEshell2* accepts users' operational commands and calls corresponding functional modules in the system.

There are five options in its main menu: 1. KBMS; 2. I/D Engine; 3. K.A. Engine; 4. DBMS; and 5. DOS with their second-level menus being as follows.

- KBMS: 1. Build a Knowledge Base (*SIKT*), 2. Adapt Knowledge, 3. Find Cycles, 4. Sort a Knowledge Base, 5. List Rule Schemata, 6. List Concrete Rules, 7. Edit a KB File, and 8. Clear Working Memory.

- I/D Engine: 1. Forward Chaining, 2. Backward Chaining, 3. Deduction, 4. Knowledge Trace, 5. Clear Evidence, and 6. Adapt Facts.

- K.A. Engine: 1. Build a Knowledge Base (*SIKT*), 2. Semantic Information, 3. Rule Induction by HCV, and 4. Rule Induction by ID3.

- DBMS: 1. Enter dBASE3, and 2. List a Relation.

- DOS: 0. Enter PC-DOS, 1. Load a KB File, 2. Save Knowledge, 3. Directory, 4. Print, 5. Copy, 6. Delete, 7. Rename, 8. Time, 9. Date, 10. List Facts, 11. Adapt Facts, 12. Save Working Memory, 13. Edit a Text File, 14. List a DOS File, and 15. Quit.

The functions of most of the second-level submodules are just what their names imply.

5.3.3 KBMS

The KBMS module in *KEshell2* is adopted from the *KEshell* system. It supports facilities for interactively building, adapting and displaying knowledge bases, checking for semantic inconsistencies including dead cycles, sorting knowledge bases into partial order to implement linear forward chaining and editing knowledge base files.

All the submodules in KBMS are the same as in *KEshell* and have been introduced in Section 5.2.

5.3.4 DBMS

The DBMS module in *KEshell2* is based on *dBASE3*, a commercial relational data base management system. Users can do conventional data base operations by simply entering *dBASE3*. However, a new function, *List a Relation*, is developed here to translate *dBASE3* files into the Prolog-based representation described in Appendix D. The representation binds relational tuples and schema together in a natural and flexible way.

For example, XDBASE3.DBF and XDBASE3.DBT are two sample *dBASE3* files from *Turbo Prolog Toolbox* [Borland 87]. The relational schema and tuples included in them are listed in Tables 5-1 and 5-2 respectively.

The *List a Relation* submodule in DBMS reads by calling the *XDBASE3.PRO* file in *Turbo Prolog Toolbox* and translates them into the following predicate.

```
relation("XDBASE3",
[field("NAME", string),
field("BIRTH_DATE", string),
field("SALARY", real),
field("AGE", integer),
```

Table 5–1: Relational Schema of XDBASE3.DBF

FIELD	*TYPE*	*WIDTH*
NAME	string	25
BIRTH_DATE	date	8
SALARY	numeric	8.2
AGE	numeric	2
MEMO	memo	

Table 5–2: Tuples in XDBASE3.DBF and XDBASE3.DBT

NAME	*BIRTH_DATE*	*SALARY*	*AGE*	*MEMO*
Frank Borland	19131205	10250.95	73	Frank Borland's memo
Joe Programmer	19600707	45000	26	Joe Programmer's memo
Bit Twiddler	19521117	37000	33	Bit Twiddler's memo
Mary Martin	19500227	37000	36	Mary Martin's memo

field("MEMO", string)],

[tuple("Frank Borland", "19131205", "10250.95","73","Frank Borland's memo"),

tuple("Joe Programmer", "19600707", "45000", "26", "Joe Programmer's memo"),

tuple("Bit Twiddler", "19521117", " 37000", "33", "Bit Twiddler's memo"),

tuple("Mary Martin", "19500227", "37000", "36", "Mary Martin's memo")])

5.3.5 I/D Engine

All the submodules except Deduction in I/D Engine are taken from *KEshell* and have been described in Section 5.2. The Deduction submodule is designed to interpret the rules produced by the HCV algorithm to solve users' problems.

Taking the rules produced by HCV for Example 3-2 in Section 3.4.2 as an example, the following is an example run of the Deduction submodule.

K: The rules (Hfl's) in the working memory are:

K: if [OUTLOOK=overcast] then [DECISION=Play]

K: if [WINDY=false] & [OUTLOOK=rain] then [DECISION=Play]

K: if [TEMPERATURE=mild] & [WINDY=false] then [DECISION=Play]

K: Please input your example in form of <field/attribute>=<value>:

M: OUTLOOK=rain, TEMPERATURE=hot, HUMIDITY=normal, WINDY=false

K: According to the 2nd hfl, (i.e.,

K: if [WINDY=false] & [OUTLOOK=rain] then [DECISION=Play]

K:), the deduction result is: Play.

5.3.6 K.A. Engine

There are four submodules in the K.A. Engine in *KEshell2*: 1) Build a Knowledge Base (*SIKT*), which is the same as in KBMS, 2) Semantic Information, which will be described below, 3) Rule Induction by HCV, which implements the HCV algorithm described in Chapter 3, and 4) Rule Induction by ID3, which implements the ID3 algorithm summarized in Chapter 2. The third, Rule Induction by HCV is the core of the *KEshell2* system.

5.3.6.1 Semantic information

The theme of *KEshell2* is the provision of mechanisms for extracting knowledge from data bases. To couple machine learning techniques with data base technology, some semantic information (see Appendix D, such as causalities and logic implications) among relational data is used to trigger induction engines in the Semantic Information submodule.

The submodule works according to the following steps.

1. Find all relations in the working directory, which can be set or changed by the Directory submodule in the DOS module;

2. Ask the user to identify which of the relations is a relationship and of either causality or logical implication type;

3. Read the schema in the identified relation and ask the user to describe the causality structure between the fields in the schema;

4. Read the tuples in the relation and translate them into executable forms for the Rule Induction by HCV submodule.

The translation in Step 4 is based on the Prolog-based representation in Appendix D.

The following is an example run of this submodule on the example shown in Tables 5-1 and 5-2.

K: All the relations in the current working memory have been detected are:

K: XDBASE3, AVAL_FLT

K: Please indicate which of them is a relationship AND

K: of either causality or logical implication type:

M: XDBASE3

K: Now, please identify the semantic labels of attributes

K: in the relationship:

K: (All the fields in the XDBASE3 relationship are

K: NAME, BIRTH_DATE, SALARY, AGE, MEMO)

K: The result field:

M: SALARY

K: and the condition fields:

M: NAME, AGE

K: O.K. The logic structure is:

K: IF NAME, AGE then SALARY

K: Would you like to induct rules from your current data bases now?

M: N

If 'Y'(yes), this submodule will automatically call the Rule Induction by HCV submodule.

5.3.6.2 Induction from data bases

Data bases can be provided in three different ways to the Rule Induction by HCV submodule.

1. Induction from *dBASE3* files.

   When data is provided by *dBASE3* files, the Rule Induction by HCV sub-module first collects all the different values the result field takes and then runs the HCV algorithm after using part of the AQ11 technique introduced in Chapter 2 (by assuming that all examples not classified as positive are negative):

   for each different value the result field takes

   (a) convert all the tuples whose result field takes the value into positive examples;

   (b) convert all the other examples into negative examples; and

   (c) apply the HCV algorithm.

2. Induction from conventional DOS data files.

   When data is provided by a data file in the form of *positive examples* and *negative examples*, for instance, like the DATA31.DB below, induction by HCV only produces rules for positive examples.

```
/* file: DATA31.DB */
attributelist(["OUTLOOK", "TEMPERATURE", "HUMIDITY", "WINDY"])
result("DECISION")
n([rain,hot,high,true])
n([rain,cool,normal,true])
p([overcast,hot,high,true])
p([overcast,mild,normal,false])
p([rain,hot,high,false])
p([overcast,cool,normal,true])
n([sunny,hot,normal,true])
n([sunny,mild,high,true])
p([sunny,mild,normal,false])
p([overcast,mild,normal,false])
p([rain,cool,normal,false])
```

```
p([rain,hot,high,false])

n([sunny,hot,high,false])

n([sunny,cool,normal,false])
```

3. Induction by using part of the AQ11 technique.

   When data in a data file takes the form of *examples* rather than being distinguished as positive examples and negative examples, for example, like the DATA33.DB file below, the Rule Induction by HCV submodule works in a similar way to induction from *dBASE3* files: it first collects all the different values the result attribute takes and then runs the HCV algorithm on each different value by assuming that all examples not classified as positive are negative.

```
/* file: DATA33.DB */

attributelist([“OUTLOOK”, “TEMPERATURE”, “HUMIDITY”, “WINDY”])

result(“DECISION”)

eg(“Don’t Play”, [rain,hot,high,true])

eg(“Don’t Play”, [rain,cool,normal,true])

eg(“Play”, [overcast,hot,high,true])

eg(“Play”, [overcast,mild,normal,false])

eg(“Play”, [rain,hot,high,false])

eg(“Play”, [overcast,cool,normal,true])

eg(“Don’t Play”, [sunny,hot,normal,true])

eg(“Don’t Play”, [sunny,mild,high,true])

eg(“Play”, [sunny,mild,normal,false])

eg(“Play”, [overcast,mild,normal,false])

eg(“Play”, [rain,cool,normal,false])

eg(“Play”, [rain,hot,high,false])

eg(“Don’t Play”, [sunny,hot,high,false])

eg(“Don’t Play”, [sunny,cool,normal,false])
```

   The Rule Induction by HCV submodule produces five rules for examples in DATA33.DB:

if [OUTLOOK=overcast] then [DECISION=Play],

if [WINDY=false] & [OUTLOOK=rain] then [DECISION=Play],

if [TEMPERATURE=mild] & [WINDY=false] then [DECISION=Play],

if [OUTLOOK=[sunny,rain]] & [WINDY=true] then [DECISION=Don't Play], and

if [OUTLOOK=sunny] & [TEMPERATURE=[cool,hot]] then [DECISION=Don't Play].

Meanwhile, the Rule Induction by ID3 submodule is designed to work only on the data in the form of positive examples and negative examples. We can, of course, incorporate part of the AQ11 technique into ID3 and run it on each subset of the examples which have the same value on the result attribute or field; the inconvenience is that all the descriptions produced for negative examples at each ID3's run need to be thrown away. We can also extend the entropy measure in ID3 to chunk examples into more than two classes. However, we have not yet done these kinds of extensions in *KEshell2*. In those cases like DATA33.DB where only two subsets exist, data files are supposed to be organised in the form of DATA31.DB for ID3 and DATA33.DB for HCV. For the example set in Example 3-2, HCV produces five conjunctive rules as shown above for both *Play* and *Don't Play* as opposed to seven by ID3 (see Appendix A).

5.4 Conclusions

KEshell2 has coupled machine learning techniques with both data base and knowledge base technology. It is an intelligent learning data base system with mechanisms for 1) translating *dBASE3* files into a form suitable to its rule induction engine, 2) using induction techniques to extract knowledge from data bases, and 3) interpreting the knowledge produced to solve users' problems.

Chapter 6

Conclusions

As we have discussed in Section 2.6, knowledge acquisition from data bases is an important research frontier for both machine learning and data base technology [Wu 93b]. Although a lot of work has been done and some commercial packages are available already, existing work has concentrated on the following four aspects: 1) building knowledge bases for expert systems, 2) designing various learning algorithms; 3) adding an induction engine to an existing data base system in an *ad hoc* way to implement rule induction from data bases; and 4) designing a specific engine to learn from a domain-specific data set. The requirements [Quinlan 89b] for knowledge acquisition from realistic data bases are still out of reach of existing systems. A crucial requirement is the time complexity of existing learning algorithms, as realistic data bases are usually fairly large.

Along these lines, the author has recently been investigating highly efficient induction algorithms [Wu 93a, Wu 93c]. The HCV algorithm described in Chapter 3 is a low-order polynomial induction algorithm. The target of finding low-order polynomial algorithms has been met by HCV. Different from both ID3-like and AQ11-like algorithms, HCV takes the little-known and inadequate extension matrix approach and improves it to be competitive with the decision tree based and the generalisation-specialisation based families of inductive algorithms.

HCV has been tested on several example sets including the famous MONK's problems outlined in Chapter 4. Experiments have shown empirically that the rules produced by HCV in variable-valued logic are more compact than the decision trees produced by the ID3-like algorithms in terms of the numbers of conjunctive rules and conjunctions. We can thus believe that the rules produced by HCV are fairly good.

We have also shown in Chapter 5 that the HCV algorithm has been incorporated in the intelligent learning data base system, *KEshell2*, to implement the whole process of knowledge acquisition from data bases. By the *whole process*, we mean an integrated data base and knowledge base system which can learn from data bases is able to 1) translate conventional data base information into a form suitable for use by the induction mechanisms, 2) use induction techniques to produce knowledge from data bases, and 3) interpret the knowledge produced to solve users' problems efficiently. Although there are still some limitations on the current *KEshell2* for putting it into large applications due to it being implemented on PC machines, all the functions and capacities shown in *KEshell2* have demonstrated that the target of building practical intelligent data base systems to extract knowledge from data bases is no longer difficult or elusive.

There are three directions for further work based on this thesis:

1. developing the capacities of the HCV algorithm to induct constructively, structuredly, incrementally and in noisy (including missing-attribute-value) environments;

2. taking the Prolog-based representation described in Appendix D as a conceptual design tool to design advanced data base systems where semantic information needs to be effectively processed; and

3. constructing practical intelligent data base systems by coupling the HCV algorithm, the Prolog-based representation, domain background knowledge, and existing data base and knowledge base technology.

Bibliography

[Bala *et al.* 92] J.W. Bala, R.S. Michalski, and J. Wnek, The Principal Axes Method for Constructive Induction, *Proceedings of the Ninth International Workshop on Machine Learning*, D. Sleeman and P. Edwards (Eds.), Morgan Kaufmann Publishers, 1992, 20–29.

[Barr *et al.* 81] A. Barr and E.A. Feigenbaum (Eds.), *The Handbook of Artificial Intelligence*, Vol. I, William Kaufmann, 1981.

[Bloedorn *et al.* 92] E. Bloedorn, R.S. Michalski, and J. Wnek, AQ17 – A Multistrategy Constructive Learning System, *Reports of Machine Learning and Inference Laboratory*, Center for Artificial Intelligence, George Mason University, USA, 1992.

[Blum 82] R.L. Blum, Discovery, Confirmation, and Incorporation of Causal Relationships from a Large Time-Oriented Clinical Data Base - The RX Project, *Computers and Biomedical Research*, **15**(1982), 2: 164–187.

[Boose *et al.* 88] J.H. Boose and B.R. Gaines (Eds.), *Knowledge Acquisition Tools for Expert Systems*, Academic Press, 1988.

[Borland 87] Borland International, Inc., *Turbo Prolog Toolbox User Guide and Reference Manual*, 4585 Scotts Valley Drive, CA 95066, USA, 1987.

[Boswell 90] R. Boswell, Manual for NewID Version 6.1, *TI/P2154/RAB/4/2.5*, The Turing Institute, Glasgow, 1990.

[Breiman *et al.* 84] L. Breiman, J.H. Friedman, R.A. Olshen and C.J. Stone, *Classification and Regression Trees*, Wadsworth, Belmont, California, 1984.

[Brodie 88] M.L. Brodie, Future Intelligent Information Systems: The Combination of Artificial Intelligence and Data Base Technology, *Readings in Artificial Intelligence and Databases*, 1988.

[Brownston *et al.* 85] L. Brownston, R. Farrell, E. Kant and N. Martin, *Programming Expert Systems in OPS5*, Addison-Wesley Publishing Company, Inc., 1985.

[Bundy *et al.* 85] A. Bundy, B. Silver and D. Plummer, An Analytical Comparison of Some Rule-Learning Programs, *Artificial Intelligence*, **27**(1985), 137–181.

[Cai *et al.* 91] Y. Cai, N. Cercone and J. Han, Learning in Relational Databases: An Attribute-Oriented Approach, *Computational Intelligence*, **7**(1991), 3: 119–132.

[Cattell *et al.* 91] R.G.G. Cattell *et al.*, Next Generation Database Systems, *Communications of the ACM* (special section), Vol. 34, No. 10, 1991.

[Carbonell 90] J.G. Carbonell (Ed.), *Machine Learning: Paradigms and Methods*, The MIT Press, 1990.

[Cestnik *et al.* 87] B. Cestnik, I. Kononenko, and I. Bratko, ASSISTANT 86: A Knowledge-Elicitation Tool for Sophisticated Users, *Progress in Machine Learning*, I. Bratko and N. Lavrac (Eds.), Wilmslow: Sigma Press, England, 1987.

[Clark 90] P. Clark, Machine Learning: Techniques and Recent Developments, *TIRM-90-041*, The Turing Institute, Glasgow, 1990.

[Clark *et al.* 89] P. Clark and T. Niblett, The CN2 Induction Algorithm, *Machine Learning*, **3**(1989), 261–283.

[Corlett 83] R.A. Corlett, Explaining Induced Decision Trees, *Expert Systems 83*, Churchill College, Cambridge, 14-16 December, 1983, 136–142.

[Dayhoff 90] J.E. Dayhoff, *Neural Network Architectures: An Introduction*, New York: Van Nostrand Reinhold, 1990.

[DeJong *et al.* 86] G. DeJong and R. Mooney, Explanation–Based Learning: An Alternative View, *Machine Learning*, **1**(1986), 145–176.

[Dowling & Gallier 84] W.F. Dowling and J.H. Gallier, Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae, *Journal of Logic Programming*, **1**(1984), 3: 267–284.

[Draxler 91] C. Draxler, Accessing Relational and NF^2 Databases Through Database Set Predicates, *Pre-conference Proceedings of the 3rd Annual Conference of Association for Logic Programming (UK)*, Edinburgh, 10th-12th, April, 1991, 86–92.

[Fayyad *et al.* 92] U.M. Fayyad and K.B. Irani, On the Handling of Continuous-Valued Attributes in Decision Tree Generation, *Machine Learning*, **8**(1992), 87–102.

[Feigenbaum 81] E.A. Feigenbaum, Expert Systems in the 1980s, *Infotech State of the Art Report on Machine Intelligence*, A. Bond (Ed.), Maidenhead: Pergamon–Infotech, 1981.

[Forgy 82] C.L. Forgy, Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, *Artificial Intelligence*, **19**(1982), 17–27.

[Frawley *et al.* 92] W.J. Frawley, G. Piatetsky-Shapiro, and C.J. Matheus, Knowledge Discovery in Databases: An Overview, *AI Magazine*, Fall 1992, 57–70.

[Ghallab 81] M. Ghallab, Decision Trees for Optimizing Pattern-Matching Algorithms in Production Systems, *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, 1981, 310–312.

[Gammerman *et al.* 91] A. Gammerman and A.R. Thatcher, Bayesian Diagnostic Probabilities without Assuming Independence of Symptoms, *Methods of Information in Medicine*, **30**(1991), 1: 15-22.

[Gams *et al.* 91] M. Gams, M. Drobnič and M. Petkovšek, Learning from Examples — A Uniform View, *International Journal of Man-Machine Studies*, **34**(1991), 49–68.

[Harmon *et al.* 85] P. Harmon and D. King, *Expert Systems,* John Wiley & Sons, Inc., 1985.

[Hong 85] J. Hong, AE1: An Extension Matrix Approximate Method for the General Covering Problem, *International Journal of Computer and Information Sciences*, **14**(1985), 6: 421–437.

[Hong 89a] J. Hong, A New Attribute–Based Learning Algorithm GS and a Comparison with Existing Algorithms, *Journal of Computer Science and Technology*, **4**(1989), 218–228.

[Hong 89b] J. Hong, Learning from Examples and a Multi-Purpose Learning System AE5, *Chinese Journal of Computers*, **12**(1989), 98–105.

[Hong *et al.* 86] J.R. Hong, I. Mozetic and R.S. Michalski, AQ15: Incremental Learning of Attribute–Based Descriptions from Examples, The Method and User's Guide, *Report ISG 85-5, UIUCDCS-F-86-949*, Dept. of Computer Science, Univ. of Illinois, Urbana, 1986.

[Hong *et al.* 87] J.R. Hong and C. Uhrik, The Extension Matrix Approach to Attribute-Based Learning, *Progress in Machine Learning*, I. Bratko and N. Lavrac (Eds.), Wilmslow: Sigma Press, England, 1987.

[Hunt *et al.* 66] E.B. Hunt, J. Marin and P.T. Stone, *Experiments in Induction*, Academic Press, New York, 1966.

[Jackson 90] P. Jackson, *Introduction to Expert Systems*, Second Edition, Addison-Wesley, 1990.

[Kazic *et al.* 90] T. Kazic, E. Lusk, R. Olson, R. Overbeek and S. Tuecke, Prototyping Databases in Prolog, *The Practice of Prolog*, L.S. Sterling (Ed.), The MIT Press, 1990, 1–29.

[Ke *et al.* 91] M. Ke and M. Ali, A Knowledge-Directed Induction Methodology for Intelligent Database Systems, *International Journal of Expert Systems*, **4**(1991), 1: 71–115.

[Kolodner 92] J.L. Kolodner, An Introduction to Case-Based Reasoning, *Artificial Intelligence Review*, **6**(1992): 3-34.

[Langley 89] P. Langley, Toward a Unified Science of Machine Learning, *Machine Learning*, **3**(1989), 253–259.

[Lee *et al.* 92] H.S. Lee and M.I. Schor, Match Algorithms for Generalized Rete Networks, *Artificial Intelligence*, **54**(1992), 249–274.

[Lenat 79] D.B. Lenat, On Automated Scientific Theory Formation: A Case Study Using the AM Program, *Machine Intelligence 9*, J. Hayes *et al.* (Eds.), New York: Halstead, 1979.

[Lenat 83] D.B. Lenat, EURISKO: A Program that Learns New Heuristics and Domain Concepts – The Nature of Heuristics III: Program Design and Results, *Artificial Intelligence*, **21**(1983), 61–98.

[Li 84] D. Li, *A Prolog Database System*, Letchworth: Research Studies Press, UK, 1984.

[Marcus 88] S. Marcus (Ed.), *Automating Knowledge Acquisition for Expert Systems*, Kluwer Academic Publishers, 1988.

[Matheus 89] C. Matheus, A Constructive Induction Framework, *Proceedings of the Sixth International Workshop on Machine Learning*, California: Morgan Kauffman, 1989.

[Matheus *et al.* 89] C. Matheus and L. Rendell, Constructive Induction on Decision Trees, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, California: Morgan Kauffman, 1989.

[McDonald 89] C. McDonald, Machine Learning: A Survey of Current Techniques, *Artificial Intelligence Review*, **3**(1989), 243–280.

[Mehra *et al.* 89] P. Mehra, L. Rendell and B. Wah, Principled Constructive Induction, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, California: Morgan Kauffman, 1989.

[Mellish 91] C. Mellish, The Description Identification Problem, *Artificial Intelligence*, **52**(1991), 151–167.

[Michalski 75] R.S. Michalski, Variable-Valued Logic and Its Applications to Pattern Recognition and Machine Learning, *Computer Science and Multiple-Valued Logic Theory and Applications*, D.C. Rine (Ed.), Amsterdam: North-Holland, 1975, 506–534.

[Michalski 86] R.S. Michalski, Understanding the Nature of Learning: Issues and Research Directions, *Machine Learning: An Artificial Intelligence Approach*, Vol. 2, R.S. Michalski, J.G. Carbonell and R.M. Mitchell (Eds.), CA: Kaufmann, 1986, 3–25.

[Michalski *et al.* 78] R.S. Michalski and J. Larson, Selection of Most Representative Training Examples and Incremental Generation of VL1 Hypothesis: the Underlying Methodology and Description of Programs ESEL and AQ11, *Tech. Report UIUCDCS-R-78-867*, Dept. of Computer Science, Univ. of Illinois at Champaign–Urbana, 1978.

[Michalski *et al.* 80] R.S. Michalski and R.L. Chilausky, Learning by Being Told and Learning from Examples: An Experimental Comparison of Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis, *International Journal of Policy Analysis and Information Systems*, **4**(1980), 125–161.

[Michalski *et al.* 86] R.S. Michalski, I. Mozetic, J. Hong and N. Lavrac, The Multi-Purpose Incremental Learning System AQ15 and Its Testing Application to Three Medical Domains, *Proceedings of AAAI 1986*, 1986, 1041–1045.

[Michie 87] D. Michie, Current Developments in Expert Systems, *Applications of Expert Systems*, J.R. Quinlan (Ed.), Reading: Addison–Wesley, 1987.

[Miranker 87] D.P. Miranker, TREAT: A Better Match Algorithm for AI Production Systems, *Proceedings of AAAI-87*, 1987, 42–47.

[Miranker *et al.* 90] D.P. Miranker, D.A. Brant, B. Iofaso and D. Gadbois, On the Performance of Lazy Matching in Production Systems, *Proceedings of AAAI-90*, 1990, 685–692.

[Mitchell 77] T. Mitchell, Version Spaces: A Candidate Elimination Approach to Rule Learning, *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Cambridge, Mass., 1977.

[Mitchell 78] T.M. Mitchell, Version Spaces: An Approach to Concept Learning, *Ph.D. Thesis*, Stanford University, 1978.

[Mitchell *et al.* 86] T. Mitchell, R. Keller and S. Kedar-Cabelli, Explanation – Based Generalization: A Unifying View, *Machine Learning*, **1**(1986), 47–80.

[Mowforth 86] P. Mowforth, Some Applications with Inductive Expert Systems Shells, *TIOP-86-002*, The Turing Institute, Glasgow, 1986.

[Muggleton 86] S.H. Muggleton, Inductive Acquisition of Expert Knowledge, *Ph.D. Thesis*, University of Edinburgh, 1986.

[Muggleton 88] S. Muggleton, A Strategy for Constructing New Predicates in First Order Logic, *Proceedings of the European Working Session on Learning*, London: Pitman, 1988.

[Muggleton 90] S. Muggleton, Inductive Logic Programming, *Proceedings of the Workshop on Algorithmic Learning Theory*, Japanese Society for Artificial Intelligence, 1990.

[Nieme *et al.* 91] T. Nieme and K. Järvelin, Prolog-Based Meta Rules for Relational Database Representation and Manipulation, *IEEE Transactions on Software Engineering*, **17**(1991), 8: 762–788.

[Núñez 91] M. Núñez, The Use of Background Knowledge in Decision Tree Induction, *Machine Learning*, **6**(1991), 231–250.

[O'Rorke 82] P. O'Rorke, A Comparative Study of Inductive Learning Systems AQ11P and ID3 Using a Chess End-Game Test Problem, *ISG 82-2*, Computer Science Department, University of Illinois at Urbana-Champaign, 1982.

[Paterson *et al.* 82] A. Paterson and T. Niblett, *ACLS Manual*, Version 1, The Turing Institute, Glasgow, 1982.

[Piatetsky-Shapiro *et al.* 91] , G. Piatetsky-Shapiro and W.J. Frawley (Eds.), *Knowledge Discovery in Databases*, The AAAI Press/MIT Press, USA, 1991.

[Quinlan 79] J.R. Quinlan, Discovering Rules by Induction from Large Collections of Examples, *Introductory Readings in Expert Systems*, D. Michie (Ed.), Gordon and Breach, London, 1979, 33–46.

[Quinlan 86a] J.R. Quinlan, Learning from Noisy Data, *Machine Learning*, Vol. 2, J. Carbonell and T. Mitchell (Eds.), Tioga, Palo Alto, USA, 1986.

[Quinlan 86b] J.R. Quinlan, Induction of Decision Trees, *Machine Learning*, **1**(1986), 81–106.

[Quinlan 87] J.R. Quinlan, Generating Production Rules from Decision Trees, *Proceedings of International Joint Conference on Artificial Intelligence*, J. McDermott (Ed.), Morgan Kaufmann Publishers, Inc., 1987, 304–307.

[Quinlan 88a] J.R. Quinlan, Induction, Knowledge and Expert Systems, *Artificial Intelligence Developments and Applications*, J.S. Gero and R. Stanton (Eds.), North–Holland: Elsevier Science Publishers B. V., 1988, 253–271.

[Quinlan 88b] J.R. Quinlan, Decision Trees and Multi-Valued Attributes, *Machine Intelligence 11: Logic and the Acquisition of Knowledge*, J.E. Hayes, D. Michie and J. Richards (Eds.), Clarendon Press, Oxford, 1988, 305–318.

[Quinlan 89a] J.R. Quinlan, Simplifying Decision Trees, *Knowledge Acquisition for Knowledge Based Systems*, B. Gaines and J. Boose (Eds.), Vol. 1, Academic Press, 1989.

[Quinlan 89b] J.R. Quinlan, Requirements for Knowledge Discovery in Databases, *Proceedings of IJCAI-89 Workshop on Knowledge Discovery in Databases*, Detroit, USA, 1989, xiv.

[Quinlan 91] J.R. Quinlan, Knowledge Acquisition from Structured Data: Using Determinate Literals to Assist Search, *IEEE Expert*, **6**(1991), 6: 32–37.

[Quinlan 92] J.R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers, 1992.

[Quinlan *et al.* 86] J.R. Quinlan, K.A. Horn and L. Lazarus, Inductive Knowledge Acquisition: A Case Study, *Proceedings of the Second Australian Conference on the Applications of Expert Systems*, New South Wales Institute of Technology, Sydney, 1986, 183–204.

[Quinlan *et al.* 87] J.R. Quinlan, P.J. Compton, K.A. Horn and L. Lazarus, Inductive Knowledge Acquisition: A Case Study, *Applications of Expert Systems*, Addison-Wesley, 1987.

[Schlimmer *et al.* 86] J.C. Schlimmer and D. Fisher, A Case Study of Incremental Concept Induction, *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, Morgan Kaufmann, USA, 1986, 496–501.

[Shapiro 87] A.D. Shapiro, *Structured Induction in Expert Systems*, Turing Institute Press in association with Addison-Wesley, Workingham, UK, 1987.

[Su *et al.* 80] S.Y.W. Su and D.H. Lo, A Semantic Association Model for Conceptual Database Design, *Entity-Relationship Approach to System Analysis and Design*, P.P. Chen (Ed.), North-Holand Publishing Company, 1980.

[Tambe & Newell 88] M. Tambe and A. Newell, Some Chunks Are Expensive, *Proceedings of the Fifth International Conference on Machine Learning*, 1988, 451–458.

[Thornton 91] C.J. Thornton, The Complexity of Constructive Learning, Dept. of Artificial Intelligence, Univ. of Edinburgh, 1991.

[Thrun *et al.* 91] S.B. Thrun, *et al.*, The MONK's Problems – A Performance Comparison of Different Learning Algorithms, *CMU-CS-91-197*, School of Computer Science, Carnegie Mellon University, 1991.

[Utgoff 89] P.E. Utgoff, Incremental Induction of Decision Trees, *Machine Learning*, **4**(1989), 161–186.

[Valiant 84] L.G. Valiant, A Theory of the Learnable, *Communications of the ACM*, **27**(1984), 11: 1134–1142.

[Waters 85] R.C. Waters, KBEmacs: A Step toward the Programmer's Apprentice, *Technical Report 753*, AI Laboratory, MIT, 1985.

[Winston 70] P. Winston, Learning Structural Description from Examples, *Ph.D Thesis*, Mass.: MIT AI Lab., 1970.

[Wirth *et al.* 88] J. Wirth and J. Catlett, Experiments on the Costs and Benefits of Windowing in ID3, *Proceedings of the Fifth International Conference on Machine Learning*, J. Laird (Ed.), Morgan Kauffman Publishers Inc., 1988.

[Wu 90] X. Wu, *Constructing Expert Systems*, Hefei: Press of the University of Science and Technology of China, China, 1990.

[Wu 91] X. Wu, *KEshell*: A "Rule Skeleton + Rule Body" Based Knowledge Engineering Shell, *Applications of Artificial Intelligence IX* (Proceedings of SPIE 1468), M.M. Trivedi (Ed.), SPIE Society, Bellingham, USA, 1991, 632–639.

[Wu 92a] X. Wu, Optimization Problems in Extension Matrixes, *Science in China*, Series A, Chinese edition: **35**(1992), 2: 200–207; English edition, **35**(1992), 3: 363–373.

[Wu 92b] X. Wu, A Frame Based Architecture for Information Integration in CIMS, *Journal of Computer Science and Technology*, **7**(1992), 4: 328–332.

[Wu 92c] X. Wu, *KEshell2*: An Intelligent Learning Data Base System, *Research and Development in Expert Systems IX*, M.A. Bramer and R.W. Milne (Eds.), Cambridge University Press, 1992, 253–272.

[Wu 92d] X. Wu, HCV User's Manual (Release 1.0 June 1992), *DAI Technical Paper No. 9*, Department of Artificial Intelligence, University of Edinburgh, 1992.

[Wu 93a] X. Wu, The HCV Induction Algorithm, *Proceedings of the 21st ACM Computer Science Conference*, S.C. Kwasny and J.F. Buck (Eds.), ACM Press, 1993, 168–175.

[Wu 93b] X. Wu, Inductive Learning: Algorithms and Frontiers, *Artificial Intelligence Review*, **7**(1993), 2: 93–108.

[Wu 93c] X. Wu, TS: A Test-Split Algorithm for Inductive Learning, *Adaptive and Learning Systems II*, F.A. Sadjadi (Ed.), SPIE Society, Bellingham, USA, 1993.

[Wu 93d] X. Wu, *LFA*: A Linear Forward-chaining Algorithm for AI Production Systems, to be published in a forthcoming issue of *Expert Systems: The International Journal of Knowledge Engineering*, **10**(1993).

[Wu *et al.* 87] X. Wu and D. Zhang, An Expert System for Multi-Objective Decision-Making—DECISION, *Proceedings of International Symposium on Fuzzy Systems and Knowledge Engineering*, 1987, 216–223.

[Wu *et al.* 88] X. Wu and Y. Zou, *Expert Systems Technology*, Beijing: The National Electronics Industry Press, China, 1988.

[Zaniolo 86] C. Zaniolo, Prolog: A Database Query Language for All Seasons, *Proceedings of the 1st International Workshop on Expert Database Systems*, L. Kerschberg (Ed.), 1986, 219–232.

Appendix A

Converting a decision tree to production rules

The decision tree generated by ID3 for the example set in Table A-1 is shown in Figure A-1, which is equivalent to the following decision rules:

if *OUTLOOK=overcast* then *Play*;

if *OUTLOOK=rain* & *WINDY=true* then *Don't Play*;

if *OUTLOOK=rain* & *WINDY=false* then *Play*;

if *OUTLOOK=sunny* & $TEMPERATURE$ = *hot* then *Don't Play*;

if *OUTLOOK=sunny* & $TEMPERATURE$ = *cool* then *Don't Play*;

if *OUTLOOK=sunny* & $TEMPERATURE$ = *mild* & *HUMIDITY=normal* then *Play*; and

if *OUTLOOK=sunny* & $TEMPERATURE$ = *mild* & *HUMIDITY=high* then *Don't Play*.

Here, no conditions in the conjunctive rules can be dropped.

Table A–1: Cases of *Play* and *Don't Play* (adapted from [Quinlan 86b])

ORDER	*OUTLOOK*	*TEMPERATURE*	*HUMIDITY*	*WINDY*	**DECISION**
1	rain	hot	high	true	*Don't Play*
2	rain	cool	normal	true	*Don't Play*
3	overcast	mild	high	true	*Play*
4	overcast	mild	normal	false	*Play*
5	rain	hot	high	false	*Play*
6	overcast	cool	normal	true	*Play*
7	sunny	hot	normal	true	*Don't Play*
8	sunny	mild	high	true	*Don't Play*
9	sunny	mild	normal	false	*Play*
10	rain	cool	normal	false	*Play*
11	rain	hot	high	false	*Play*
12	sunny	hot	high	false	*Don't Play*
13	sunny	cool	normal	false	*Don't Play*
14	rain	mild	normal	true	*Don't Play*

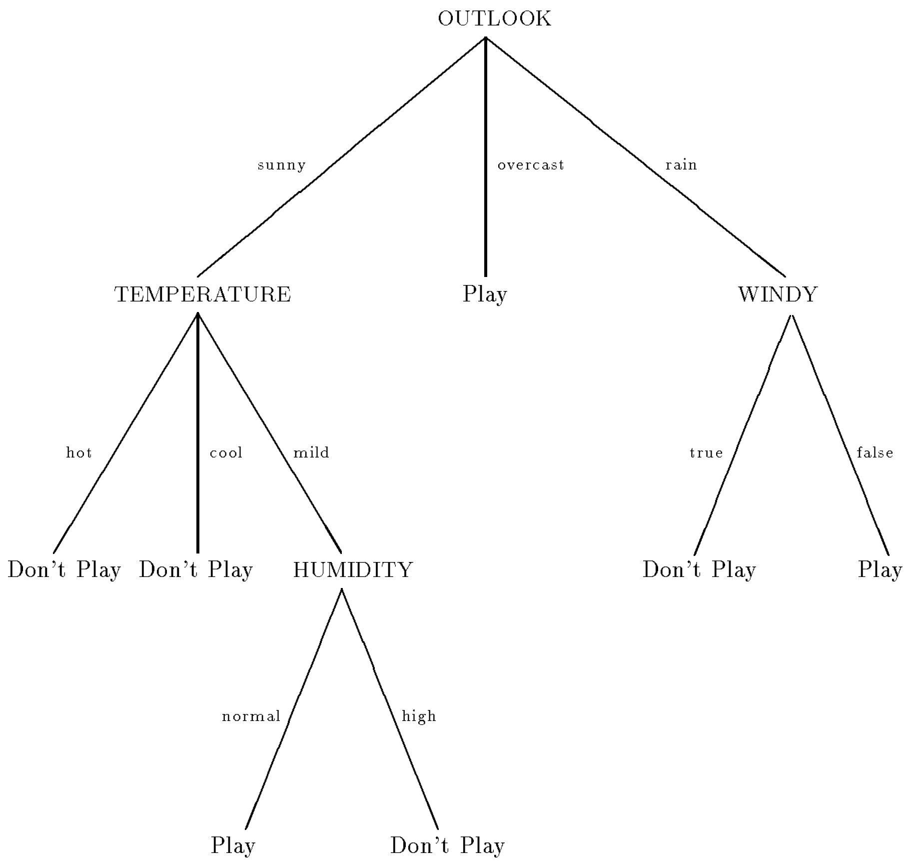

Figure A–1: A decision tree (by ID3) for Table A-1

Appendix B

Three decision trees for the same example set

For the example set in Table B-1, the decision trees produced by ID3 and the new *gain ratio* heuristic mentioned in Section 2.4.8 are shown in Figure B-1 and Figure B-2 respectively.

The rules correspond to the decision tree in Figure B-1 (the conditions in the **bold** type style can be dropped):

	X2=b		
$\vee$	X2=a & X1=0		X2=a & X1=1
$\vee$	X2=c & X3=a	$\vee$	X2=c & X3=c
$\vee$	**X2=c** & X3=b & X1=0	$\vee$	**X2=c** & X3=b & X1=1
$\rightarrow$		$\rightarrow$	
	The *T* class.		The *F* class.

The decision tree in Figure A-2 is equivalent to the following decision rules:

	X1=1 & X2=b		
$\vee$	X1=0 & X3=a		X1=1 & X2=a
$\vee$	X1=0 & X3=b	$\vee$	X1=1 & X2=b
$\vee$	X1=0 & X3=c & X2=a	$\vee$	**X1=0** & X3=c & X2=c
$\rightarrow$		$\rightarrow$	
	The *T* class.		The *F* class.

Table B–1: Cases of T and F

ORDER	*X1*	*X2*	*X3*	*X4*	**CLASS**
1	1	a	a	1	F
2	1	a	b	1	F
3	1	a	c	1	F
4	1	a	a	0	F
5	1	b	c	1	T
6	0	b	b	0	T
7	0	a	c	1	T
8	1	b	a	0	T
9	1	b	a	1	T
10	1	c	c	0	F
11	1	c	b	1	F
12	0	c	b	0	T
13	0	a	a	0	T
14	0	c	c	1	F
15	0	c	a	0	T
16	1	a	b	0	F
17	0	a	a	1	T
18	0	b	a	1	T

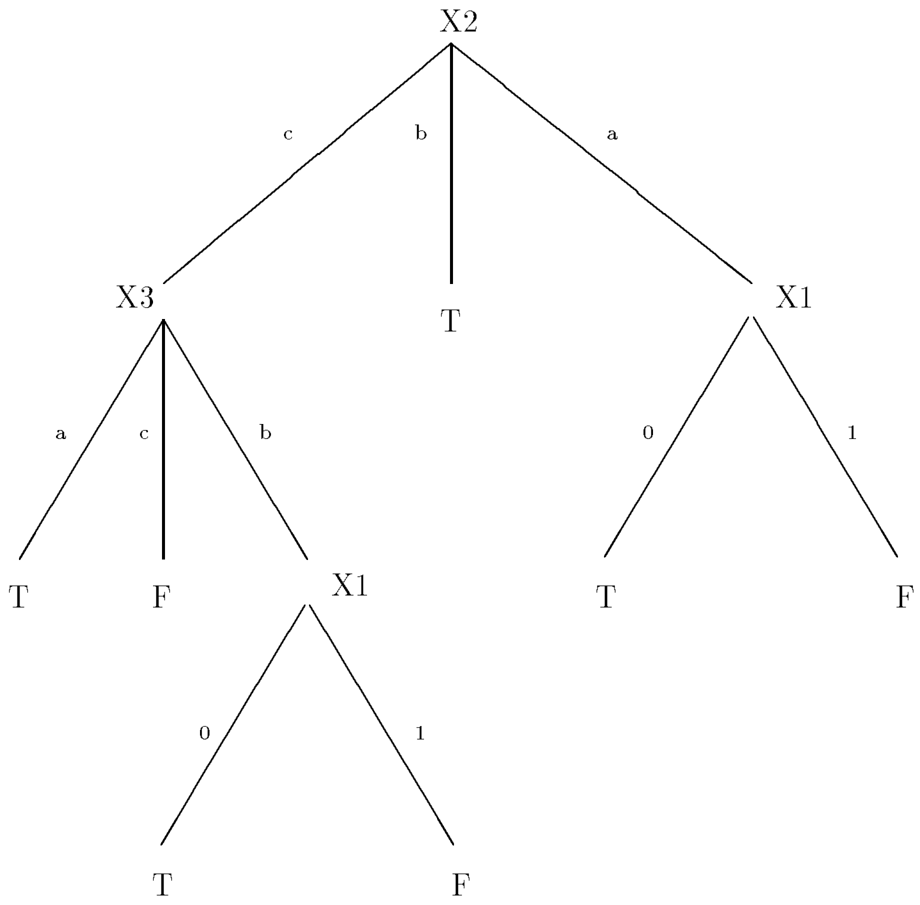

Figure B–1: A decision tree (by ID3) for Table B-1

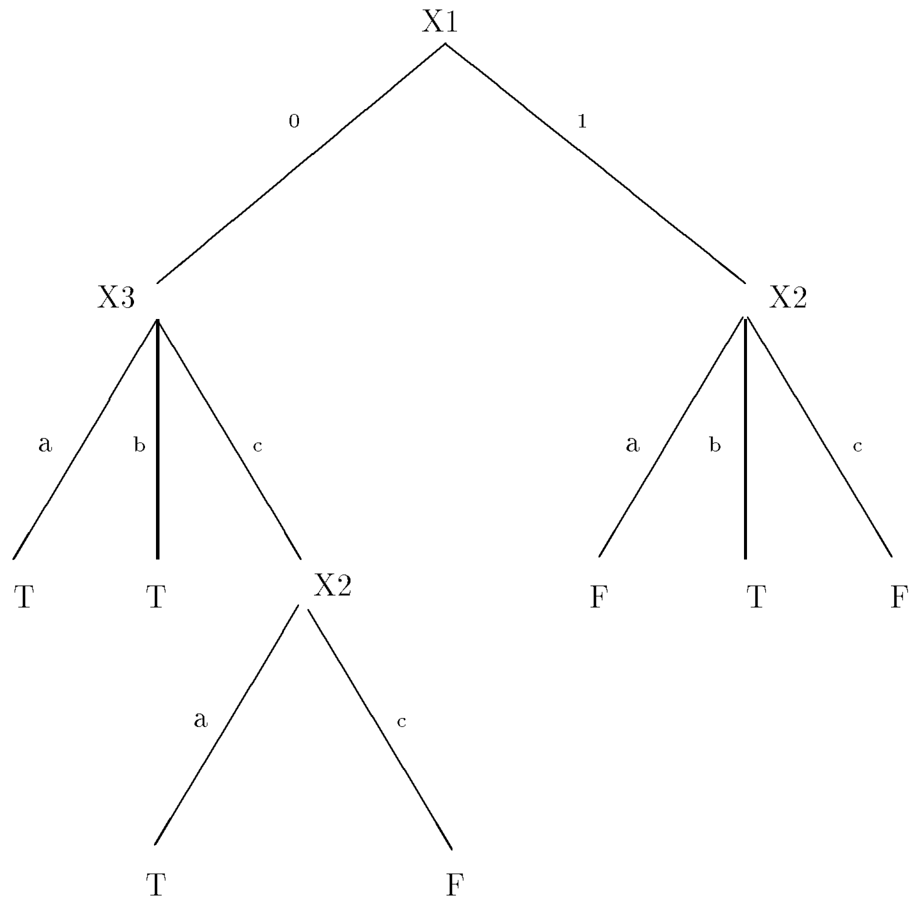

Figure B–2: A decision tree (by the *gain ratio* heuristic) for Table B-1

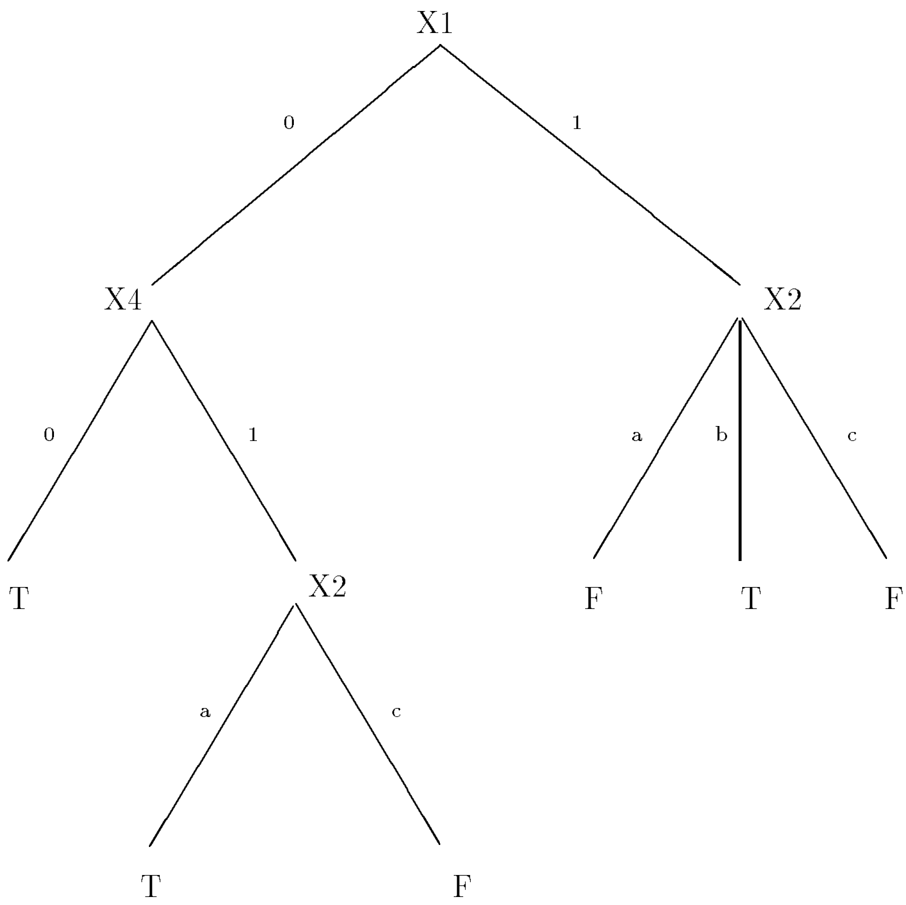

Figure B–3: A briefer decision tree (by hand) for Table B-1

Meanwhile, Figure B-3 gives another tree made by hand, which correctly classifies the same example set and corresponds to the following decision rules.

	X1=1 & X2=b		X1=1 & X2=a
$\vee$	X1=0 & X4=0	$\vee$	X1=1 & X2=c
$\vee$	X1=0 & X4=1 & X2=a	$\vee$	**X1=0** & X4=1 & X2=c
$\rightarrow$		$\rightarrow$	
	The *T* class.		The *F* class.

Figure B-3 is clearly smaller than both Figure B-1 and Figure B-2 in terms of the numbers of leaves and nodes (including leaves) in decision trees or the numbers of conjunctive rules and conjunctions in the equivalent rules to the decision trees.

Appendix C

Results produced by HCV on the MONK's problems

C.1 The M1 problem

124 examples in the training file (6 attributes).

Rules for the Non-M1 class:

The 1st conjunctive rule:

[body_shape=[octagon,square]] $\wedge$

[jacket_color=[blue,green,yellow]] $\wedge$

[head_shape=[round]]

$\rightarrow$ the Non-M1 class.

(Positive examples covered: 31)

The 2nd conjunctive rule:

[head_shape=[square]] $\wedge$

[jacket_color=[blue,yellow,green]] $\wedge$

[body_shape=[octagon,round]]

$\rightarrow$ the Non-M1 class.

(Positive examples covered: 20)

The 3rd conjunctive rule:

[head_shape=[octagon]] $\wedge$

[jacket_color=[blue,green,yellow]] $\wedge$

[body_shape=[square,round]]

$\rightarrow$ the Non-M1 class.

(Positive examples covered: 11)

Net cpu time for computing this class: 51.97 seconds.

Rules for the M1 class:

The 4th conjunctive rule:

[body_shape=[round]] $\wedge$

[head_shape=[round]]

$\rightarrow$ the M1 class.

(Positive examples covered: 10)

The 5th conjunctive rule:

[jacket_color=[red]]

$\rightarrow$ the M1 class.

(Positive examples covered: 28)

The 6th conjunctive rule:

[head_shape=[square]] $\wedge$

[body_shape=[square]]

$\rightarrow$ the M1 class.

(Positive examples covered: 12)

The 7th conjunctive rule:

[head_shape=[octagon]] $\wedge$

[body_shape=[octagon]]

→ the M1 class.

(Positive examples covered: 12)

DEFAULT → the Non-M1 class.

(Positive examples covered: 62)

There are 432 test examples in the test example file.

432 examples have been correctly classified; while 0 were misclassified.

The accuracy of the rules produced by HCV on the test examples is 100.0%.

C.2 The M2 problem

169 examples in the training file (6 attributes).

Rules for the M2 class:

The 1st conjunctive rule:

[has_tie=[no]] $\wedge$

[jacket_color=[yellow,blue]] $\wedge$

[body_shape=[round]] $\wedge$

[head_shape=[round]] $\wedge$

[holding=[balloon,flag]]

→ the M2 class.

(Positive examples covered: 2)

The 2nd conjunctive rule:

[jacket_color=[blue,green,yellow]] $\wedge$

[has_tie=[no]] $\wedge$

[is_smiling=[yes]] $\wedge$

[holding=[balloon,flag]] $\wedge$

[head_shape=[round]]

$\rightarrow$ the M2 class.

(Positive examples covered: 9)

The 3rd conjunctive rule:

[body_shape=[octagon,square]] $\wedge$

[is_smiling=[no]] $\wedge$

[holding=[balloon,flag]] $\wedge$

[jacket_color=[yellow,blue,green]] $\wedge$

[has_tie=[yes]] $\wedge$

[head_shape=[round]]

$\rightarrow$ the M2 class.

(Positive examples covered: 5)

The 4th conjunctive rule:

[body_shape=[octagon,square]] $\wedge$

[is_smiling=[no]] $\wedge$

[has_tie=[no]] $\wedge$

[jacket_color=[red]]

$\rightarrow$ the M2 class.

(Positive examples covered: 3)

The 5th conjunctive rule:

[is_smiling=[no]] $\wedge$

[jacket_color=[yellow,green]] $\wedge$

[has_tie=[no]] $\wedge$

[holding=[sword]] $\wedge$

[body_shape=[round,octagon]]

$\rightarrow$ the M2 class.

(Positive examples covered: 4)

The 6th conjunctive rule:

[holding=[balloon,flag]] $\wedge$

[jacket_color=[blue,green,yellow]] $\wedge$

[has_tie=[no]] $\wedge$

[is_smiling=[yes]] $\wedge$

[body_shape=[round]]

$\rightarrow$ the M2 class.

(Positive examples covered: 8)

The 7th conjunctive rule:

[holding=[flag]] $\wedge$

[is_smiling=[no]] $\wedge$

[has_tie=[yes]] $\wedge$

[jacket_color=[red,yellow]] $\wedge$

[body_shape=[square,round]] $\wedge$

[head_shape=[square]]

$\rightarrow$ the M2 class.

(Positive examples covered: 2)

The 8th conjunctive rule:

[head_shape=[octagon,square]] $\wedge$

[is_smiling=[no]] $\wedge$

[has_tie=[no]] $\wedge$

[jacket_color=[red]]

$\rightarrow$ the M2 class.

(Positive examples covered: 2)

The 9th conjunctive rule:

[head_shape=[square]] $\wedge$

[body_shape=[octagon,square]] $\wedge$

[jacket_color=[yellow,blue]] $\wedge$

[is_smiling=[yes]] $\wedge$

[holding=[balloon,sword]]

$\rightarrow$ the M2 class.

(Positive examples covered: 4)

The 10th conjunctive rule:

[head_shape=[octagon,square]] $\wedge$

[holding=[balloon,flag]] $\wedge$

[jacket_color=[yellow,green]] $\wedge$

[has_tie=[yes]] $\wedge$

[is_smiling=[yes]]

$\rightarrow$ the M2 class.

(Positive examples covered: 6)

The 11th conjunctive rule:

[head_shape=[octagon,square]] $\wedge$

[body_shape=[octagon,square]] $\wedge$

[is_smiling=[no]] $\wedge$

[jacket_color=[green,yellow,blue]] $\wedge$

[has_tie=[yes]] $\wedge$

[holding=[sword]]

$\rightarrow$ the M2 class.

(Positive examples covered: 4)

The 12th conjunctive rule:

[head_shape=[square]] $\wedge$

[body_shape=[octagon,square]] $\wedge$

[holding=[balloon]] $\wedge$

[jacket_color=[red]]

$\rightarrow$ the M2 class.

(Positive examples covered: 2)

The 13th conjunctive rule:

[head_shape=[octagon,square]] $\wedge$

[body_shape=[square,octagon]] $\wedge$

[has_tie=[no]] $\wedge$

[holding=[sword]] $\wedge$

[is_smiling=[yes]] $\wedge$

[jacket_color=[yellow,green]]

$\rightarrow$ the M2 class.

(Positive examples covered: 4)

The 14th conjunctive rule:

[head_shape=[octagon,square]] $\wedge$

[body_shape=[square,octagon]] $\wedge$

[is_smiling=[no]] $\wedge$

[holding=[balloon,flag]] $\wedge$

[jacket_color=[red]]

$\rightarrow$ the M2 class.

(Positive examples covered: 5)

The 15th conjunctive rule:

[has_tie=[no]] $\wedge$

[holding=[sword]] $\wedge$

[jacket_color=[blue]] $\wedge$

[body_shape=[round]]

$\rightarrow$ the M2 class.

(Positive examples covered: 1)

The 16th conjunctive rule:

[has_tie=[yes]] $\wedge$

[body_shape=[round]] $\wedge$

[jacket_color=[yellow]] $\wedge$

[is_smiling=[no]]

$\rightarrow$ the M2 class.

(Positive examples covered: 2)

The 17th conjunctive rule:

[head_shape=[octagon]] $\wedge$

[body_shape=[square]] $\wedge$

[holding=[flag]] $\wedge$

[jacket_color=[red]]

$\rightarrow$ the M2 class.

(Positive examples covered: 1)

Rules for the Non-M2 class:

The 18th conjunctive rule:

[is_smiling=[yes]] $\wedge$

[body_shape=[round]] $\wedge$

[head_shape=[round]]

$\rightarrow$ the Non-M2 class.

(Positive examples covered: 9)

The 19th conjunctive rule:

[jacket_color=[green,red]] $\wedge$

[body_shape=[round]] $\wedge$

[head_shape=[round]]

$\rightarrow$ the Non-M2 class.

(Positive examples covered: 4)

The 20th conjunctive rule:

[has_tie=[yes]] $\wedge$

[body_shape=[octagon,round]] $\wedge$

[holding=[sword,flag]] $\wedge$

[head_shape=[round]]

$\rightarrow$ the Non-M2 class.

(Positive examples covered: 6)

The 21st conjunctive rule:

[jacket_color=[blue,red]] $\wedge$

[holding=[flag,sword]] $\wedge$

[is_smiling=[yes]] $\wedge$

[head_shape=[round]]

$\rightarrow$ the Non-M2 class.

(Positive examples covered: 4)

The 22nd conjunctive rule:

[has_tie=[yes]] $\wedge$

[is_smiling=[yes]] $\wedge$

[head_shape=[round]]

$\rightarrow$ the Non-M2 class.

(Positive examples covered: 5)

The 23rd conjunctive rule:

[jacket_color=[red]] $\wedge$

[is_smiling=[yes]] $\wedge$

[body_shape=[round,square]] $\wedge$

[head_shape=[square,round]]

$\rightarrow$ the Non-M2 class.

(Positive examples covered: 4)

The 24th conjunctive rule:

[holding=[sword]] $\wedge$

[jacket_color=[blue,yellow]] $\wedge$

[head_shape=[round]]

→ the Non-M2 class.

(Positive examples covered: 3)

The 25th conjunctive rule:

[has_tie=[yes]] $\wedge$

[jacket_color=[red]] $\wedge$

[head_shape=[round]]

→ the Non-M2 class.

(Positive examples covered: 1)

The 26th conjunctive rule:

[body_shape=[octagon,square]] $\wedge$

[is_smiling=[no]] $\wedge$

[has_tie=[no]] $\wedge$

[jacket_color=[yellow,blue,green]] $\wedge$

[holding=[flag,balloon]]

→ the Non-M2 class.

(Positive examples covered: 15)

The 27th conjunctive rule:

[holding=[sword]] $\wedge$

[is_smiling=[yes]] $\wedge$

[head_shape=[round]]

→ the Non-M2 class.

(Positive examples covered: 1)

The 28th conjunctive rule:

[holding=[sword]] $\wedge$

[is_smiling=[yes]] ∧

[jacket_color=[green,yellow]] ∧

[body_shape=[square,round]] ∧

[head_shape=[square]]

→ the Non-M2 class.

(Positive examples covered: 3)

The 29th conjunctive rule:

[has_tie=[yes]] ∧

[jacket_color=[green,red,blue]] ∧

[body_shape=[round]]

→ the Non-M2 class.

(Positive examples covered: 10)

The 30th conjunctive rule:

[head_shape=[octagon,square]] ∧

[is_smiling=[no]] ∧

[holding=[balloon,flag]] ∧

[has_tie=[no]] ∧

[jacket_color=[green,yellow,blue]]

→ the Non-M2 class.

(Positive examples covered: 6)

The 31st conjunctive rule:

[head_shape=[square]] ∧

[body_shape=[square]] ∧

[jacket_color=[yellow,green]] ∧

[has_tie=[no]]

$\rightarrow$ the Non-M2 class.

(Positive examples covered: 3)

The 32nd conjunctive rule:

[jacket_color=[red]] $\wedge$

[holding=[sword]] $\wedge$

[has_tie=[yes]]

$\rightarrow$ the Non-M2 class.

(Positive examples covered: 5)

The 33rd conjunctive rule:

[head_shape=[octagon,square]] $\wedge$

[body_shape=[octagon,square]] $\wedge$

[is_smiling=[no]] $\wedge$

[has_tie=[no]] $\wedge$

[jacket_color=[green,yellow,blue]]

$\rightarrow$ the Non-M2 class.

(Positive examples covered: 4)

The 34th conjunctive rule:

[head_shape=[octagon,square]] $\wedge$

[body_shape=[octagon,square]] $\wedge$

[is_smiling=[no]] $\wedge$

[jacket_color=[yellow,green]] $\wedge$

[holding=[balloon,flag]]

$\rightarrow$ the Non-M2 class.

(Positive examples covered: 8)

The 35th conjunctive rule:

[head_shape=[octagon,square]] $\wedge$

[body_shape=[octagon]] $\wedge$

[jacket_color=[yellow,blue,red]] $\wedge$

[holding=[flag,sword]] $\wedge$

[is_smiling=[yes]]

$\rightarrow$ the Non-M2 class.

(Positive examples covered: 4)

The 36th conjunctive rule:

[head_shape=[octagon,square]] $\wedge$

[body_shape=[square,octagon]] $\wedge$

[has_tie=[no]] $\wedge$

[holding=[balloon,flag]] $\wedge$

[jacket_color=[green]]

$\rightarrow$ the Non-M2 class.

(Positive examples covered: 3)

The 37th conjunctive rule:

[jacket_color=[red]] $\wedge$

[is_smiling=[yes]] $\wedge$

[body_shape=[round]]

$\rightarrow$ the Non-M2 class.

(Positive examples covered: 2)

The 38th conjunctive rule:

[head_shape=[octagon]] ∧

[jacket_color=[blue,red]] ∧

[holding=[balloon,sword]] ∧

[is_smiling=[yes]]

→ the Non-M2 class.

(Positive examples covered: 3)

The 39th conjunctive rule:

[has_tie=[yes]] ∧

[holding=[sword]] ∧

[is_smiling=[yes]]

→ the Non-M2 class.

(Positive examples covered: 2)

DEFAULT → the Non-M2 class.

(Positive examples covered: 105)

There are 432 test examples in the test example file.

351 examples have been correctly classified; while 81 were misclassified.

The accuracy of the rules produced by HCV on the test examples is 81.25%.

C.3 The M3 problem

122 examples in the training file (6 attributes).

Rules for the M3 class:

The 1st conjunctive rule:

[jacket_color=[yellow,red]] $\wedge$

[body_shape=[square,round]] $\wedge$

[head_shape=[octagon,round]]

$\rightarrow$ the M3 class.

(Positive examples covered: 31)

The 2nd conjunctive rule:

[holding=[balloon]] $\wedge$

[jacket_color=[green]] $\wedge$

[body_shape=[square,round]] $\wedge$

[head_shape=[octagon,round]]

$\rightarrow$ the M3 class.

(Positive examples covered: 4)

The 3rd conjunctive rule:

[body_shape=[square]] $\wedge$

[holding=[sword]] $\wedge$

[jacket_color=[yellow,green]]

$\rightarrow$ the M3 class.

(Positive examples covered: 5)

The 4th conjunctive rule:

[head_shape=[octagon,square]] $\wedge$

[jacket_color=[green,yellow,red]] $\wedge$

[body_shape=[round]]

$\rightarrow$ the M3 class.

(Positive examples covered: 10)

The 5th conjunctive rule:

[jacket_color=[red]] $\wedge$

[body_shape=[square]]

$\rightarrow$ the M3 class.

(Positive examples covered: 3)

The 6th conjunctive rule:

[jacket_color=[yellow]] $\wedge$

[body_shape=[square]] $\wedge$

[has_tie=[no]]

$\rightarrow$ the M3 class.

(Positive examples covered: 1)

The 7th conjunctive rule:

[head_shape=[octagon,square]] $\wedge$

[jacket_color=[green]] $\wedge$

[holding=[flag]] $\wedge$

[body_shape=[square]]

$\rightarrow$ the M3 class.

(Positive examples covered: 3)

The 8th conjunctive rule:

[head_shape=[octagon,square]] ∧

[body_shape=[octagon]] ∧

[holding=[sword]] ∧

[is_smiling=[yes]]

→ the M3 class.

(Positive examples covered: 3)

Rules for the Non-M3 class:

The 9th conjunctive rule:

[jacket_color=[blue,green]] ∧

[holding=[flag,sword]] ∧

[body_shape=[octagon,round]] ∧

[head_shape=[round]]

→ the Non-M3 class.

(Positive examples covered: 12)

The 10th conjunctive rule:

[body_shape=[octagon,square]] ∧

[holding=[balloon,flag]] ∧

[jacket_color=[blue,green]] ∧

[has_tie=[yes]] ∧

[head_shape=[round]]

→ the Non-M3 class.

(Positive examples covered: 4)

The 11th conjunctive rule:

[body_shape=[octagon,square]] ∧

[jacket_color=[red,yellow,blue]] $\wedge$

[holding=[sword]] $\wedge$

[head_shape=[round]]

$\rightarrow$ the Non-M3 class.

(Positive examples covered: 5)

The 12th conjunctive rule:

[holding=[balloon]] $\wedge$

[is_smiling=[yes]] $\wedge$

[body_shape=[round,octagon]] $\wedge$

[has_tie=[yes]]

$\rightarrow$ the Non-M3 class.

(Positive examples covered: 6)

The 13th conjunctive rule:

[jacket_color=[blue]] $\wedge$

[head_shape=[square]]

$\rightarrow$ the Non-M3 class.

(Positive examples covered: 9)

The 14th conjunctive rule:

[head_shape=[square]] $\wedge$

[jacket_color=[green,yellow]] $\wedge$

[has_tie=[yes]] $\wedge$

[is_smiling=[yes]] $\wedge$

[holding=[balloon,flag]]

$\rightarrow$ the Non-M3 class.

(Positive examples covered: 3)

The 15th conjunctive rule:

[head_shape=[octagon,square]] $\wedge$

[body_shape=[octagon,square]] $\wedge$

[holding=[flag,balloon]] $\wedge$

[jacket_color=[blue,green]] $\wedge$

[is_smiling=[yes]] $\wedge$

[has_tie=[no]]

$\rightarrow$ the Non-M3 class.

(Positive examples covered: 3)

The 16th conjunctive rule:

[body_shape=[octagon]] $\wedge$

[jacket_color=[yellow,red]]

$\rightarrow$ the Non-M3 class.

(Positive examples covered: 11)

The 17th conjunctive rule:

[body_shape=[octagon]] $\wedge$

[holding=[balloon,flag]]

$\rightarrow$ the Non-M3 class.

(Positive examples covered: 4)

The 18th conjunctive rule:

[jacket_color=[blue]] $\wedge$

[body_shape=[square,round]]

$\rightarrow$ the Non-M3 class.

(Positive examples covered: 5)

DEFAULT → the Non-M3 class.

(Positive examples covered: 62)

There are 432 test examples in the test example file.

390 examples have been correctly classified; while 42 were misclassified.

The accuracy of the rules produced by HCV on the test examples

is 90.277777777777779%.

Appendix D

A representation for integrating knowledge and data

D.1 Introduction

Although the history of data base systems research is one of exceptional productivity and startling economic impact, many advanced applications have revealed deficiencies of the conventional data base management systems (DBMSs) in representing and processing complex objects and knowledge [Cattell *et al.* 91]. Object-oriented approaches are currently very popular in processing structurally complex objects while deductive data bases or logic data bases have been proposed as a solution to those applications where both knowledge and data models are needed. However, it has been characteristic of the current deductive data bases that only actual data is represented explicitly in logic while the data schema is implicitly described in the form of predicates. In this appendix, we present a Prolog-based representation. It binds the actual data and data schema together in a natural and flexible way. In addition to expressing all the information which can be represented in the entity-relationship (E-R) model, the representation can represent other kinds of semantic information as well.

Based on the representation, an approach to generation of semantic networks from relational data base schemata is described in this appendix. ML facilities in an IDB system can be triggered by the relationships of causality and logical implication types acquired in the approach.

D.2 Motivations

Over the past twenty years data base research has evolved technologies that are now widely used in almost every computing and scientific field. However, many new advanced applications including computer-aided design (CAD), computer-aided manufacturing (CAM), computer-aided software engineering (CASE), image processing, and office automation (OA) have revealed that traditional DBMSs are inadequate, especially on the following cases:

- Conventional data base technology has laid particular stress on dealing with large amounts of persistent and highly structured data efficiently and using transactions for concurrency control and recovery. For some applications like CAD/CAM [Wu 92b] where the data schemata need to vary frequently, new data models are needed.

- In some applications like geographical data and image data, the semantic relationships among data need to be represented as well as the data itself. Conventional data models in data base technology cannot support any representation facility for complex semantic information.

- Traditional data base technology can only support facilities for processing data. Along with the developments of other subjects, like decision science and AI, more and more applications need facilities for supporting both data and knowledge management.

To widen the applicability of data base technology to these new kinds of applications, object-oriented data models have been proposed as the data models of next-generation DBMSs [Cattell *et al.* 91] to handle more complex kinds of data and objects and deductive data bases have been expected to support a solution to process both knowledge and data models.

In object-oriented approaches, complex data structures (e.g. multimedia data) can be defined in terms of objects. Data that might span many tuples in a relational DBMS can be represented and manipulated as a data object. Procedures/operations as well as data types can be stored with a set of structural built-in objects and those procedures can be used as methods to encapsulate object semantics. Containment relationships between objects may be used to define composite or complex objects from atomic objects. An object can be assigned a unique identifier which is equivalent to a primary key in a relation. Relationships between objects can also be represented more efficiently in object-oriented data models by using a more convenient syntax than relational joins. Also, most object-oriented DBMSs have type inheritance and version management as well as most of the important features of conventional DBMSs.

Deductive data base systems provide knowledge management, supporting a number of rules for automatic data inferring and management of integrity constraints between data. Rules in deductive data bases are also called intensional data bases while the explicitly stored data are called extensional data bases (EDBs). There are several different approaches [Draxler 91] to implement deductive data base systems, such as integration and coupling on a physical or a logic level, but their EDBs are mostly relational. As the relational data model and Prolog have a common theoretical foundation [Zaniolo 86] and Prolog is a programming language that contains within it the language of relations and can thus be used in a very direct way to implement relational data bases, much of the research on both deductive data base systems and even conventional relational DBMSs has been implemented in Prolog [Li 84, Kazic *et al.* 90, Nieme *et al.* 91].

The normal way in existing deductive data base systems to model relational data bases in Prolog is based on the following analogies: a relational tuple corresponds to a fact in Prolog, the collection of tuples in a relation corresponds to the facts with the same predicate name, and constraints and queries are represented as Prolog rules. There are two disadvantages in this conventional approach:

- It does not represent data schemata explicitly. Users must remember exactly all structures of different fact collections when, e.g. defining relational operations, which means it is impossible to manipulate relations by giving only relation names and field names. From the view of the DBA (data base analyst), the management of larger applications also becomes more difficult.

- It is inconvenient for data restructuring which presupposes the capability to add, modify and remove schema components and causes corresponding changes in the actual data.

One of the motivations of the representation described below is to represent relational data bases in such a way that the above disadvantages of the conventional approach can be eliminated. The other motivation is the insufficiency of the E-R model, which is a widely adopted data abstract model for the conceptual structure design of data bases, in expressing semantic information. The simple relationship types in the E-R model, such as one to many (1:N) and many to many (M:N), cannot describe well the different explicit semantic features of the relationships among entities, still less the variations and developments of entities in function, performance, structure, status and attributes etc. with time and external variables' variations. The aim of the representation is to integrate knowledge and data in such a natural way that all the information which can be represented in the E-R model and other kinds of semantic information which cannot be described well in the E-R model can both be easily expressed and that the semantic information can be used to couple ML facilities with data base and knowledge base technology in order to implement knowledge acquisition from data bases.

D.3 The representation

Our representation consists of two parts: the first part for relational data bases and the second part for semantic information.

D.3.1 Representation for the relational model

There are two ways to represent relational tuples. One represents them as labeled n-tuples and the other as ordered n-tuples. In the second, an n-tuple is usually represented in the form of $(V_1, ..., V_n)$ where the values $V_1, ..., V_n$ appear in the same order as their field names in the relation schema. As lists are a common form of representation in Prolog where the relative positions of elements can be taken as important, the representation below is based on the ordered n-tuples way.

The following is a BNF (*Backus Normal form*) notation for representing a relational data base within our representation.

```
<Database> := <Relation>{, <Relation>}*
<Relation> := relation(<Relation Name><Field List>{<Tuples>}_0^1)
<Relation Name> := <Prolog Name>
<Field List> := <Field>{,<Field>}*
<Field> := <Field Name><Field Type>
<Field Name> := <Prolog Name>
<Field Type> := char|string|logical|integer|real|date
<Tuples> := <Tuple>{,<Tuple>}*
<Tuple> := <Element>{, <Element>}*
<Element> := char(Char)|string(String)|logical(Boolean)|
             integer(Integer)|real(Real)|date(String)
<Prolog Name> := (any legal Prolog atom)
```

A relation generated by the above BNF notation has the structure of

$$relation(RelationName, FieldList, Tuples) \tag{D.1}$$

or

$$relation(RelationName, FieldList).$$

Each relation in a relational data base has a unique name, *RelationName*. The predicate *relation* describes all the fields and possible tuples in the relation *RelationName*. Fields in a relation are described by an ordered list, *FieldList*. Their types are identified by the atoms *char, string, logical, integer, real* and *date*, which denote the domain of single characters, character strings, truth-values, integers, real numbers and specific strings for date description. Each field can be uniquely identified as

$$field(RelationName, FieldName, Type). \tag{D.2}$$

The component *Tuples* in a relation supports a Prolog representation of relational tuples. It contains those tuples of which the relation value consists. In the *Tuples* in a relation, the value of each field appears in the same position as the field name in the field list. It is easy to define structural constraints which check that each tuple confirms to the fields description in a relation and is uniquely defined. This is the way our representation binds relational schemata and relational tuples. In other words, the *Tuples* component describes the relational tuples, whereas the components *RelationName* and *FieldList* belong to the relational data schemata. All of *RelationName, FieldList* and *Tuples* are represented explicitly and can thus be manipulated easily. Constraints between fields and dependency types in relationships will be represented in Section D.3.2.

It is convenient to define a predicate:

$$keyfield(RelationName, KeyFieldList)$$

where $KeyFieldList := field\{, field\}^*$ as the key fields of relation *RelationName*. Since in some relational DBMSs (e.g. dBASE3), key fields are not explicitly defined, we did not include the *keyfield* predicate in our representation.

D.3.2 Representation for more semantic information

The E-R model is one of the most successful methods of formulating useful abstract models in the conceptual structure design of data bases and the key design aid for conventional data bases implemented under a wide variety of commercially available systems [Kazic *et al.* 90]. By focusing on the entities and their relationships, it structures the way designers approach the problem of creating extensible data bases. However, there are two substantial problems here. One is that transforming an E-R model into a relational model during the logical design of data bases results in loss of some semantic information that exists in the E-R model. In other words, the entities and relationships are not distinguished in the relational data model. It is impossible for the relational data model to describe the changes of relationship(s) and other entities caused by an entity in an E-R model. For example, *age* is an important factor for counting an employee's *salary* in many British institutions. However, we cannot explicitly express whether the employee's salary will increase according to the change of his/her age in the relational data model. The other problem is that the E-R model itself is insufficient in expressing complex semantic information as its relationship types, such as one to many and many to many, are too simple to describe explicitly semantic features of the relationships between entities and within entities themselves. For example, different types of relationships, such as logical implication and conceptual inheritance, cannot be expressed in the E-R model.

The E-R model and the relational data model are successful in those applications where only the ability to deal with large amounts of persistent and fixed-format data efficiently is needed. For new applications, such as those mentioned in the motivations, new representation models are in demand. Object-oriented data models are a new generation of extended data models, based on the relational data model. However, as we can see from their main features, briefly summarized in Section D.2, object-oriented models are themselves data models although some systems (e.g. POSTGRES [Cattell *et al.* 91]) have included rule processing facilities. Data management, object management and knowledge management are three different dimensions of problem solving techniques. They would all be needed in some complex applications.

Knowledge management entails the ability to represent, acquire and enforce a collection of expertise such as those which form part of the semantics of an application. Such expertise describes integrity constraints among data in the application as well as allowing the derivation of data which is usually called virtual data contrasting to the real data stored in the data base(s). The task of knowledge management is a key motivation of deductive data base research.

The representation described in this appendix is basically designed for the approach that generates semantic networks from relational data base schemata (see Section D.5). Therefore, we have put an emphasis on representing the semantic information which cannot be represented in the relational data model and the E-R model.

Semantic information in the real world includes four different categories:

- descriptive knowledge about entities,
- inherent laws and constraints between attributes or fields in entities,
- relationships among entities which can be further divided into six types [1], i.e., hierarchy, fellow member, attribute, role, causality and logical implication, and
- dependency types in the relationships between entities.

The following are some predicates in our representation used to express semantic information. The examples for those predicates will be mainly drawn from the sample data base schemata in Figure D-1.

D.3.2.1 Distinguishing entities and relationships

Each relationship (Relation) is distinguished with a predicate as

$$is - assoc(Relation). \tag{D.3}$$

In Figure D-1, *Dependant* and *Employee* are two entities whereas *Assignment* is a relationship indicating a manager monitors employees to work for a project.

---

[1] In order to give a more precise semantic classification, it is possible to divide one or more of the relationship types here into greater detail. The completeness of a semantic model can only be defined in terms of specific applications. We cannot say whether all the relationships here are necessary for every application. Neither can we say they are complete. However, as we can see from Figure D-1, they do exist in the real world.

Clearly, each entity (Entity) satisfies the feature below:

entity(Entity) :-

relation(Entity,_,_), not(is-assoc(Entity)).

Each entity-relationship association is described with predicate *assoc-entity*

$$assoc - entity(Relation, EntityList, AssocTypeList) \tag{D.4}$$

where $AssocType \in \{1, N\}$ denotes the nature of an entity is single or multiple valued in an association.

For instance, relationship *Assignment* contains entities *Employee, Manager* and *Project.*

Information about (D.3) and (D.4) can be found in the E-R model but it is lost when the E-R model is transformed into the relational data model.

D.3.2.2 Identifying the semantic type of each relationship

There are examples of six types of relationships in Figure D-1:

- hierarchy which indicates conceptual inheritance: the relationships between *Employee* and *All Employee* and between *Home Address* and *Address*,

- fellow member: the relationship between *Home Address* and *Office Address*,

- attribute: *Labour* and *Budget* are two attribute entities of entity *Project*,

- role: *Employee Experience* and *Manager Experience* are two role entities in the *Assignment* relationship,

- causality: the *Labour.Title* of an employee in *Employee Qualification* may be a reason for his/her *Employee.Title* assignment in relationship *Assignment* and

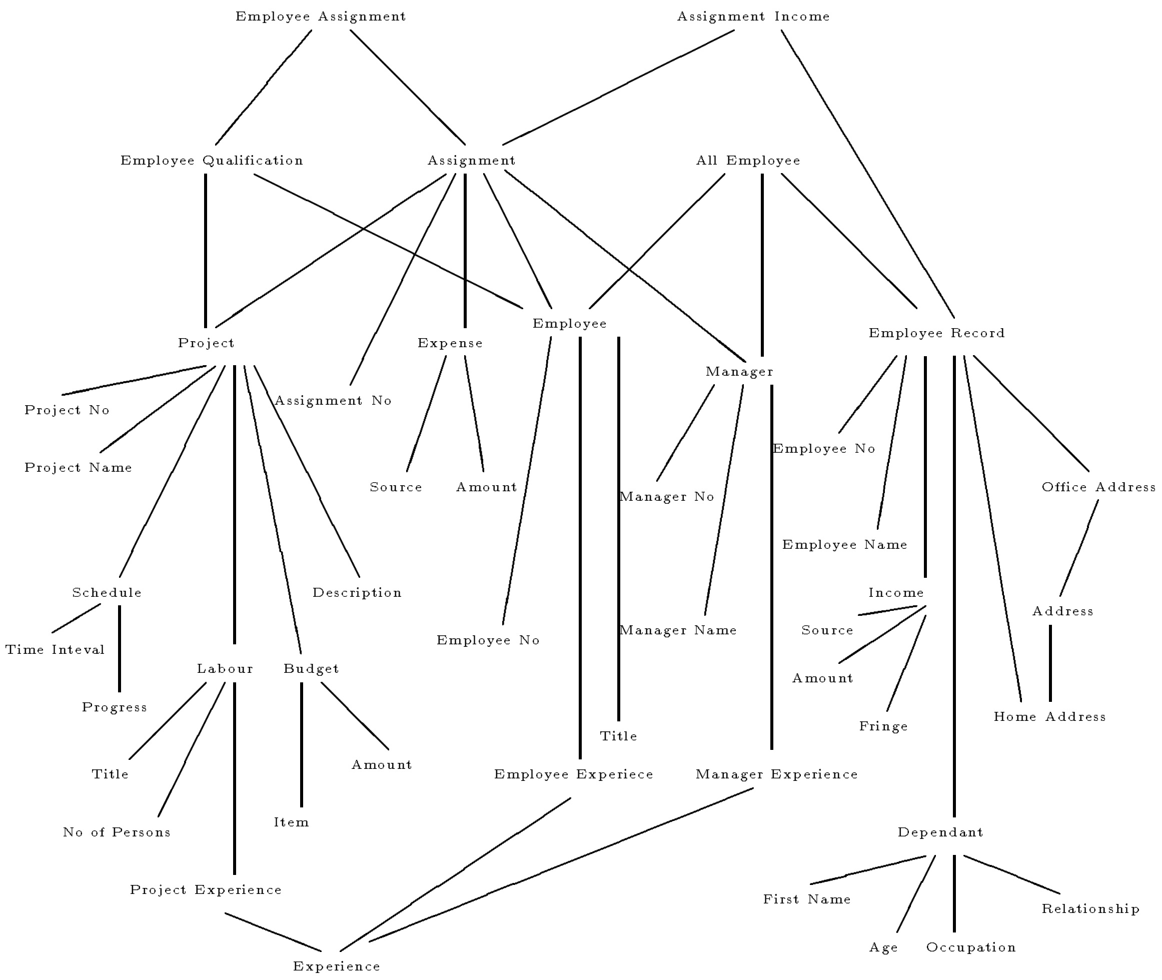

Figure D–1: Sample Data Base Schemata (Derived from [Su *et al.* 80])

- logical implication: the *Income.Fringe* of an employee can be concluded from his *Project* in *Assignment*, say (Employee No = 14, Project No = 4 → Income.Fringe = 150), in the *Assignment Income* relationship.

The semantic type (AssocType) of each relationship (Relation) is identified by

$$assoc - type(Relation, AssocType). \tag{D.5}$$

Different types of relationships have different 1) structural features in describing the formulation of the relationships, 2) semantic integrity constraints on data, and 3) operational features or behaviour, such as insertion, deletion, comparison and retrieval, on the data in the relationships [Su *et al.* 80].

D.3.2.3 Representing semantic labels in each relationship

Semantic labels are useful for processing natural-language like queries and firing machine learning engines in intelligent data base systems.

For each type of relationship, there are different semantic labels to identify different roles in the relationship. For example, in a causality relationship, there are two kinds of labels, *cause* and *effect*. In a logical implication relationship, there are also two kinds of labels, condition (if) and conclusion ($then$). A key entity in a relationship can be given a *key* label to identify the relationship. For example, if a *Project* needs a specific *Assignment*, we say the *Project* entity is a key entity in the *Assignment* relationship.

Each entity's semantic label in each relationship is identified by

$$label(Entity, Relation, Label). \tag{D.6}$$

For example, in the *Assignment Income* relationship, we have the following labels.

$$label(Project, AssignmentIncome, cause)$$

$$label(Income, AssignmentIncome, effect)$$

D.3.2.4 Representing deductive knowledge

Knowledge about causality and logical implication is necessary for deductive data bases to establish virtual data. In existing deductive systems, this is often represented as production rules. As there are several disadvantages inherent in conventional production rules, we represent deductive knowledge in the form of "rule schema + rule body" [Wu 90,Wu 91] (see also Section 6.2). The Prolog representation is thus

$$schema(Relation, CauseEntityList, ResultEntity), \tag{D.7}$$

$$body-left(Relation, No, CauseOrResultEntity, Attri, RelSym, Value), \tag{D.8}$$

$$body-right(Relation, No, ResultEntity, Attri, Value) \tag{D.9}$$

where No is used to identify different parts of the same body, $Attri$ indicates an attribute and $RelSym$ denotes a conventional arithmetic or symbolic relation.

For the example given for the logical implication, we can express it as:

$$schema(AssignmentIncome, Project, EmployeeRecord),$$

$$body-left(AssignmentIncome, 1, Project, Project\_No, =, 4),$$

$$body-left(AssignmentIncome, 1, EmployeeRecord, EmployeeNo, =, 14),$$

$$body-right(AssignmentIncome, 1, EmployeeRecord, IncomeFringe, 150).$$

D.3.2.5 Representing constraints knowledge

Constraints are important in the relational data model. Three sorts of constraints have been classified and represented in our representation. The first is about the integrity of attributes in each relation,

$$constraint1(Relation, Attribute, RelSym, Value). \tag{D.10}$$

For example, in the $Dependant$ relation, the AGE attribute is supposed to be always less than 120.

The second is the dependency type of each relationship, such as one-to-one (which means a result entity tuple has a unique corresponding tuple of each cause entity, e.g. an *Assignment* tuple corresponds to a unique *Project* tuple), full (which means all possible tuples of the result entity have their corresponding cause entitys' tuples, e.g. each *Assignment* tuple must have its corresponding *Project*, *Expense* and *Employee* tuples) or dual (each tuple of a result entity corresponds to a tuple of each cause entity and *vice versa*, e.g. each *Assignment* tuple has its own *Project* tuple and *vice versa*),

$$constraint2(Relation, MappingType). \tag{D.11}$$

The third is the constraint relationship between an attribute in a relation and outer variables,

$$constraint3(Relation, Attribute, OuterVariableList, ConstraintString). \tag{D.12}$$

See the example in Section D.3.2.6 where *Year* could be an outer variable of Figure D-1.

Here, semantic constraints about relational data have also been explicitly expressed rather than being hidden in application programs. This feature of our representation makes it easier to maintain and adapt application programs.

D.3.2.6 Representing regularities between attributes

These represent inherent regularities between attributes, for example, the time-dependent function of an attribute, and the function or logical dependency relationship among the attributes,

$$function((Relation, Attribute), (Rel, Attri)*, Function) \tag{D.13}$$

where $(Rel, Attri)*$ indicates a list of relational attributes. For instance, if an employee was born in 1950, his age can be computed by the following regular knowledge.

$$function((Employee, Age), [(Time, Year)], Age = Year - 1950)$$

D.4 Discussions

Predicates (D.1), (D.2), (D.3) and (D.5) above are homologous to the node descriptions in domain semantic networks, while Predicates (D.4), (D.6) and (D.12) homologous to directed arcs. Predicates (D.7), (D.8) and (D.9) are homologous to reasoning networks in production systems and Predicates (D.10), (D.11) and (D.13) may be used to define deep[2] knowledge of problem domains. It is still difficult to adopt semantic networks to represent reasoning networks and deep knowledge with the existing techniques. The above thirteen predicates have thus formed a Prolog-based representation for complex applications where both knowledge and data management is needed. Such a representation can represent any information that can be expressed in the E-R model.

Also, the representation which consists of the thirteen basic predicates describes explicitly relational schemata as well as relational tuples, thus the disadvantages of the normal way to model relational data bases in Prolog discussed in Section D.2 have been eliminated.

D.5 An approach to generation of semantic networks from relational data base schemata

Based on the Prolog-based representation above, an approach to generation of semantic networks from relational data base schemata has been designed and partly integrated into *KEshell2* (see Chapter 6) to couple ML facilities with data base and knowledge base technology.

---

[2]In contrast to the shallow knowledge (which is directly used for problem solving) in knowledge bases in expert systems, deep knowledge in problem domains can be used to detect inconsistencies in shallow knowledge and data.

The main idea of the approach is: first transform the static description of descriptive data base schema into an active knowledge description and then acquire the semantic information lost in the relational data model and the E-R model.

The generation approach covers three steps.

Step 1: Generating predicate description (D.1) and (D.2) from relational schema description.

Step 2: Identifying entities and relationships by using Predicates (D.3) and (D.4).

Predicate (D.3) can be used to conduct the acquisition of Predicate (D.4) and the semantic association types and the deductive knowledge in Step 3. Predicate (D.4) can be used to conduct the acquisition of semantic labels in Step 3 and to confine the consistency test of target data bases.

Step 3: Acquiring the information that cannot be described in the E-R model.

1) Identify the semantic type (predicate (D.5)) of each relationship and acquire each entity's semantic label (predicate (D.6)) in each relationship according to the structural properties and operational features of each relationship type.

2) Acquire deductive knowledge.

For each causality relationship, first generate a rule schema (Predicate (D.7)) and then acquire the corresponding rule body of the rule schema in interactive mode, its structure being Predicates (D.8) and (D.9).

For each logical implication relationship, first acquire a rule body and then generate a corresponding rule schema.

3) Acquire constraints knowledge (Predicates (D.10), (D.11) and (D.12)).

4) Acquire the regular knowledge of the attributes themselves (predicate (D.13)).

Appendix E

LFA: a linear forward chaining algorithm

E.1 Domain reasoning networks

Based on the rule schema + rule body representation in Section 5.2.2, this appendix describes a linear forward chaining algorithm, *LFA* [Wu 93d].

Definition E-1. A domain reasoning network is an AND/OR tree associated with a knowledge base in rule schema + rule body by the following analogies:

1) Nodes in the tree correspond to factors in the knowledge base.

2) A rule schema *IF* $E_1, ..., E_n$ *THEN* A in the knowledge base corresponds to the arcs, which indicates the hierarchy among factors, in Figure E-1 in the tree.

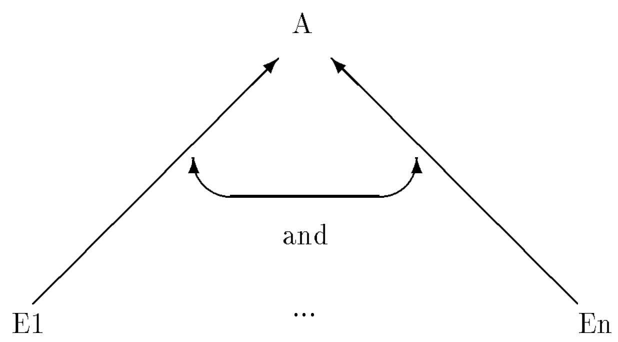

Figure E–1: A Rule Schema

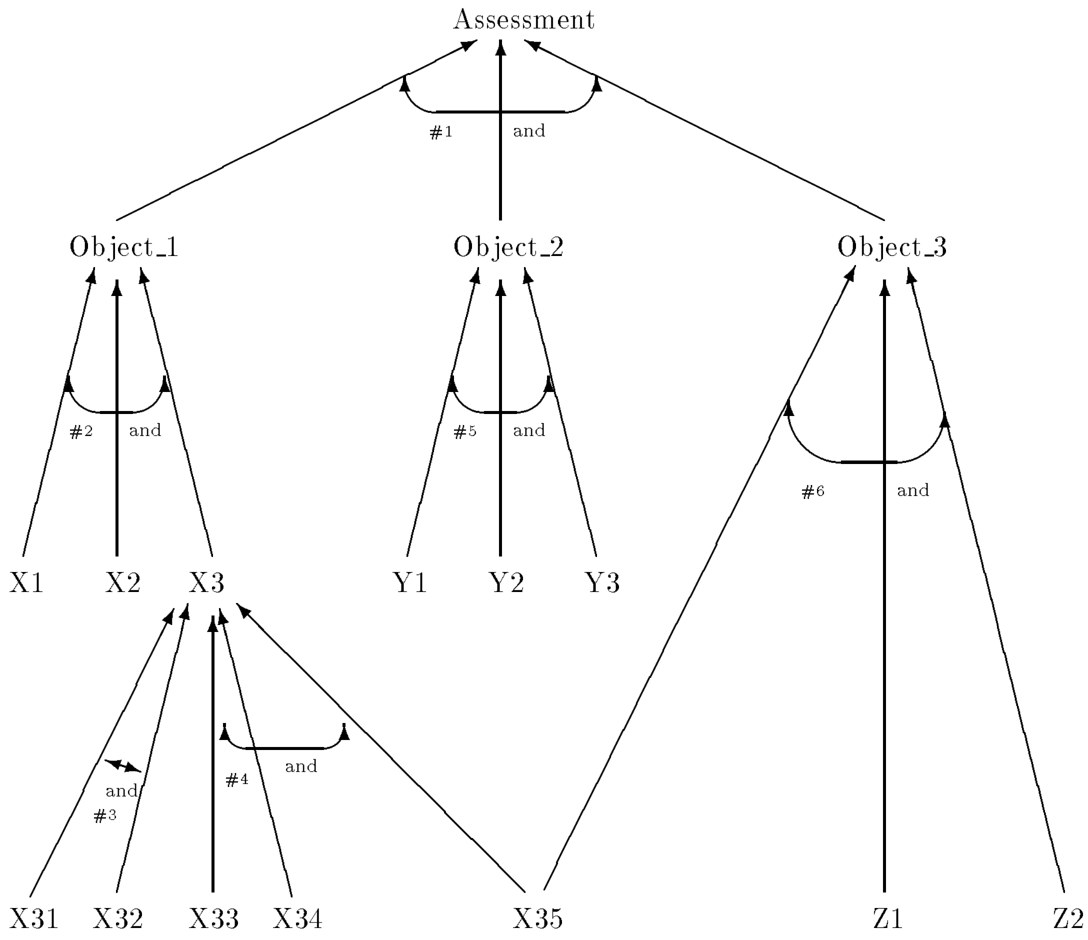

Figure E–2: The Domain Reasoning Network for Example 5-3

The domain reasoning network corresponding to the knowledge base in Example 5-3 is shown in Figure E-2.

We can easily find some similarities between the nodes here in domain reasoning networks and the features in the Rete-like discrimination networks [Lee *et al.* 92]. However, both the Rete-like discrimination networks and the decision trees used in [Ghallab 81] only compile the LHSs of rules in a knowledge base and deal with detailed attribute values while the domain reasoning networks compile both LHSs and RHSs of rule schemas and have no specific attribute values involved.

Definition E.2. In a domain reasoning network, a top node is a goal which is supposed to be a solution (e.g. a disease) to the problem domain, a terminal node is a user node whose possible data (e.g. a symptom) is supposed to be given by users, and a middle node is a subgoal.

Definition E.3. A knowledge base in rule skeleton + rule body is in *a partial order* means if Rule Schema $\#N$ is `if Factor-1, ..., Factor-n, then Factor` then all the rule schemata with `Factor-1, ..., Factor-n` as their conclusion factors have rule-set ordinals smaller than N.

The *LFA* algorithm, which performs forward chaining on knowledge bases in rule schema + rule body, comprises two major strategies: sorting the knowledge in a knowledge base into a partial order according to the hierarchy among factors at the end of knowledge acquisition or knowledge modification and using the renumbered knowledge in 2-phase "matching – action" cycles during problem solving.

E.2 Sorting knowledge in a knowledge base into a partial order

The sorting process covers four steps.

(1) Find and remove dead cycles.

Definition E.4. A cycle in a domain reasoning network is *a dead cycle* if none of the nodes involved in the cycle are terminal nodes in the domain reasoning network, which means that their evidence is not supposed to be given by users, and there is no other rule schema whose conclusion factor is one of them.

Example E-1. The cycle "$A_5 \rightarrow A_3 \rightarrow A_1 \rightarrow A_5$" in Figure E-3 is a dead one.

A dead cycle in a domain reasoning network is an error in the corresponding knowledge base because none of the factors involved can be computed during problem solving.

(2) Renumber all the rule schemata whose premise factors are all terminal nodes in the domain reasoning network. For any factor F, if all the schemata with

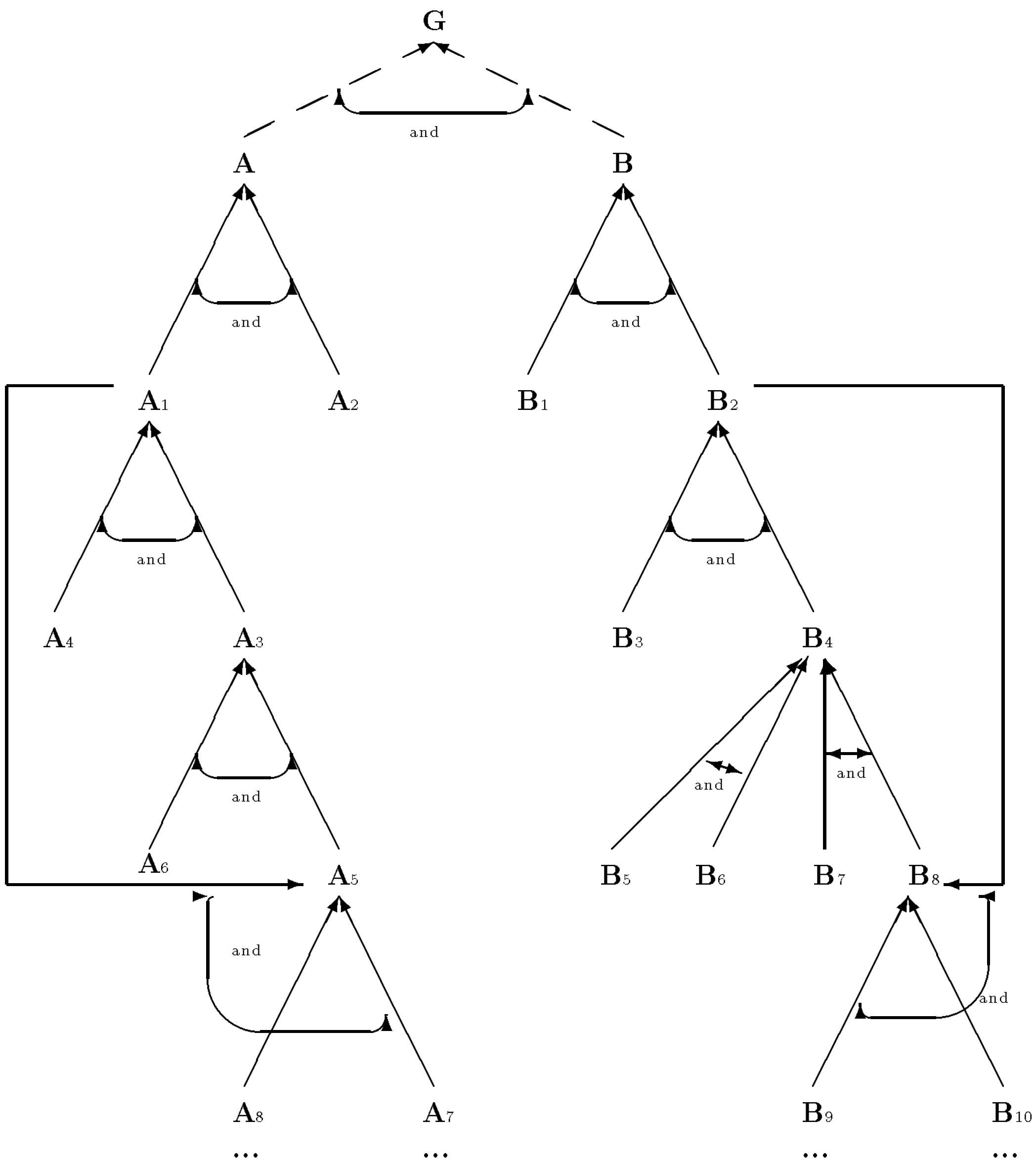

Figure E–3: A Dead Cycle and a Live Cycle it as their conclusion factor have been renumbered, it is treated as a terminal node for further renumbering. If all the rule schemata in a knowledge base have been renumbered, goto (4).

(3) Resolve a live cycle and goto (2).

Definition E.5. A cycle is called *a live cycle* when there is such a F in the cycle, called *a live node*, that is either a terminal node or is a conclusion factor of a rule schema IF $E_1, ..., E_n$ THEN F and none of $E_1, ..., E_n$ are involved in any dead cycles.

Example E-2. The cycle "$B_8 \rightarrow B_4 \rightarrow B_2 \rightarrow B_8$" in Figure 6-3 is a live one and B_4 is a live node in the cycle.

A live cycle can be resolved by treating one of its live nodes as a terminal node for further renumbering.

(4) Stop.

For instance, a partial order for the knowledge base in Example 5-3 is as follows.

old ordinal	*new ordinal*
#3	#1
#4	#2
#2	#3
#5	#4
#6	#5
#1	#6

For those problems which can be represented in the rule schema + rule body language, the sorting process above is always feasible if all possible dead cycles can be removed by domain experts. However, for other problems which require dynamic creation of nodes in their reasoning networks, neither the rule schema + rule body language nor the sorting process will be well suited.

E.3 Linear forward chaining

After knowledge sorting, the process for forward inference is designed as follows.

for the first renumbered schema to the last one in the knowledge base do

if there exists data in the working memory for each of the

premise factors of the schema

then fire the corresponding rule body of the schema

next schema

We can easily prove that the time complexity of the above algorithm is $O(n)$ where n is the number of rule sets.

Consider Rule Schema $\#N$.

```
Rule Schema #N: If Factor-1, ..., Factor-n then Factor
```

When $N = 1$, all the Factor-i's $(i = 1, ..., n)$ must be terminal nodes in the domain reasoning network and their possible data are supposed to be given by users before forward inference starts. So it is clear at the start of inference whether or not Rule Schema #1 is successful in matching the working memory. When $N = 2$, there are two possible cases. One is that all the Factor-i's are terminal nodes, this case is similar to $N = 1$. The other is that there exists some Factor-$j(j \in [1, n])$ which is not a terminal node. In the latter case, the Factor-j must be the conclusion factor of Rule Schema #1. So after the matching and possible action of Rule Set #1, whether Rule Schema #2 is successful in matching the working memory and whether its corresponding body is able to be operated are clear. When $N = K > 2$, there are two possible cases similar to $N = 2$. The first case is that all the Factor-i's are terminal nodes, which is similar to $N = 1$. The second is that there exists one or more Factor-j(s) as non-terminal node(s). From the sorting process of above, each possible non-terminal node Factor-j must be the conclusion of one or more Rule Schema #1, ..., Rule Schema $\#(K-1)$.

When it is time to match Rule Schema #K, whether each Factor-i has evidence and their possible evidence are known already. Therefore, whether Rule Schema $\#K$ is successful in matching the working memory and whether its corresponding rule body is able to be operated are clear then. As soon as the matching of a rule schema is successful, which means there exist data for all of its premise factors, operate the corresponding rule body to produce new data of its conclusion.

From the analysis above, we can see that the most significant feature of the *LFA* algorithm is that it can properly choose one rule set from the sorted knowledge base at each cycle without any specific conflict resolution.

E.4 Restrictions on LFA

There are two restrictions on the *LFA* algorithm: static inference, which means the knowledge in a sorted knowledge base cannot be changed by either users or the system itself during inference, and collecting all possible evidence at the start of inference to give all possible solutions. Data once acquired cannot be changed by users. When problems require dynamic creation of nodes in their reasoning networks or external modification of rules and data during inference, or their initial evidence is not available at the start of inference, the algorithm is not well suited.

Appendix F

An example run of SIKT in KEshell

The following records the dialogue between an expert and the *SIKT* module in *KEshell* to build the knowledge base in Example 5-3 in Chapter 5.

The sentences with K: at the beginning are generated by *SIKT* and those with M: are input of the expert. Words between /* and */ are explanatory notes.

K: Give all the goals (i.e. the top nodes in your domain reasoning network):

M: Assessment

K: Define the domain type of Assessment: (1:Logical True/False;2:A Finite Set; 3:A Real)

M: 2

K: Give all the factors which can determine Assessment in the 1st method:

M: Object_1, Object_2, Object_3

K: Any other method to determine Assessment? (Y|N)

M: N

K: Define the domain type of Object_1: (1:Logical True/False;2:A Finite Set;3:A Real)

M: 3

K: Are users expected to provide data for Object_1? (Y|N)

M: N

K: Give all the factors which can determine Object_1 in the 1st method:

M: X1, X2, X3

K: Any other method to determine Object_1? (Y|N)

M: N

K: Define the domain type of X1: (1:Logical True/False;2:A Finite Set;3:A Real)

M: 3

K: Are users expected to provide data for X1? (Y|N)

M: Y

K: Define the domain type of X2: (1:Logical True/False;2:A Finite Set;3:A Real)

M: 3

K: Are users expected to provide data for X2? (Y|N)

M: Y

K: Define the domain type of X3: (1:Logical True/False;2:A Finite Set;3:A Real)

M: 3

K: Are users expected to provide data for X3? (Y|N)

M: N

K: Give all the factors which can define X3 in the 1st method:

M: X31, X32

K: Any other method to determine X3? (Y|N)

M: Y

K: Give all the factors which can determine X3 in the 2nd method:

M: X33, X34, X35

K: Any other method to determine X3? (Y|N)

M: N

K: Define the domain type of X33: (1:Logical True/False;2:A Finite Set;3:A Real)

M: 3

K: Are users expected to provide data for X33? (Y|N)

M: Y

... /* the same way as above to transfer information about X34, X35, X31

and X32 */

K: Define the domain type of Object_2: (1:Logical True/False;2:A Finite Set;3:A Real)

M: 2

K: Give its domain:

M: A, B, C

K: Are users expected to provide data for Object_2? (Y|N)

M: N

K: Give all the factors which can determine Object_2 in the 1st method:

M: Y1, Y2, Y3

K: Any other method to determine Object_2? (Y|N)

M: N

K: Define the domain type of Y1: (1:Logical True/False;2:A Finite Set;3:A Real)

M: 2

K: Give its domain:

M: A, B, C

K: Are users expected to provide data for Y1? (Y|N)

M: Y

... /* the same way as above to transfer information about Y2 and Y3 */

K: Define the domain type of Object_3: (1:Logical True/False;2:A Finite Set;3:A Real)

M: 1

K: Are users expected to provide data for Object_3? (Y|N)

M: N

K: Give all the factors which can determine Object_3 in the 1st method:

M: X35, Z1, Z2

K: Any other method to determine Object_3? (Y|N)

M: N

K: Define the domain type of Z1: (1:Logical True/False;2:A Finite Set;3:A Real)

M: 1

K: Are users expected to provide data for Z1? (Y|N)

M: Y

... /* the same way as above to transfer information about Z2 */

K: OK. All the rule schemata you have input are:

K: Schema #1: IF Object_1, Object_2, Object_3 THEN Assessment

K: Schema #2: IF X1, X2, X3 THEN Object_1

K: Schema #3: IF X31, X32 THEN X3

K: Schema #4: IF X33, X34, X35 THEN X3

K: Schema #5: IF Y1, Y2, Y3 THEN Object_2

K: Schema #6: IF X35, Z1, Z2 THEN Object_3

K: No evident inconsistencies have been found in those rule schemata.

K: Now, please give your detailed computing and/or inference rules corresponding

K: to the aforesaid methods:

K: The 1st rule schema is: IF Object_1, Object_2, Object_3 THEN Assessment

K: Give your concrete rules to determine the value/CF of Assessment:

K: (Each rule in a line (i.e. ending with a <RETURN>) and A SPACE LINE at the end!)

M: IF Object_1>85, Object_2=A, CF(Object_3)>0.7 THEN Assessment=A

K: — The (1,1) rule accepted.

M: IF Object_1>85, Object_2=100 THEN Assessment=B

K: — There are errors in *** Object_2=100 THEN Assessment=B ***!

K: Rewrite the whole rule:

M: IF Object_1>85, Object_2=A THEN Assessment=B

K: — The (1,2) rule accepted.

M: IF CF(Object_3)>0.5, Object_1>60, Object_2<>C THEN Assessment=B

K: — The (1,3) rule accepted.

M: Assessment=C

K: — The (1,4) rule accepted.

... /* The same way as above to receive other rules in Rule Set #1. */

K: — The (1,6) rule accepted.

M:

/* an empty line */

K: The 2nd rule schema is: IF X1, X2, X3 THEN Object_1

K: Give your concrete rules to determine the value/CF of Object_1:

K: (Each rule in a line (i.e. ending with a <RETURN>) and A SPACE LINE at the end!)

... /* The same way as above to receive the rules in Rule Bodies #2 – 6. */

K: Thank you. A knowledge base has been basically set up according to your input.

K: No evident inconsistencies have been found in the knowledge base.

K: Do you want to check or modify it? (Y|N)

M: Y

K: Choose one of the options in the following window:

K: A: add a rule schema; D: delete a rule schema;

K: C: change a rule schema; L: list all the schemata;

K: B: adapt rules in a rule set; Q: quit.

M: B

K: Which rule set? (Give the ordinal)

M: 6

K: All the rules in the 6th rule body are:

K: IF X35>75 THEN CF(Object_3)=(CF(Z1)+CF(Z2)-CF(Z1)CF(Z2))/2

K: IF THEN CF(Object_3)=(CF(Z1)+CF(Z2))/(1-min{CF(Z1),CF(Z2)})

K: No more rules.

K: Choose one of the options in the following window:

/* The contents in the former window will be covered by the following contents. */

K: A: append specific rules; D: delete a specific rule;

K: C: change a specific rule; L: list all the rules in the rule set;

K: Q: quit

M: Q

/* The former window is recovered now. */

M: Q

K: OK. Try to sort the knowledge base into a partial order now ...

K: Sorting is successful. The new ordinals are:

K: #3 -> #1

K: #4 -> #2

K: #2 -> #3

K: #5 -> #4

K: #6 -> #5

K: #1 -> #6