EXTRACTING RULE SCHEMAS FROM RULES, FOR AN INTELLIGENT LEARNING DATABASE SYSTEM

GEOFF SUTCLIFFE and XINDONG WU
Department of Computer Science, James Cook University
Townsville, Qld, 4811, Australia

ABSTRACT

A software module for extracting rule schemas from rules, in the context of an intelligent learning data base system (ILDB), is described. The ILDB system employs a two level knowledge representation scheme, comprising rule schemas and rule bodies. This format allows particularly efficient deduction to be performed. An interactive tool may be used to capture knowledge in this format. Alternatively, a machine learning component may be used to induce rules only. Without the corresponding schemas, the induced rules cannot be used efficiently by the deduction engine. The extraction system described in this paper overcomes this weakness. The technique has been tested on well-known data sets.

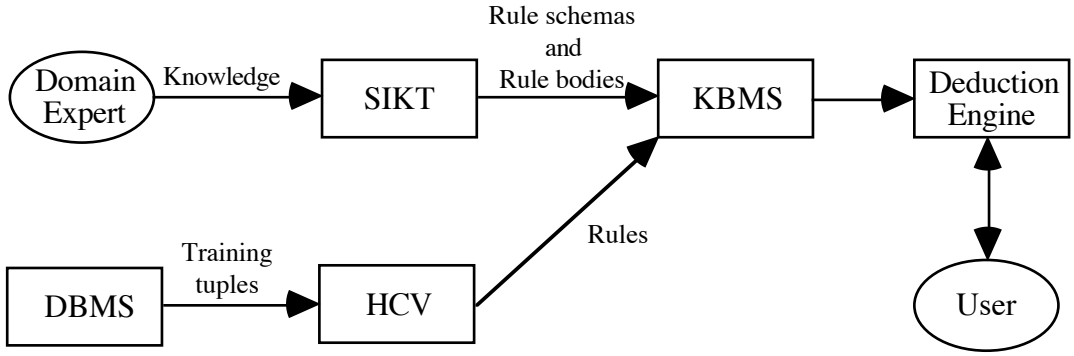
1. Introduction

An intelligent learning data base (ILDB) system is an integrated learning system which implements automatic knowledge acquisition from data bases by providing formalisms for 1) translation of standard data base information into a form suitable for use by its induction engines, 2) using induction techniques to produce knowledge from databases, and 3) interpreting the knowledge produced efficiently to solve users' problems. The translation of data base information to a form suitable for use by induction engines has received some attention[1]. Induction of rules from data base information is a well established field[2,3,4,5,6,7,8,9,10]. There are now induction systems that are capable of dealing with realistically large databases. These induction systems produce knowledge in the form of rule sets that can then be interpreted to solve users' problems. Three typical families of induction algorithms are the generalisation-specialisation based AQ11-like algorithms[3,8], the decision tree based ID3-like algorithms[5,6], and the extension matrix based algorithms[4,10].

An aspect of the third component of an ILDB - "interpreting the knowledge produced efficiently to solve users' problems" - is considered in this paper. This is done in the context of an existing ILDB system, KEShell2[11]. KEShell2 is based on three existing tools : KEShell[12], dBASE3, and HCV[10]. These components are described in Section 2. Briefly, the expert system shell KEShell provides a knowledge structuring system in which the rules are grouped according to schemas. This grouping captures the structures of the rules and allows for their efficient use. HCV is a low-order polynomial induction algorithm that generates rules suitable for use in KEShell. However, the rules produced by HCV are not grouped according to schemas. This paper describes how schemas are extracted from the HCV rules, so that the rules can be grouped and used efficiently in KEShell. Section 3 describes the process of schema extraction, and gives test results for the MONK's problems[13] and the Soybean data[3]. Section 4 concludes the paper.

2. KEShell2 - An ILDB System

KEShell2 is an integrated learning system which couples machine learning techniques with database and knowledge base technology[11]. Figure 1 shows the main components of the KEShell2 system.



DBMS - Database Management System (dBase3) KBMS - Knowledge Base Management System
SIKT - Structured Interactive Knowledge Transfer facility HCV - Heuristic CoVering algorithm

Figure 1: KEShell2 Architecture

In the context of this paper, the substructure including the Deduction Engine, SIKT, and HCV, are of interest. Each of these modules deals with the knowledge that is stored in the KBMS.

KEShell's knowledge representation scheme has two levels: Rule Schemas + Rule Bodies[14], and is based on *(factor,value)* pairs. This is in contrast to typical rule-based knowledge representation, such as in OPS5-like languages[15]. A *factor*, which is similar to an attribute in M.1, EXPERT and KES[16], is a name involved in a domain of expertise. It can be a logical assertion, a discrete set variable or a continuous numeric variable. A rule schema is of the form $\texttt{IF AF}_1\texttt{, … , AF}_n\texttt{ THEN KF}$, where the $\texttt{AF}_i$ and KF are factors. Each $\texttt{AF}_i$ is an antecedent factor, and KF is the consequence factor†. Associated with each schema is a rule body. Rule bodies contain one or more rules. Rules are of the form $\texttt{IF A}_1\texttt{, …, A}_n\texttt{ THEN K}$, where each $\texttt{A}_i$ is antecedent of the rule, and K is the consequence of the rule. The $\texttt{A}_i$ and K are conditions and actions (including arithmetic expressions). Each $\texttt{A}_i$ examines the value of the factor $\texttt{AF}_i$, and the consequence K is performed iff each antecedent $\texttt{A}_i$ has an appropriate value. An example of a rule schema and associated rule body is 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*B - 4*A*C))/(2*A)
IF A≠0 THEN X=(-B - sqrt(B*B - 4*A*C))/(2*A)
```

† The consequence factor is denoted by KF rather than CF to avoid confusion with the common usage of CF to denote confidence factors.

The two level representation provides significant advantages over other representations[15]

- It provides a meta-rule structure by which a deduction engine can avoid attempting to sequentially match each rule in a knowledge base with the data in working memory. This issue, which motivated this work, is discussed below.
- The schemas highlight structure in the rules, thus providing semantic information about the domain.
- The bodies can capture computations directly.

The deduction engine in KEshell2 supports both forward and backward chaining, with the forward chaining working in linear time[17]. Deductions are efficiently implemented, partially due to the two level knowledge representation scheme. A rule cannot be used unless each antecedent factor has a value. Thus, before examining any rules, the deduction engine checks the factors in working memory against the rule schemas. A rule schema is satisfied if all it's antecedent factors have values. Only rule bodies associated with satisfied rule schemas are examined by the deduction engine.

Knowledge acquisition in KEShell2 uses one of two tools : SIKT - a Structured Interactive Knowledge Transfer facility; or HCV - a Heuristic CoVering algorithm.

SIKT[18] is an interactive facility that allows experts and professionals to capture their knowledge in a top down fashion. The knowledge is captured according to the two level knowledge representation scheme, i.e., the user must first enter rule schemas, and then provide the rule bodies for the schemas. A rule body contains several concrete rules. SIKT was inspired by other knowledge-based tools, such as TEIRESIAS[19] and KADS[20].

HCV[10] is a low-order polynomial induction algorithm, based on the extension matrix approach. The rules or formulae produced by HCV take the same form of variable-valued logic[21] as used in AQ11 and have been shown empirically to be more compact than both ID3-like and AQ11-like algorithms. The input to HCV is a set of training examples, each a tuple of factor values attached to an indicator of the class the tuple belongs to. These training examples are obtained from the DBMS in KEShell2. HCV induces rules that recognise tuples in each class. These rules are the output from HCV. The rules output by HCV have to be expanded to a simpler form before they can be used by the KEShell2 deduction engine; the expansion is very simple. More significantly, in contrast to the rule bodies captured using SIKT, the HCV rules (and thus the transformed equivalents) are not grouped by schemas. This means that the deduction engine cannot operate as efficiently with HCV output as it can with user supplied knowledge. It is precisely this gap in KEShell2 that has been filled by the work described in Section 3. The gap is filled by generating schemas for the rules output by HCV.

3. Extracting Schemas from Rules

The process of extracting schemas from the HCV output, and grouping the rules into rule bodies according to the schemas, has been implemented in Prolog.

As a pre-processing step, the HCV output is passed through a simple text processing phase to extract the rules. An example of a typical HCV rule is:

```
       [ head_shape=[square] ] ^
       [ body_shape<>[round,oblong] ] ^
       [ jacket_color=[yellow,blue] ] ^
       [ is_smiling=[yes] ] ^
    --> class='M2'.
```

The antecedent precedes the `-->`, and is a conjunction (as indicated by the `^`s). Each antecedent element specifies values that the factor may (`=`) or may not (`<>`) take. The equality form `[ jacket_color=[yellow,blue] ]` indicates a disjunction, i.e., the `jacket_color` may be `yellow` or `blue`. The inequality form `[ body_shape<>[round,oblong] ]` indicates a conjunction, i.e., the `body_shape` may be neither `round` nor `oblong`. HCV can also produce a special default rule, in the form:

```
       DEFAULT --> the Non-M1 class.
```

which is transformed into

```
       ['TRUE'] --> class='Non-M1'.
```

The `TRUE` antecedent is interpreted specially in the KEShell2 deduction engine. Such rules are executed last, iff no other rules can fire.

Once in this form, the rules are read directly by the Prolog schema extraction program. Each rule is transformed into an `Antecedent --> Consequence` rule structure in Prolog, where the `Antecedent` is a Prolog list of the antecedents. The antecedent elements are ordered using Prolog's built in ordering function, applied to the antecedents' factors. Note that one HCV rule may be transformed into more than one `Antecedent --> Consequence` rule structure, due to the existence of disjunctions in HCV rules. For example, the first HCV rule above is translated into the following two rules :

```
([head_shape=square,body_shape<>[round,oblong],jacket_color=yellow,
  is_smiling=yes] -->class='M2')
```

and

```
([head_shape=square,body_shape<>[round,oblong],jacket_color=blue,
  is_smiling=yes] -->class='M2')
```

Each structure thus formed is examined, and a schema corresponding to the rule is generated and paired with the rule. The schemas generated contain the factors from the antecedent of the rule and the factor from the consequent. The antecedent factors are, like the antecedents themselves, ordered according to Prolog's built-in ordering function. For example, the schema - rule pair formed from the first rule above is:

```
([head_shape,body_shape,jacket_color,is_smiling]-->class)-
([head_shape=square,body_shape<>[round,oblong],jacket_color=blue,
  is_smiling=yes] -->class='M2')
```

All the schema - rule pairs are grouped into a Prolog list. Finally, Prolog's `setof` predicate is used to extract the unique schemas, and to group the rules into rule bodies, according to the unique schemas. The Prolog code required to do this is pleasantly elegant:

```
setof(UniqueSchema-RuleBody,
    setof(Rule,member(UniqueSchema-Rule,SchemaRulePairs),RuleBody),
SchemasAndBodies),
```

where `SchemaRulePairs` is the Prolog list of schema - body pairs. The output list, `SchemasAndBodies`, is a list of pairs. Each pair consists of a unique rule schema and rule body. Each rule body is a Prolog list of rules that fit the associated rule schema. The earlier ordering of the antecedent factors in the schemas and rules means that the components of rule antecedents may initially be in any order. After the ordering, equivalent schema antecedents will be syntactically equivalent, which simplifies the grouping task. The rule schemas and associated rule bodies are then output in an appropriate format. Example output, containing the two rules above, looks like:

```
Rule Set #2 with 3 rules.
  Schema:
    IF head_shape,body_shape,jacket_color,is_smiling THEN class
  Body:
    IF head_shape=square, body_shape<>[round,oblong],
       jacket_color=yellow, is_smiling=yes THEN class=M2
    IF head_shape=square, body_shape<>[round,oblong],
       jacket_color=blue, is_smiling=yes THEN class=M2
    IF head_shape=round, body_shape=round, jacket_color=yellow,
       is_smiling=no THEN class=Non-M2
```

Note that the rules associated with a given rule schema need not have the same consequence. The only restriction is that the factors in the antecedents and consequences must be the same. The rule schemas in the KBMS can be further ordered to achieve linear time complexity deductions[17].

4. Results

The extraction process has been run on the HCV output for the three MONK's data sets[13] and the Soybean data[3]. The MONK's problems are derived from an artificial robot domain, in which robots (examples) are described by six multiple-valued attributes. The sizes of the attribute value sets range from 2 to 4, giving 432 possible examples. The MONK's problems are all binary classifications, defined over the same space. The three problems differ in the type of the concept to be learned and in the amount of noise in the training examples. The Soybean data characterises diseases in soybean plants. There are 35 attributes with value set sizes ranging from 2 to 7. There are approximately 10^{15} possible examples with 19 different classifications (diseases). The table below shows results for each data set. The table lists the sizes of the data sets (numbers of examples), number of rules induced by HCV, expanded number of rules for KEShell2, and number of schemas extracted. The maximum number of KEShell2 rules associated with a schema is also shown. In each example there are about half the number of schemas as there are KEShell2 rules.

Data Set	Size of Data Set	HCV Rules	KEShell2 Rules	Schemas	Max. Rules
MONK1	124	7	8	4	3
MONK2	169	40	50	26	7
MONK3	122	19	24	13	5

Soyabean	683	48	131	48	12

The schema extraction program is added into the KEShell2 architecture, as shown in Figure 2.

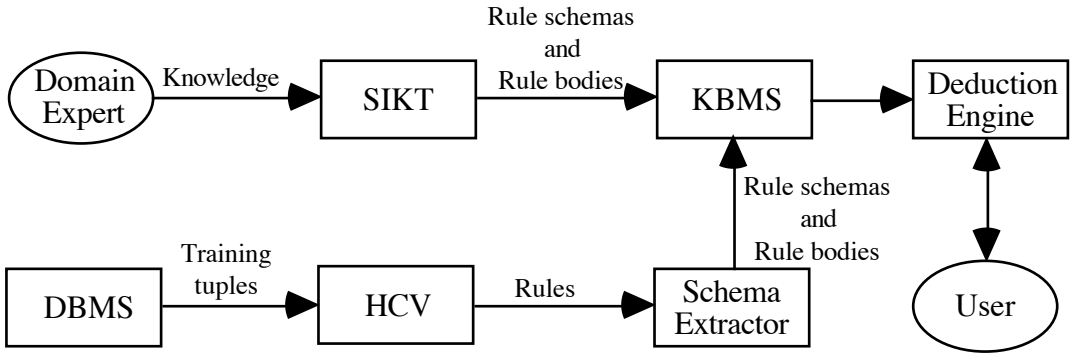


Figure 2 : Extended KEShell2 Architecture

The first effect of grouping the rules into rule bodies (according to the schemas) is to speed up the deduction process. There are two cases, depending whether the deduction strategy is forward chaining or backward chaining:

In forward chaining the deduction engine is given a tuple of factor values, and asked to classify the tuple. Without the rule schemas (or equivalently, with one rule schema per rule) the deduction engine will on average have to scan through half the rules to find a match, or through all the rules if there is no applicable rule. In contrast, to find an applicable rule with rule schemas, half the rule schemas have to be examined to find the appropriate rule body and half of the rules in the rule body would be scanned to find the match. In the case of no applicable rule, half the rule schemas have to be scanned to find the appropriate rule body (if one exists, otherwise all of the rule schemas), then all of those rules are examined.

The effect is even more dramatic in backward chaining, when the deduction engine has the attributes in working memory, and iterates through the knowledge base to find an applicable rule. Without rule schemas, the analysis is as above. With rule schemas, if there is an applicable rule, it is found by checking each rule schema against the working memory. If there is no data for one of a schema's antecedent factors, the associated rule body is ignored. Otherwise the associated rules must be checked. The number of rules that have to be checked in this process depends on what values the factors have in the working memory. The number will certainly be less than or equal to the number checked without rule schemas, due to the exclusion of some rule bodies. If there is no applicable rule then the same search process is used. If there is no applicable rule because factor values are missing from the working memory, then only the rule schemas are examined (in contrast to the case without schemas where the rules are examined). If there is no applicable rule because the factors have inappropriate values, then the rule schemas and the rules associated with rule schemas where the factor values are available, will be examined. Again, the number of checks will be less than the number without rule

schemas.

The second effect of grouping the rules into rule bodies is to highlight semantic relationships between factors. For example, in the MONK's data examples that have been tested, the `body_shape` and `jacket_color` factors appear together in most schema attributes. This indicates that these two factors are independent of each other in terms of the classification made. In contrast, factors that seldom appear together in schema antecedents are likely to be dependent on each other. This issue is open to further investigation.

5. Conclusion

The integration of database systems, knowledge acquisition techniques and efficient deduction engines is an important frontier in machine learning and expert systems research. It is important that the integration should be done in a seamless fashion, so as to take full advantage of existing technology in each of the components. KEShell2 is a recent example of such an integrated system. However, the junction between the machine learning component (HCV) and the deduction engine was not efficient enough. KEShell2 used to put all the rules produced by HCV into a single rule body, with all attributes in the example set as schema factors. Section 4 shows how the single rule body can be split into a number of smaller rule bodies to improce the deduction efficiency. The schema extraction software built in this work bridges that gap. The software has been implemented in Prolog, and is easily integrated into the KEShell2 system.

The benefits of schema extraction are (i) to improve the efficiency of deduction in KEShell2, and (ii) to highlight semantic relationships between factors. The first benefit is easily quantified in terms of effort in the deduction engine. The second benefit is less easy to quantify, but has been briefly analysed in Section 4.

To date the schema extraction system has only been tested on the MONK's problems and the soybean data. The next step is to test the system on more large rule sets. In particular, the system will be tested on rule sets that define multi-step deductions. Of interest will be the ratio between the number of rules in a set and the number of schemas extracted. If the number of schemas is consistently much smaller than the number of rules, then the effort of schema extraction is justified in terms of deduction efficiency. The extent to which semantic information can be gathered from the grouping of rules into rule bodies needs further investigation.

6. References

1. Nieme T., Järvelin K. (1991), Prolog-Based Meta Rules for Relational Database Representation and Manipulation, *IEEE Transactions on Software Engineering* **17**(8), 762-788.
2. Wu X. (1993) Inductive Learning: Algorithms and Frontiers, *Artificial Intelligence Review* **7**(2), 93-108.
3. Michalski R.S., Chilausky R.L. (1980), 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**, 125-161.
4. Hong J. (1985), AE1: An Extension Matrix Approximate Method for the General Covering Problem, *International Journal of Computer and Information Sciences* **14**(6), 421-437.
5. Quinlan J.R. (1986), Induction of Decision Trees, *Machine Learning* **1**, 81-106.
6. Quinlan J.R. (1993), *C4.5: Programs for Machine Learning*, Morgan Kaufmann.
7. Cai Y., Cercone N., Han J. (1991), Learning in Relational Databases: An Attribute-Oriented Approach, *Computational Intelligence* **7**(3), 119-132.
8. Clark P., Niblett T. (1989), The CN2 Induction Algorithm, *Machine Learning* **3**, 261-283.
9. Ke M., Ali M. (1991), A Knowledge-Directed Induction Methodology for Intelligent Database Systems, *International Journal of Expert Systems* **4**(1), 71-115.
10. Wu X. (1993), The HCV Induction Algorithm, *Proceedings of the 21st ACM Computer Science Conference*, Kwasny S.C., Buck J. (Eds), ACM Press, 168-175.
11. Wu X (1992), KEShell2: An Intelligent Learning Data Base System, *Research and Development in Expert Systems IX*, Bramer K.A., Milne R.W. (Eds), Cambridge University Press, 253-272.
12. Wu X. (1991), KEshell: A "Rule Skeleton + Rule Body" Based Knowledge Engineering Shell, *Applications of Artificial Intelligence IX*, Trivedi M.M. (Ed.), SPIE Press, U.S.A., 632-639.
13. Thrun S.B., et al. (1991), The MONK's Problems - A Performance Comparison of Different Learning Algorithms, *CMU-CS-91-197*, School of Computer Science, Carnegie Mellon University.
14. Wu X. (1992), Rule Schema + Rule Body: A 2-Level Representation Language, *DAI Research Paper No. 579*, Department of Artificial Intelligence, University of Edinburgh. To appear in the International Journal of Computers and Their Applications.
15. Brownston L., Farrell R., Kant E., Martin N. (1985), *Programming Expert Systems in OPS5*, Addison-Wesley.
16. Harmon P., King D. (1985), *Expert Systems*, John Wiley & Sons, USA.
17. Wu X. (1993), LFA: A Linear Forward-chaining Algorithm for AI Production Systems, *Expert Systems: The International Journal of Knowledge Engineering* **10**(4), 237-242.
18. Wu X. (1994), SIKT: A Structured Interactive Knowledge Transfer Program, Submitted.
19. Davis R. (1982), Applications of Meta-Level Knowledge to the Construction, Maintenance and Use of Large Knowledge Bases, Ph.D. Thesis, *Technical Report Stan-CS-76-564*, Stanford University, 1976; Reprinted in *Knowledge-Based Systems in Artificial Intelligence*, Davis R., Lenat D.B. (Eds.), McGraw-Hill.
20. Wielinga B.J., Schreiber A.T., Breuker J.A. (1992), KADS: A Modelling Approach to Knowledge Engineering, *Knowledge Acquisition Journal* **4**(1), 5-53.

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