AN INTEGRATED APPROACH TO THE CONSTRUCTION OF KNOWLEDGE-BASED SYSTEMS: EXPERIENCE WITH ADVISE AND RELATED PROGRAMS

by

A. B. Baskin
R. S. Michalski

Topics in Expert System Design, G. Guida and C. Tasso (Eds.),
New York: North Holland, pp. 111-143, 1989.

Topics in Expert System Design
G. Guida and C. Tasso (Editors)
© Elsevier Science Publishers B.V. (North-Holland), 1989

AN INTEGRATED APPROACH TO THE CONSTRUCTION OF KNOWLEDGE-BASED SYSTEMS: EXPERIENCE WITH ADVISE AND RELATED PROGRAMS

Arthur B. BASKIN [°] **and Ryszard S. MICHALSKI** [°°]

Department of Computer Science [°]
University of Illinois
Urbana, IL, USA

Department of Computer Science [°°]
George Mason University
Fairfax, VA, USA

1. INTRODUCTION

Over the last few years, knowledge-based systems have clearly demonstrated the potential for substantial impact in a number of diverse areas. The successful construction of such knowledge-based systems has required that it be possible to encode sufficient domain expertise within a machine manipulable form to support intelligent problem solving behavior. A number of different approaches have been taken to the problems of knowledge acquisition, knowledge representation, uncertainty propagation and problem solving methodology. Choosing an architecture for a particular

The research program outlined in this paper has been supported by a number of sources. The primary support has come from the Office of Naval Research, N00014-92-K-0186, with additional support from DARPA, N00014-85-K-0878 and N00014-87-K-0874, and the Alcatel European Support Center's Artificial Intelligence Group, 85-ESC-5004.

expert system application requires not only domain knowledge but, also, the ability to match aspects of the problem domain with the characteristics of different competing paradigms for knowledge-based systems [1].

In proposing an integrated approach to the construction of knowledge-based systems, this paper presents the generalization of experience gained by designing and implementing the ADVISE system at the University of Illinois [2]. By integrating different, apparently competing, knowledge-based system techniques together with powerful machine learning techniques for knowledge acquisition and refinement, knowledge-based systems of a qualitatively different character can be constructed.

The emphasis in the ADVISE effort has been to develop a unified approach to the construction of knowledge-based systems incorporating multiple control schemes, multiple knowledge representations, different uncertainty propagation schemes, and extensive learning capabilities. The integrated approach described here goes beyond the degree of integration which was achieved with the actual implementation of ADVISE but is, none the less, grounded in the practical experience gained with that system.

The next section discusses two of the major design issues which influenced the original ADVISE effort and the current integrated approach as well. The discussion of an integrated approach to knowledge representation is followed by a discussion of the variety of machine learning tools and problem solving methods which can be brought to bear on knowledge organized in the form of networks, rules, and relational tables of data. The paper concludes with a description of an integrated language for knowledge-based systems called $KBVL_2$.

2. MATCHING THE PROBLEM TO THE SOLUTION

The advent of an increasing number of expert system shells for computers ranging from a personal computer to large main frames has produced an interesting phenomenon. In most cases, users of expert system shells are people with a solution in search of a suitable problem. That is, the knowledge representation and problem solving capabilities of the system are predefined by the choice of shell, and in that situation, the knowledge engineer is searching for a problem suited to the solution technique already chosen.

The knowledge structuring and manipulation operations supported by a given expert system shell limit or foster the description of domain knowledge for inclusion in the knowledge base as they clash or conform to the natural expression of the domain knowledge. For instance, when large numbers of facts and simple relationships between them characterize the domain, then posting and retrieving simple assertions on a blackboard can be quite natural. In other situations where the domain knowledge is best expressed as a non-deterministic exploration of a group of possibilities at each step in the solution, the concept of exploring *multiple worlds* can be much more concise. Finally, when causal or associational reasoning chains are known, then standard production rule systems are most natural.

Frequently, the enforced regimen of constructing a knowledge base is, itself, a contribution to the domain being studied. When the representation for the knowledge is ***cognitively close*** to the way that domain knowledge is already expressed and manipulated, then the construction of the knowledge base is simple, the validation of correctness is more likely, and the knowledge base can be used by itself as a ***distillation*** of knowledge in the domain. It is important to note that there is no such thing as a single ***best*** knowledge representation or problem solving paradigm. The different approaches have evolved as part of experience with different classes of problems and reflect a response to the requirements of different domains.

Matching solution technique to problem is made more complex by the cost and complexity of building the knowledge bases in complex domains. Once domain experts and knowledge engineers have been trained on a particular system, they are not generally able to use another system with differing primitives for knowledge structuring. It is as though each separate system is a different foreign language and retraining on radically different (and therefore potentially more suitable) approaches is simply not feasible. This problem is especially important because, in many cases, knowledge acquisition and refinement are the limiting factors in the construction of knowledge-based systems.

3. LIMITATIONS OF CURRENT EXPERT SYSTEMS

At present, the technology of expert systems is undergoing very rapid growth and is being applied to a wide spectrum of practical problems. Although the application of existing expert system techniques to practical problems can now produce useful expert systems, the current techniques have different limited areas of applicability and do not work together well.

Current expert systems suffer from a number of limitations that restrict their usefulness. They typically employ only one form of knowledge representation, have no learning capabilities, use only one type of inference procedure, employ only a single control strategy, and often do not deal with data or situations that are time-dependent [3] [4, 5]. Many of the techniques explored in today's expert system shells have been extensively studied and have known areas of applicability as well as known deficiencies. For a discussion of specific problems or limitations with individual techniques consult [1].

Limiting an expert system to both a single knowledge representation and a single control scheme is a common way to simplify the construction of the expert system. Unfortunately, different problem domains and even different aspects of a single domain may not be well suited to the same knowledge representation or the same problem solving control scheme. Limitations in an expert system technique which do not correspond to requirements of a particular domain do not hinder the operation of the system, but subproblems with different requirements will frequently not fit a single paradigm.

Single paradigm approaches impose their particular assumptions about the nature of the data, the knowledge representation, and the problem solving in the given domain. Assumptions about the nature of the data commonly made include assumptions about:

- distribution of training examples in the space of possible situations
- conditional dependence/independence of individual findings
- time dependence/independence of data
- ambiguity, reproducibility, completeness, and redundancy of data
- ability to influence data collection
- level of specificity, timeliness, and reliability of the data
- data and situations do not change during the consultation.

Complex domains which are uniformly structured according to one version of all of the assumptions listed above are quite rare. All too often, portions of the problem solving process or areas of the domain will require conflicting assumptions.

Current expert system shells make a number of assumptions about the nature of the knowledge to represent and the best approach to that representation. In some cases, blackboards form ideal models for communication between cooperating problem solving processes but do not form good representations for intermediate decision making. Some domains are data driven and easily organized for constraint propagation or spreading activation; other domains may contain procedural knowledge which is properly represented by rules, while still others are underconstrained and benefit from a hypothesize and test approach.

Domains which are highly goal directed or diagnostic in nature are well suited to backward chaining while domains such as mechanical component design are much less well structured and require some construction of the goal itself. Explicit representation for some control information is also a requirement not always met by current techniques. Quite often in industrial design applications (such as switching system configuration), a domain expert will know not only detailed information about the design process, but also general approaches and precedence relations defined over data collection, problem decomposition, and candidate solutions to try before conducting a search. This strategic information needs to be separated from the more tactical information normally incorporated in non-deterministically scheduled rulebases so as not to loose the strategic information in a sea of tactical rules for intermediate decision making.

Knowledge acquisition is perhaps the most important part of the construction of any intelligent system. The rate at which domain knowledge can be captured is directly limited by the suitability of the knowledge representation primitives to the domain and to the level of detail at which the domain knowledge is specified. Generally, high level non-deterministic descriptions of the desired problem solving process are the most concise to acquire. Knowledge acquisition that depends on ***learning by instruction*** has been a major factor in the development of expert systems to date. Such an approach can be extremely limiting when general heuristics are not known or domain expert time is at a premium. Machine learning techniques can be used to reduce the dependence on pre-digestion of the problem solving process by domain experts and to refine knowledge bases regardless of their source.

4. INTEGRATED APPROACH TO KNOWLEDGE-BASED SYSTEMS

The expert system shells currently in the marketplace are predominantly built upon the expert systems research of the middle to late 1970s. More recent attempts to build ***knowledge-based systems*** reflect the need for even more knowledge intensive paradigms for constructing intelligent computational models of real world expertise. Thus, while current knowledge-based system research is grounded in earlier work in expert systems, it should not be considered as merely an extension of the that earlier work.

There have been a number of efforts to develop special purpose knowledge representations and/or problem solving strategies for knowledge-based systems. Rather than attempt to develop another radically different approach, the ADVISE effort centered around using a variety of existing techniques within a single knowledge-based system paradigm. Thus, the knowledge representation and problem solving strategies could be selected from among the range supported by the system.

Most knowledge-based systems emphasize the deductive application of the knowledge base to a presenting problem in order to produce a result. Our efforts to develop an integrated approach are different in that they have emphasized the incorporation of machine learning techniques to build and modify the knowledge base.

Figure 1 shows the three knowledge representation techniques supported in our unified approach and the inference operations defined over them. The arrows represent inferential transformations which operate on one representation and produce another. Thus, the traditional expert system operation is depicted as a deductive application arrow from the rulebase to a particular database element (a database tuple consisting of the presenting data and the deductively produced result). Similarly, the arc from data to rules represents the learning from examples arc and depicts the inductive generalization of rules from examples of the decision. The remainder of this paper consists of more detailed presentation of information about each component of the integrated approach to knowledge-based systems portrayed in Figure 1.

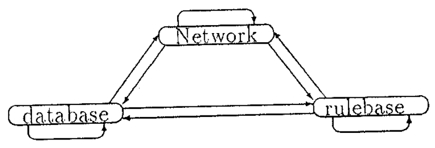

FIGURE 1
Knowledge representation alternatives and transformation operations linking them.

Each arc in Figure 1 corresponds to a transformation operation which takes in one form of knowledge from the knowledge base and produces a potentially different knowledge representation. Some of the transformations shown in Figure 1 correspond to inductive learning operations and some correspond to deductive application of the knowledge base for problem solving. A few arcs such as the one from database to database can be both inductive learning and deductive problem solving. The meaning of the various arcs is briefly outlined in the remainder of this section and described in greater detail in sections 6 and 7 below.

4.1. Learning Transformation Operations

Figure 2 shows the subset of the transformation arcs from Figure 1 which represent learning components of the integrated approach. Each of these arcs corresponds to inductive learning operators which operate on a portion of the knowledge base to inductively derive revised or additional knowledge base contents. The learning transformation operations are:

database ⟶ **database** — used to reduce database complexity by selecting most representative data examples (called events) and/or by eliminating unnecessary attributes from further consideration;

network ⟶ **network** — used to reason by analogy and produce a revised or extended network structure from an existing network structure;

rulebase ⟶ **rulebase** — used to derive improved or extended rules from inefficient or overly restrictive rules;

database ⟶ **network** — used to organize data elements together into groups (conceptual clusters) and to derive hierarchies of such groups;

rulebase ⟶ **network** — used to organize the rules together into groups and to derive hierarchies of such groups;

database ⟶ **rulebase** — used to derive rules which summarize the decision behavior implicit in a set of examples taken from the database.

Among the learning transformation operations shown in Figure 2, the learning of rules from examples has received the most attention and is the most developed.

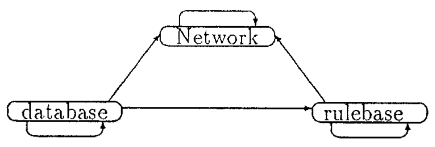

FIGURE 2
Transformation operations which involve learning.

4.2. Problem Solving Transformation Operations

Figure 3 shows the subset of the transformation arcs from Figure 1 which represents the use of portions of the knowledge base for problem solving. Each of these transformations corresponds to one or more (usually deductive) operators which applies the general knowledge in the knowledge base to a specific situation. The problem solving transformations are:

database $\longrightarrow$ **database** — direct retrieval of a solution from the fact base;

network $\longrightarrow$ **network** — plausible reasoning and/or reasoning by analogy to determine network relationships which hold for the present problem;

rulebase $\longrightarrow$ **rulebase** — rule-based reasoning about the control over which rules to use and in which order to pursue them;

network $\longrightarrow$ **rulebase** — network-based reasoning about the control over which rules to use and in which order to pursue them;

rulebase $\longrightarrow$ **database** — the application of the rulebase to the situation at hand to define solution (and, thus, an event in the database).

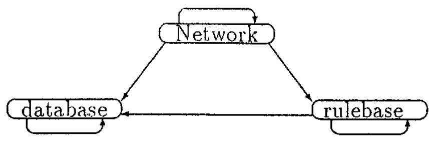

FIGURE 3
Transformation operations for problem solving.

Before discussing the transformation operations shown in the figures above in greater detail, additional information about the various knowledge representations must be presented.

5. MULTIPLE KNOWLEDGE REPRESENTATIONS

In our integrated approach to knowledge representation, we identified three different knowledge representation techniques, each well suited to a different role:

- networks of objects
- relational database of facts
- and a rulebase for more procedural information.

A unified underlying access protocol is used to support these conceptually different forms of knowledge. The knowledge base is structured into segments for ease of management and efficiency of some inference operations. A segment is merely a subset of the knowledge base and can either contain only information of one type or can span all three types of representation. Finally, a segment can itself contain segments to as many levels as needed.

The network of objects is used to contain strategic information about the domain and is used for planning during problem solving. The relational database stores facts or assertions about the problem at hand and facts about the domain or about specific decisions (usually examples supplied by a human expert). Finally, the rulebase is used to store detailed procedural information about the domain and the solution to problems.

5.1. Networks of Objects

The network component of the knowledge base consists of network links between objects used to capture general domain knowledge about interrelationships among various conceptual units. For example, it can include hierarchies of attributes from the application domain indicating the level of generality of such attributes (a *generalization tree*), a representation of the structure of the rulebase, and precedence relations defined over queries to the user. Links between nodes in the network represent *static* relationships between concepts. The network organization is a form of the *Logic Net* formalism described in [6].

Each conceptual unit in the knowledge base behaves like an object in an object oriented programming environment. Thus, each object can have attached attributes and procedures. The networks of objects are used to capture not only the traditional object oriented inheritance hierarchies but also knowledge engineer designed relationships between concepts.

In combination with the notion of knowledge base segments, which can themselves be conceptual objects, the networks of objects portion of the knowledge base provides a powerful structuring mechanism. Rules can be gathered together into segments and structured as networks; individual findings can be associated together in a network with inference defined over the links, and structure within the range of values of a variable or among a group of variables can be captured as a generalization tree.

Although the original ADVISE system, on which this integrated approach is based, only supported construction of static network structures, the generalization to dynamic structures is quite desirable. The concept of *worlds* common to a number of expert system shells amounts to the dynamic management of subsets of facts in the knowledge base. In a similar manner, the dynamic management of segments containing rules is a way to provide for adjustment of the knowledge base to changing situations or a predictable temporal evolution.

5.1.1. Instance Variables

A knowledge engineer uses predefined objects and knowledge engineer defined objects in describing a domain knowledge to the system. The instance variables of an object behave like the slots in a frame and the instance variable inheritance is similar to slot inheritance in a frame hierarchy. When defining an object, the knowledge engineer can start with the predefined *object* which contains nothing and buid an object entirely to suite the situation at hand, or the knowledge engineer can build upon the predefined object classes.

An instance variable of an object may be a full object itself or simply a value. A simple value behaves like an object with a single instance variable called *value*. Predefined object classes for variables used in logic rules such as *integer, real*, and *string* have instance variables used to contain: a prompt string, a default value, whether or not the variable is askable, the range of values the variable is allowed, and the units with which the variable is measured. These properties may be modified or extended by changing the object oriented *core* of the system.

5.1.2. Methods

Unlike most object oriented systems where the methods on objects are specified in some programming language, the methods on objects are here defined using the same rule format used to define the segments of rules in the rulebase. The methods are deterministically scheduled rules which are investigated in the order in which they occur in the method. These logic rules may modify instance variables and they may also modify knowledge engineer defined variables which are also used by the rules in the rulebase segments.

A number of predefined methods exist for uncertainty management, rule scheduling in the rule segments, data collection, and presentation of results. These methods may be modified or extended by defining new methods. In general, the predefined methods provide higher performance than the knowledge engineer defined methods because they can be *pre-compiled* into the system rather than being interpretively executed.

5.1.3. Using Networks of Objects

The simplest network of objects used in the system is the inheritance hierarchy for the classes of objects defined within the system. This hierarchy provides for default structure of objects, default methods, and default values for individual instance variables. In addition to the predefined inheritance hierarchy, the knowledge engineer can define additional networks for use in the problem solving process. For instance, a goal tree can be defined to capture a static problem decomposition and a method can be defined to traverse the tree in a case specific order. A knowledge engineer can also define a specificity hierarchy of attributes where such a structure exists for the domain or can be derived using the learning techniques discussed in the next section. Separate specificity hierarchies can be used for attributes themselves and for structure within the range of values of an attribute.

Network-based problem solving can take the form of *spreading activation* where changes in values or certainties are propagated along network links. Methods attached to network segments control this form of problem solving in a similar way to the rule scheduling methods attached to rulebase segments. In general, the network portion of the knowledge base is best suited to describing general properties or organizational principles of the domain and not detailed problem solving. The network structures can provide partial orderings of goals to pursue and/or data to solicit.

5.2. Rule base

The rulebase is the most commonly used knowledge structuring for knowledge-based systems. It is best suited to capturing tactical information. Individual rules or groups of rules can be used to answer specific questions or differentiate among a small number of alternatives.

5.2.1. Generalized Decision Rules

The rule format supported by the ADVISE system is based on the GVL rule syntax developed by Michalski [7]. As with most decision rules, the rules are divided into condition/action pairs. The condition can be a disjunction of multiple conjunctions of constraints on attribute values. The action portion of a rule contains one or more assignments of values to attributes. The rulebase consists of rules of the following basic form:

$$\text{CTX: CONDITION} \Rightarrow \text{CONCLUSION} : \alpha, \beta$$

where:

CTX is an experssion describing the context within which the rule is applicable;

CONDITION is a formal expression (in VL_2 [8]) which involves elementary conditional statements (called *selectors*), linked by various logic operators (including quantifiers);

CONCLUSION defines the decision or action which is executed when the CONDITION is satisfied by a given situation;

α is the strength of the evidence which supports the CONCLUSION when the CONDITION is completely satisfied ($0 \leq \alpha \leq 1$) and

β is the strength of the evidence which supports the CONDITION when the CONCLUSION is completely satisfied ($0 \leq \beta \leq 1$).

The rule above is read: In the context CTX, CONDITION implies CONCLUSION with forward strength "α" and backward strength "β". Specifically, the rule states that: if the context and the left hand side (LHS) of the decision assignment operator ($\Rightarrow$) are satisfied, then the right hand side (RHS) is asserted with a degree of confidence α, and if the context and the RHS are both satisfied, then the LHS is asserted with a degree of confidence β. The decision assignment operator is equivalent to logical implication when $\alpha = 1$ and $\beta = 0$, and is equivalent to logical equivalence when $\alpha = 1$ and $\beta = 1$. Thus, by providing both "α" and "β" for each rule, it is possible to use rules in both forward and backward directions.

5.2.2. Application of Reversible Rules to Design Problems

Reversible logic rules pose some problems in that they necessarily support the same logical operations on both sides of the decision assignment operator. When nested disjunctive operations are supported in both the CONDITION and the CONCLUSION some form of *world* management is required. Despite these difficulties, reversible rules support a very important class of problems as can be illustrated briefly here.

Particularly in mechanical design, there are deep models and mathematical simulation programs which can predict an outcome from specified design parameters. For instance, the stress or wear on a given metal can be predicted from a simulation of the known behavior of the metal in a given environment. Such outcome knowledge can be captured in a knowledge-based system as, for instance, production rules which specify α to a great degree of precision. Unfortunately, design problems rarely present in this form. All too often, the design problem takes the form of a desired outcome (say lack of metal fatigue) and the problem is to find the design parameters which lead to that conclusion. Once those parameters have been found, they can be *validated* with the *forward* logic.

The reversible rules described above directly address the design problem. In particular, the rules are initially constructed from the deep knowledge to run in the *forward* direction. Later the β values are added and the rules are actually used in the *reverse* direction to solve a design problem. Once a candidate solution has been obtained, it can be evaluated by using the same rules in the *forward* direction. It should be noted that not all rules are reversible (as signified by either $\alpha = 0$ or $\beta = 0$) and that the definitions of *forward* and *backward*, while they must be used consistently for a given domain, are arbitrary.

5.2.3. Options for Uncertainty Management

The term *strength of evidence* used above was intentionally vague. In much the same way that the original ADVISE system sought to solve the problem of choosing the correct knowledge representation by supporting multiple representations which could be chosen based on the problem, it also supported multiple mechanisms for uncertainty propagation. Thus, both the CONDITION and CONCLUSION portions of rules allowed several different strategies to be used to propagate uncertainty. Each attribute (similar to an assertion in a blackboard system) to which a value can be assigned can also be assigned a strength of evidence supporting that value. In the case that there are multiple values possible, each may have its own strength of supporting evidence. The strengths of evidence are all assumed to increase with increasing evidence and decrease with decreasing evidence; only the Bayesian control scheme uses a strict probabilistic interpretation of strengths of evidence. By choosing among the predefined methods for combining strengths of evidence (or writing new methods), the knowledge engineer can utilize a range of uncertainty managements including probabilistic, MYCIN-like certainty factors, and normalized/unnormalized fuzzy logics.

Each constraint on an attribute within a rule (called a selector) can be parameterized with an α-weight and a β-weight analogous to the α and β of an entire rule. The weights indicate the importance of the attributes to the truth or falsity respectively of the term in which they occur. The weighted strength of evidence of each constraint may be linearly combined with the weighted certainty of other selectors to form a linear module. Individual selectors or linear modules may be *anded* or *ored* together in any combination to form the strength of evidence of the condition.

The combining of the strengths of evidence for a variable with the weight on the selector and the subsequent folding together of the weighted strengths of other selectors can be done using any of several different uncertainty propagation schemes. The conjunction of strengths of evidence may consist of maximum, average, probabilistic sum or Bayesian updating of strengths. Similarly, disjunction may be minimum or average. The uncertainty propagation scheme may be changed for every rule within the knowledge base, but usually all rules within a single knowledge base segment will use the same uncertainty propagation scheme.

In general, the difficulty with utilizing different uncertainty propagation mechanisms within a single system lies in combining strengths of evidence arising from different sources and calculated using different assumptions. Unfortunately, there is no simple solution to this problem. The problem can be mitigated by using homogeneous propagation mechanisms within a knowledge base segment and, thus, minimizing the exchange of strengths arising from different assumptions. For instance, a knowledge base segment might be used to derive a rank ordered list of candidate hypotheses for investigation by another segment which utilizes the partial ordering and not the actual strengths of evidence. In such a case, there is no need for the uncertainty management to be the same in the two segments.

5.3. Relational Database of Facts

The relational database contains relational tables which represent any factual information, e.g., examples of experts' past decisions. Also, the operation of the deductive inference component of the system essentially produces additional examples which can be stored in the relational tables — these are results of the operation of the machine-based expert. A modified relational algebra has been developed using constructs from Variable-valued Logic in order to integrate database operations concisely into the knowledge base formalism.

The relational database serves as the starting point for all inductive learning from examples. In many ways, the relational database of facts forms the bridge between the inductive learning process and the largely deductive problem solving process. Logic-based relational operations can be performed interactively by a knowledge engineer to control knowledge acquisition and refinement programs, or these same operations can be controlled from within logic rules during problem solving.

5.3.1. Table Creation

Relational tables can be constructed as part of batch loading of a knowledge base, operation of rules during problem solving, and interactive knowledge acquisition from a human expert. In any case, the table must be named and the attributes (columns) of the table must be defined. In addition, *key* attributes may be specified along with whether or not the table is normalized. The key attributes may be used to select rows from the table and to sort entries. A normalized table requires that all rows (corresponding to events or examples) must have unique values for the keys for that row. An unnormalized table by have duplicate values for the key fields in two or more rows.

The predefined object class, *table*, can be used to define a new table. Either a new class of table can be defined by adding properties to the system *table* or an instance of the predefined table can be created. For example, the table for holding information on blood tests might contain information about the specimens and laboratory result such as:

```
labvals : table (spec#, MCV : integer; Hgb : real; RBC_morph: string) key = spec#
```

where the spec# is the specimen number used to retrieve the results for the test and the additional attributes represent the mean corpuscular volume, hemoglobin, and red cell morphology respectively.

The ***table, integer, real, and string*** objects used in the table declarations above are not just simple data types. They are predefined object class definitions which the knowledge engineer can modify or extend. For instance, each has default methods for soliciting values from the user and a default prompt string to be used when requesting a value. The ***key*** instance variable is an instance variable of the object ***table*** which is used to contain the set of attribute names which are keys for the table. It is initialized to contain one attribute in the example above.

5.3.2. Operations on Tables

The system table object comes with a number of default methods which operate on the tables and behave as simple operators. These operators include the following editing operations: **add** — appends a new row onto a table, **change** — modifies selected rows in a table, **delete** — removes the selected rows from a table, and the following relational data operations:

union — the union of two relational tables is a table containing all of the rows (without duplication) from the two input tables. The input tables for union, intersection and difference must have ***identical attribute lists.***

intersect — the intersection of two tables is a table containing only the rows common to the two tables.

difference — the difference of two tables is a table made up of the rows of the first table which are not also rows of the second table.

product — the Cartesian product of two tables is a table made up of the concatenation of each row from the first table with all of the rows of the second table.

select — a selection from one input table is a table formed from the input table by applying a logical constraint rule to each row in the input table. Only rows which meet the logical constraint are included in the output table.

project — the projection of a table is a table containing only the indicated columns (with duplicate rows removed).

join — the join of two or more tables is a table containing a row for each row containing identical values for the shared attributes among the tables. The resultant row is formed by concatenating the input rows and using only one copy of the shared attributes. When the input tables share no attributes, then the result is the same as for the Cartesian product of the input tables.

The operations listed above can be used to manipulate tables of data either for problem solving or for submission to various learning operations which use tables. In either case, the operations can be incorporated into the rulebase or used interactively by the content expert in building or refining the knowledge base.

6. Learning Capabilities

Inferential learning techniques (largely inductive) can be used for knowledge acquisition and refinement. The inclusion of machine-based inference as a part of the knowledge acquisition process is intended to reduce the burden on human experts who would otherwise have to directly encode the contents of the knowledge base. By defining inference procedures over each component of the knowledge base, the system no longer relies on the human expert to organize and present a complete, concise, and error free knowledge base. In much the same way that the relational table operators have been incorporated into the rule formalism as operators which can operate directly on the knowledge base, the learning operators can be interactively invoked or invoked from within rules during problem solving.

Recent progress in the area of machine learning [9] opens a number of possibilities for improving knowledge acquisition methods for knowledge-based systems. Various experiments have demonstrated that it is possible to learn the decision rules from examples of experts' decisions, and that these rules perform as well, and sometimes even better than rules obtained by encoding the experts' rules e.g., [10].

Important progress has been made recently in studying other machine learning strategies, such as learning by analogy [11, 12, 13] and learning from observation and discovery [14, 15]. These findings indicate possibilities for developing new techniques for knowledge acquisition for knowledge based systems using learning techniques.

The discussion of learning capabilities for integrated knowledge-based system building below is organized according to the transformations from Figure 2. Additional details about each category of transformation are provided below. In principal, there is no limit to the number and variety of inferential operators that might be included in this integrated approach. The discussion below has been limited to a meaningful length by only including those inferential operations which we have explicitly explored rather than all possibilities.

6.1. From Data to Rules – Learning from Examples

Inductive derivation of rules from expert supplied sets of examples is one of the most classic applications of machine learning to knowledge acquisition as demonstrated by such programs as AQVAL [16], INDUCE [17], and ID3 [18]. Such programs are usually used to ***differentiate*** among outcomes and produce rules which summarize tables of examples of the desired decision. In our work, we have investigated rule inference in three different areas: attribute-based decision rules – differentiate, structural descriptions – structure, temporal process prediction – predict. Each of these efforts has lead to an operator in the integrated approach.

6.1.1. Differentiate

We have explored the differentiate operation using the programs GEM, and AQ11 [19]. The GEM program takes in two or more relational tables each containing a set of examples of a decision class and produces as output a set of consistent and optimized decision rules expressed as statements in the *attribute calculus* known as VL_1 (Variable-valued Logic system 1) [16]. The output of the induction process is a collection of decision rules which can be stored in the knowledge base. The GEM operator can be used to interactively generate optimized decision rules when only examples of the decisions are well known. It has been used to generate soybean diagnosis rules from examples. In that situation, the generated rules outperformed the rules which were hand-crafted from expert knowledge [20]. The AQ11 and AQ15 programs take a number of event classes and attempt to find the conceptually simplest rules that will determine the class of each event. These programs use the same basic *Aq* algorithm which is used in GEM but support a more extended knowledge representation than that supported by GEM.

6.1.2. Structure

The program INDUCE/3 is able to learn structural descriptions of classes of objects from examples. It can solve learning problems similar to those solved by GEM, but unlike GEM, it processes structured examples described in an extended predicate calculus (system VL_2). The program incorporates an inference mechanism of *constructive induction* for applying background knowledge rules to examples to produce new descriptors. The INDUCE program can be used to generate decision rules when examples of the decisions are available and the examples have internal structure.

6.1.3. Predict

The SPARC family of programs generate rules describing processes which evolve over time (temporal process prediction). Examples of temporal processes can be represented in a conceptual network, or in *Annotated Predicate Calculus*. In previous research [21], the program was used to play the card game Eleusis in which the goal is to discover a card sequence prediction rule to direct the play of the cards. The program has recently been extended to solve a more general problem of process prediction.

6.2. From Data to Data – Event and Attribute Selection

Transformations of tables of data elements into other tables of data elements are common operations during knowledge acquisition and refinement. The relational table operations such as project, join, and select can be used to build new tables from existing tables. Also, a table can be extended or modified to reflect new information. In addition to these interactive operations, there are two inductive operations we have investigated: attribute (variable) selection – VARSEL, event (example) selection – ESEL. If the table of examples is organized with the attributes across the top as column headers and the examples themselves as rows, then these operations correspond to selecting a subset of the columns or rows for further use. Selecting a subset of the available examples can be particularly important for time saving with computationally intensive learning algorithms.

6.2.1. Attribute Selection

The VARSEL operation invokes the program called PROMISE [22] which selects the most *promising* attributes for differentiating between classes of events. Its output is therefore intended for use with the Differentiate operation described above.

The VARSEL operation takes two or more relational tables each containing examples of a particular class and selects a projection of the tables which consists of relevant selected variables. A variable is relevant if its values contribute significantly to differentiating the examples in one class from those in another class. The output consists of tables containing the same examples as the input, but with columns of values for only the relevant attributes. The VARSEL operator is used to reduce the complexity of example sets in the relational database by removing attributes which are irrelevant.

6.2.2. Event Selection

The ESEL operation invokes the program ESEL/2 [23], a program that takes a large number of examples and selects a small subset of examples that is most representative of the larger group. The smaller sample will require less computation when used as input to other inductive operators. Large numbers of examples (more than 200) can require substantial processing time when used as input to other inference operators.

The ESEL operator takes one or more relational tables containing examples and selects a subset of examples which are most representative of the set. Several different selection algorithms are available, based on conceptual measures similar to the conceptual cohesiveness measure used by the CLUSTER operator (described below). The ESEL operator is used to scale down large example sets while trying to retain the embedded knowledge.

6.3. From Rules to Rules – Incremental Rule Refinement

In our experiments, we have not developed separate tools devoted to incremental rule refinement, but we have used other existing tools to meet this need. We have identified two different cases for rule refinement: Refinement based on experience – Refine, Refinement from consistency checking – Consistency. Each of these cases involves taking in a segment of rules and producing a revised segment of rules that are more consistent or more general.

6.3.1. Refine

When additional examples are provided or the results of problem solving are critiqued, errors in the problem solving behavior of the system can be detected. Ignoring for the moment the substantial problem of blame assignment, rules in the knowledge base can be modified with special exception clauses which specifically exclude the known counter examples from the more general terms in the rules. This form of knowledge base update can be performed by storing a table of exceptions to check first or by modifying individual rules.

As rules accumulate exception clauses, they become cumbersome to manipulate and are less likely to fully represent the structure of the original problem. Either the original training set of examples can be modified to include the new examples for an entirely new inductive learning process, or the rules can be incrementally modified. Incremental modification of rules preserves as much of the previous solution as possible while improving the performance, understandability, and predictive value of the refined rules. (The INDUCE operation provides limited support for this function.)

6.3.2. Consistency

A knowledge base can be refined by analyzing it for consistency and completeness [24]. A segment of rules can be inspected for its coverage of the full range of input values (completeness) or for the uniqueness of its reported result for any given input pattern (consistency). When inconsistencies or omissions are detected, additional rules can be generated to force the supplied rule segment to be either complete or consistent. Usually, such rules will need to be inspected by an expert with domain knowledge to evaluate the suitability of including the inductively derived rules into the knowledge base.

The consistency and completeness operations are combined in a single program which is invoked by the consistency operator [24]. This operator takes a rule segment and returns a (potentially) modified rule segment where rules have been modified or added to make the knowledge base more complete or consistent. The rulebase is made complete by simply constructing a rule (or rules) which explicitly enumerate the incompleteness by explicitly assigning the decision variable(s) to undefined. Rules are added or modified to produce a consistent rulebase by eliminating competing rule chains that produce conflicting results.

6.4. From Rules to Networks – Completeness

The completeness operator described above is also capable of detecting structure within a rulebase. The operator groups rules together into functional groups which share access to common variables and participate in common rule chains. The result is similar to that produced by Jacob and Froscher [26] but is based upon the intermediate results of the consistency analysis. The grouping of the rules is structured as segments defined within the original segment and these segments are organized in a tree structure using the network of object primitives.

6.5. From Data to Networks – Conceptual Clustering

The purpose of the ***cluster*** operation is to divide a collection of objects into smaller groups of similar objects based upon some criterion or measure of similarity. Clustering is the process of developing a taxonomy or classification scheme for the objects of a study. The program invoked by the ***cluster*** command is called *CLSUTER/paf* [25]. Unlike most numerical taxonomic techniques, this program uses a ***concept-based*** method of clustering that produces descriptions of the clusters (categories) that it derives. It also permits the user to specify the criteria which are to be used to evaluate clusters.

The ***cluster*** operation takes one relational table containing examples and automatically builds classifications of the examples. A classification is a hierarchy of conjunctive concepts expressed in the ***attribute calculus*** of VL_1. A more powerful extension of the operator invokes the CLUSTER/S program which accepts a collection of structured objects and automatically builds classifications composed of conjunctive statments in the extended predicate calculus (VL_2). The CLUSTER operator is used to organize and give conceptual structure to a collection of examples. This can improve ease and speed of access to the data, the efficiency of subsequent inferential operations, and can reveal to the domain expert underlying patterns.

6.6. From Networks to Networks

In the work on ADVISE, we have not explored machine learning as applied to refining network structures from networks. Michalski and Collins, however, have explored the use of transformation operators which operate to produce a form of plausible reasoning [27]. The application of the theory of plausible reasoning they have outlined can produce organizations of new material based on reasoning over existing knowledge. Thus, a specificity hierarchy can serve to guide the construction of a similar hierarchy using transformation operators such as generalization, specialization, similarity and dissimilarity as applied to both attributes and their values.

7. Problem Solving Using Multiple Representations

The reason for supporting an integrated approach to problem solving over multiple knowledge representations is to allow the selection of the knowledge representation which most suits the problem domain. By choosing the knowledge representation and associated problem solving technique most natural for a problem or subproblem, the knowledge engineer can minimize the distance between the way that information about the problem is presented in the outside world and recorded in the knowledge base.

This section explores the problem solving transformations shown in Figure 2. In most cases, we have explored these problem solving strategies as separate components of the ADVISE system. A brief outline of problem solving over each knowledge representation is presented in the subsections which follow. Additional information about some of the problem solving methods is presented in the example applications in the next section.

7.1. From Data to Data – Deductive and Inductive Retrieval

Deductive retrieval represents the simplest and most straight forward method for conducting problem solving (unfortunately it is seldom sufficient). The relational commands supported over the knowledge base provide for simple deductive retrieval of examples. When an example in the knowledge base (such as the exceptions discussed under learning from rules to rules) is directly retrieved based on a match between the presenting information and the present case, the stored decision can be used. Although direct retrieval is not a good general purpose way to implement a machine-based problem solver, it can be used to encode exceptional, prototypical, and exemplary cases for use in problem solving, generalization, and explanation.

The deductive retrieval process can be implemented as part of the problem solving process, e.g. a preamble to more traditional machine-based reasoning techniques or as an end in itself. If deductive retrieval is included, then it is quite natural to provide for relaxed criteria matching on the retrieval and, thus, support inductive retrieval. In the case of inductive retrieval, other than a perfect match between the presenting data and the retrieved data can be used to generate a decision. The amount of the generalization allowed in the inductive retrieval can be controlled and used to decide whether additional methods should be used to verify a given result. When the results of the inductive retrieval are themselves stored back in the knowledge base, then a form of incremental learning results.

7.2. From Rules to Data – Deductive Rule Application

Deductive rule application is the most common form of problem solving for expert systems. In our experiments with ADVISE, we have investigated the use of multiple control schemes for scheduling rules and multiple mechanisms for propagating uncertainty. Reinke [28] empirically investigated the role of uncertainty propagation mechanisms within the ADVISE approach and constructed tools for executing ADVISE rules in batch mode for analysis.

The notion of a rulegroup in the ADVISE system has been generalized to a knowledge base segment in this integrated approach. A rule group forms a unit of uniform scheduling and uncertainty propagation scheme and usually corresponds to a subproblem in a task decomposition. Support for multiple schemes for conducting deductive inference over the rules allows the problem solving to be tailored to the problem domain.

7.3. From Rules to Rules – Rule-based Scheduling

Reasoning from rules to rules is equivalent to reasoning about the control of rule exploration using rules. In our integrated approach, we have used the concept of a rulegroup (here generalized to a knowledge base segment) as a rulebase structuring element. An entire rule segment appears like an encapsulated logic operation which has a similar external structure to that of a rule. Thus, an entire rule segment can be invoked in a rule of another segment much like a function call. The acquisition of the value of the function is scheduled in the same way that scalar attribute collection is scheduled (e.g. based on cost, frequency, strength of implication, etc.).

In its most basic form, an invocation of a rule segment from within a rule segment looks like a single rule which summarizes the behavior of the the invoked rule segment. This use of rule segments and their encapsulation can provide a simplification of knowledge base management similar to that afforded traditional programs by equivalent software engineering techniques [26].

7.4. From Networks to Data – Network Problem Solving

Although experiments with purely network-based problem solving were conducted with the BABY system (described below), true network-based problem solving was never incorporated into the ADVISE system. Systems such as AL/X and PROSPECTOR in addition to numerous efforts involving spreading activation indicate that such techniques should be part of the integrated approach.

When reasoning from networks to data, the problem solving strategy consists of a traversal of the network in order to evaluate the possible conclusions in light of the prevailing evidence. The acquisition of evidence can be controlled by the topology of the network linkages and sequential Bayesian updating of probabilities or analogous operations on certainty factors of various kinds can support reasoning over uncertain evidence.

7.5. From Networks to Rules – Network-based Scheduling

Reasoning from networks to rules is similar to reasoning from rules to rules in that it amounts to a form of rule scheduling. This option for rule scheduling was used extensively in the BABY system described in the next section to support an evolution of the reasoning process.

Spreading activation or sequential Bayesian traversal of the network can be used to evaluate what amount to preconditions on the invocation of rule segments or individual rules. In either case, rule sequencing information has been included in the network. This approach is particularly well suited to situations where data arrives naturally in the course of operation of the system and does not need to be solicited. Temporal progressions or developing situations can be described by the network and appropriate rules to react to the evolving situation can be activated by the the individual nodes in the network.

7.6. From Networks to Networks – Plausible Reasoning

Reasoning from networks to networks amounts to reasoning from known structure of the problem domain (or the problem at hand) to a new structure of the problem at hand in light of available evidence. Such reasoning can take at least two forms: reasoning by analogy and responding to changing situations. These two distinct problem solving activities can share a common knowledge representation and many of the operations over that representation. Reasoning by analogy as a part of plausible reasoning was discussed in the section on inference of networks from networks above and is more fully discussed in [27].

When reasoning from networks to networks is used to respond to changing situations, network descriptions of the structure of the various possible situations must be constructed. These descriptions serve to guide the instantiation of sub-networks which provide detailed descriptions of the evolving situations.

8. Experiments Using the Integrated Approach

The range of applicability of many of the ideas presented in this integrated approach can be seen from three example knowledge-based systems which were developed in the ADVISE approach and an application of the ESSAI system (which incorporates many ideas from the integrated approach). The system which was most fully developed was the PLANT/ds which serves as a consultant for soybean disease diagnosis. The PLANT/cd system predicts the cutworm damage to corn and the BABY system was developed as a consultant for the neo-natal intensive care unit. The ESSAI system, derived from the original ADVISE system, was applied to the problem of configuring a System 12 switching system. Each of these systems made use of different knowledge structuring primitives and different problem solving strategies.

8.1. PLANT/ds

The PLANT/ds system uses data about the condition of plants in a soybean field to predict which of the 19 most common disease(s) may be present [29]. The questions the system asks are organized in a collection of forms which allow many related questions to be answered at one time. The selection of the forms to present to the user is determined by evaluating the *utility* of all of the variables for which a value is not known. The form containing the more useful variable is then presented to the user and the form is filled out. The evaluation of the utility measure for the next variable to select uses all of the data volunteered on the previous forms in evaluating variables and rules.

The PLANT/ds control scheme has been used with expert derived rules and rules derived from machine induction. A version of PLANT was developed that used both expert and machine rules to arrive at a diagnosis. In addition, the utility measure control scheme was also used in a project to develop a turf management expert system using expert rules.

8.2. PLANT/cd

The PLANT/cd expert system uses data about insects and corn plants to predict the damage to corn due to cutworms [30]. The control scheme for PLANT/cd is a traditional backward chaining theorem proving model. Rules, and, finally variables are selected for evaluation based on the potential strength of implication of the rules. The PLANT/cd system is disjunctive because of its use of an external climatologic database and a deep model of insect growth. These two sources of information external to the expert system itself were consulted in much the same way that the human user was asked a question. Because of the nature of backward chaining, questions were asked serially and there was limited ability for the user to volunteer data.

8.3. BABY

The BABY system is a consultant that is intended to monitor on-line data, laboratory data, and clinical data in a neo-natal intensive care unit [31]. (The system used transaction files to simulate direct connection to a premature infant.) Data is supplied to BABY as it is generated and the majority of the data input into the system comes automatically. Clinical findings which a doctor might specify are requested by the system when they would be important for a diagnosis. Unlike the other systems, BABY must contend with data that arrives at different time intervals and with differing levels of detail and reliability.

BABY uses input data that changes in time as normal (or abnormal) bodily functions proceed in the infant. In addition, the situations (the patterns of normal and abnormal physiology) change with the age of the infant. Because BABY is intended to operate over long periods of time with little operator intervention, it handles data and situations that change in time.

The control scheme for BABY is a hybrid of a forward chaining and a network-based partial sequential Bayesian updating of certainty factors. Rules are used to recognize patterns in the data and a causal network is used to identify physiologic states that should cause medical intervention or further investigation. Changes in certainty that result from evaluation of rules are propagated through the network using a modified sequential Bayesian updating of the odds for each node in the network.

As part of the management of time, each certainty factor associated with a variable is *aged* if the value is not supplied again. The aging factor used is tailored to the normal rate of change of each variable. The changing situations are managed using context arcs in the Bayesian network. The context arcs control the portions of the graph that are instantiated at any given time. The instantiation (or deinstantiation) of portions of the network also control the rules that are evaluated. Thus, it becomes possible for the system to properly recognize that patterns of findings that would signal alarm for a full term infant are *normal* for premature infants. Similarly, in the presence of known problems, expected patterns of abnormality are not pointed out, but those that should not be expected under the circumstances are.

8.4. ESSAI

The ESSAI system, incorporating many of the ADVISE ideas, was developed by the Tools and Technology Group of the telecommunications company, Alcatel, at their base in Paris, France. It was applied to the customer application engineering (CAE) process whereby the exchange requirements of a telephone administration are translated into detailed design and layout specifications needed for the exchange installation. The CAE process is naturally subdivided into 20 major functional units — each of which can be associated with a knowledge base segment. The equipment placement portion of the CAE process is concerned with determining a valid placement within a building for the various items of equipment that constitute a System 12 telephone exchange. This equipment includes the telephonic equipment such as the suites into which are placed the racks, power supplies, and the main distribution frame containing peripheral equipment. The placement process involves knowledge of physical constraints affecting equipment position and access as well as knowledge of exchange design rules affecting power supplies and cabling requirements.

The object oriented nature of the system has proven particularly effective in supporting end user interaction. Here, the major challenge was to enable the end user to prepare a representation of the building plan using a high resolution graphics workstation. The building plan supplies the physical space dimensions of each of the rooms in the building and shows the psoition of physical obstacles, such as pillars, doors, and location of windows; the plan may also indicate customer preferences for positioning equipment, access points, and a specification of more detailed aspects of the switching room. The layout produced by the expert system is stored in a tabular database to enable extensions or modifications to the plan by the customer, and the layout can be printed on a plotter to produce a scale drawing of the floor plan.

The experience with ESSAI has shown the value of an integrated paradigm which combines features of rule-based systems with those of object oriented programming. This combination enables a high degree of control and configurability without losing the advantages of declarative representations and non-deterministic problem solving.

9. Design of a Language for Integrated Knowledge-based Systems

The integrated approach described here is of little value unless all of its components can be unified in a single knowledge-based system. At first glance, it is not apparent that the diverse operations and representations described in the integrated approach can coexist within a single formalism. The Knowledge Base Variable-valued Logic system 2 ($KBVL_2$) has been developed to provide a unifying syntax and semantics for expressing operations and knowledge within the integrated approach. This section discusses a few major design constraints on $KBVL_2$ and illustrates major components of the language with simple examples.

9.1. Requirements for $KBVL_2$ Design

The $KBVL_2$ language is intended to support the three representations which are part of the integrated approach, the inductive learning operations defined over them, and the problem solving operations defined over them. This formalism is intended to integrate the various operations and knowledge base representations as much as possible rather than simply merge otherwise separate systems for rules, networks, and data. The $KBVL_2$ formalism is intended to support batch operation in which a knowledge base can be described and tested [28] as well as interactively constructed and refined [32, 24]. Finally, all knowledge structuring primitives as well as inferential operators must be available to operations coded in the knowledge base language itself.

The last constraint on the design of $KBVL_2$ is implicit within the entire integrated approach. In many knowledge-based system environments, the logic of the control of the problem solving is implicit within the design of the system itself and is not subject to change by the knowledge base builder. Because of limitations in the knowledge representation and because of this implicit definition of the control information, considerable portions of sophisticated exert systems for real world problems must be encoded directly in a procedural language of some form rather than be captured in the knowledge base itself. This problem arises when the implicit procedural control of problem solving does not match the explicitly required problem solving behavior of a portion of the solution to a complex problem.

The theme of explicitly encoding problem solving control information within the knowledge base runs through the integrated approach (see reasoning from networks to networks, networks to rules, and rules to rules above). The $KBVL_2$ formalism is intended to directly address the need to provide for explicit declaration of the sort of problem solving control information which is traditionally implicit within a particular expert system shell. Thus, the knowledge base builder can construct a knowledge-based system where strategic knowledge about how to go about the problem solving process can be integrated within the knowledge base formalism along with the traditional tactical information about the problem at hand.

9.2. Knowledge Base Objects

In $KBVL_2$, each atomic entity and all of the composite enties are viewed as objects. That is, they can have instance variables and methods attached. Instance variables themselves may be objects and methods are rules which are executed when the method is invoked. A few objects such as integer, real, string, and a generic object with no properties are predefined in the system. Additional objects for manging networks, rules, and a relational database are described below.

In $KBVL_2$ there is a distinction between class definitions and instances as found in most object oriented systems. A class definition and instance definition are represented as:

```
< new_class_name >=< existing_class_name > with < object options >
< new_class_name > : < existing_class_name > with <object options>
```

respectively where the *with <object options>* is optional and defines additional instance variables and methods if required for the new object. Thus, to define a class of integer valued laboratory tests which can be performed on one of two different instruments with a cost per test which is dependent on the instrument used, one would write:

```
two_instruments=integer with
    {instrument:nominal(instrument1,instrument2);
     cost_instrument1 : real; cost_instrument2 : real;
     price : method {[instrument = instrument1] ⇒ [price = cost_instrument1];
                     [instrument = instrument2] ⇒ [price = cost_instrument2]; } }
```

where the method called *price* calculates the price of the test by sequentially scheduling the rules in the body of the method for evaluation and potential execution. The class called *two_instrument_test* is formed from the builtin class *integer* by adding the indicated instance variables and methods. In order to define two instances of the new class and initialize their costs, one would write:

```
test1:two_instrument_test with cost_instrument1 = 2.34, cost_instrument2 = 3.45;
test2:two_instrument_test with cost_instrument1 = 3.31, cost_instrument2 = 2.05;
```

where the two instances differ from each other only in their initial values for their costs based on each instrument. Using these structuring primitives, a knowledge engineer can construct concept descriptions which are natural to the domain and tailor them to individual problems by changing initial values, adding instance variables, or adding methods.

9.3. Structuring the Knowledge Base – Segments

A large knowledge base can quickly become unmanageable if representations for knowledge base structuring are not used. In $KBVL_2$, knowledge base segments are defined and used to delineate portions of the knowledge base which behave as structural units. (The method definition in the previous subsection is an example of a knowledge base segment nested within the definition of a knowledge base *object*. In keeping with the three forms of knowledge representation, there are three predefined knowledge base segments which can be instantiated by a knowledge base builder.

The class and instance definitions shown above cannot stand alone; they must be included in a knowledge base segment. The class definition would most naturally be included in a network segment of class definitions linked by the *subclass* and *superclass* relations. The individual laboratory tests would be included in a table of other attributes and both might be included together in a knowledge base *segment* called *test_kb*:

```
test_kb : {class : network (subclass, superclass) {
          two_instrument_test = integer with
          {instrument : nominal (instrument1, instrument2);
          cost_instrument1 : real; cost_instrument2 : real;
          price:method {[instrument = instrument1] => [price = cost_instrument1];
          [instrument = instrument2] => [price = cost_instrument2]; }
          } }
          attribute : table(name : string, value : object) {
          test1:two_instrument_test with cost_instrument1 = 2.34,
                                         cost_instrument2 = 3.45;
          test2:two_instrument_test with cost_instrument1 = 3.31,
                                         cost_instrument2 = 2.05;
          } }
```

Multiple knowledge base segments can be constructed for subproblems and each can use concept descriptions from global segments as well as individualized knowledge representations and problem solving techniques.

The example above shows a knowledge base segment containing different kinds of knowledge. A rule segment could be added which produces interpretive reports for the two laboratory tests and the segment would form a solution to the subproblem of generating reports. Segments can be used to define structure within the knowledge base or to portray its organization. For instance, given a class definition for a blackboard as a table with *post*, *remove*, and *search* methods operating on blackboard entries, a structured blackboard can be constructed by creating a network of sub-blackboards. The decomposition of the blackboard can parallel decompositions within the rulebase or can simply reflect structure within the solution to the problem and not the manner of problem solving.

9.4. A More Complete Example – The Monkey and the Banana

The $KBVL_2$ language can be illustrated with a simple example of a knowledge base for solving a problem. The problem and solution shown in this section have been constructed to present a range of features of $KBVL_2$ and not with the intent of presenting an *optimal* solution.

The monkey and banana problem has been used frequently to illustrate machine-based problem solving systems. The $KBVL_2$ language is well suited to describing both the problem and its solution. The problem to be solved is that of instructing a monkey alone in a room with several objects of furniture and a bunch of bananas suspended from the ceiling. The monkey's goal is to assemble the furniture into a pile so that the bananas can be reached by climbing on the furniture. The instructions can be presented as a sequence of commands to move about within the room, carry objects, pile objects, climb on, and grasp objects. The $KBVL_2$ object classes and objects which can be used to define the monkey and banana problem are (omitting the optional *object with* from the declarations):

```
location = {zpos : integer; xypos: {xpos,ypos:integer}}
movable_object = {height:integer;
                  identity : nominal(table, stool, box, banana);
                  xyzpos:location;}
room : {objects : set of movable_object;
        banana : location with identity : nominal(banana); }
monkey : movable_object with {grip : movable_object;}
```

where the operations grasp, climb_on, moveto, and release are also defined as truth-valued functions that return true if their indicated function was accomplished by the monkey. The situation can be defined by initializing the instance variables for each movable_object in the room and placing them in the set of objects associated with the room. The goal of the monkey can be stated as an attempt to make the logical condition [*identity of grip of monkey = banana*] be true. The named rules which define the possible actions of the monkey are:

```
GRASP :For_all obj in objects of room [grip of monkey = empty]
       and [location of monkey = location of obj]
       ⇒ [grip(obj)] ⟶ [grip of monkey = obj]

MOVE :For_all obj in objects of room [xypos of obj ≠ xypos of monkey]
       ⇒ [moveto(xypos of obj)]⟶ [xypos of monkey = xypos of obj]
```

CLIMB :For_all obj in objects of room [location of obj = location of monkey]
$\Rightarrow$ [climb_on(obj)]$\longrightarrow$ [zpos of monkey = zpos of monkey + height of obj]

RELEASE : [grip of monkey $\neq$ empty] $\Rightarrow$ [release] $\longrightarrow$ [grip of monkey = empty]

where the implication arrows on the right hand side of the rules describe the side effects of the instructions to the monkey. (The syntax of the quantification has been altered slightly for the purposes of readability in this example.) The rules above constitute both a statement of the degrees of freedom of the monkey and the solution to the problem. Taken together with the logical statement of the goal, the rules non-deterministically define a solution to the problem at hand. Unfortunately, neither a forward nor a backward chaining search will produce a solution to this problem with a short search.

While the rules above accurately portray the solution to the problem, they do not lend themselves to its rapid solution. The performance of the rules can be substantially improved by incorporating information about the goal directly into them. In the event that the rules of behavior are modified to include information pertinent to the monkey's search for the banana, the rules become less useful for other purposes. Indeed, it is just such a mixing of tactical and strategic information that the integrated approach is intended to avoid.

Information about good strategies to employ can be readily incorporated into a network structure which can direct the search. For instance, the concept of *carry* can be represented as *release* $\rightarrow$ *move_to* $\rightarrow$ *grasp* $\rightarrow$ *move_to*. Similarly, the concept of *pile* can be constructed as a loop involving *carry* $\rightarrow$ *climb* $\rightarrow$ *release*. Finally, the overall strategy can be specified as *pile* followed by grasping the banana.

Separating the strategic information from the rules about the world of monkies and bananas allows the strategic information to be treated as only one of many possible annotations on the group of rules. Different annotations can be added and kept distinct from each other as well as distinct from the rules themselves.

A data table (not shown in the example above) can be used to accumulate a trace of the actual commands issued to the moneky. In the event that a correct solution which can be correctly executed is derived, then the table will contain a trace of the monkey's steps in obtaining the banana. In any case, the tabular trace, or tables of such tables, can be used to inductively derive the sort of strategic information discussed above.

10. Bringing It All Together – QUIN

The original research on the ADVISE system which emphasized the use of multiple knowledge representations, multiple control and uncertainty propagation schemes, together with a major learning component has been generalized to form the integrated approach to knowledge-based systems presented here. The integrated approach emphasizes the construction of knowledge bases using the best problem solving and representational tools at hand rather than using a single prescribed representation and problem solving scheme. The approach emphasizes the integration of machine learning tools directly within the knowledge base language and within an interactive user interface for knowledge acquisition and refinement.

The $KBVL_2$ language was developed as part of this integrated approach and provides for description of libraries of knowledge base segments which can be integrated into a single knowledge-based system or used as building blocks for defining a family of knowledge-based systems. The $KBVL_2$ language directly supports the definition of problem solving control information separated from tactical decision rules. This separation together with the rich combination of network, rule and database structuring primitives for knowledge representation provides a more widely useful knowledge-based system tool than would otherwise be possible.

The QUery and INference system, QUIN, provides interactive access to the knowledge represented in a $KBVL_2$ knowledge base. QUIN is primarily a marriage between relational database and inductive learning technologies [33]. Its purpose is the management of large amounts of data for input to and output from the inductive transformation operators shown in Figure 1. The system provides rudimentary access to problem solving using the knowledge base for deductive and inductive retrieval as well as learning.

QUIN may be used for the management and analysis of data as well as to browse through a knowledge base. Management here refers to the creation, retrieval, and modification of the data, while analysis refers to activities that attempt to discover more about 1) interrelationships within the data and 2) phenomena that produce those interrelationships. Although the QUIN interface was originally developed to operate only on tables of relational data, some editing and display operations on networks and rules have been added in this integrated approach.

The operations in QUIN provide access to the inductive inference, analysis, and problem solving transformations shown in Figure 1. In general, each operation requires an options object in addition to the table(s) of input data on which to operate. Output from the inferential process in the form of tables, networks or rules can be inspected by the user and introduced into a new segment or stored over the information in the segment from which it came. (Details about each operation can be found in sections 6 and 7 above.)

The induction operations within QUIN interact to form a set of utilities that can be used in sequence or in cycles. The databases used to test and experiment with these learning algorithms are more easily handled with database management techniques that store, modify, and restructure data for eventual input to the inference programs. The cycle of knowledge refinement by iteration of the mechanized inference with a human critic produces better results than either alone.

The integrated approach to the construction of knowledge-based systems discussed in this paper brings together interactive application of machine learning techniques, variations in knowledge representation, problem solving strategy, and uncertainty management in a single paradigm. The integrated approach, while not fully realized in any existing system, has been substantially incorporated in experimental meta-expert system tools and has demonstrated its potential for qualitative improvement in knowledge-based system technology.

REFERENCES

[1] Klein, P.J. and Dolins, S. B., ***Choosing Architectures For Expert Systems***, CCSC Technical Report # 85-01-001, Texas Instruments Incorporated, Dallas, TX, 1985.

[2] Michalski, R.S., Baskin, A.B., Uhrik, C. and Channic, T., ***The ADVISE.1 Meta-Expert System: The General Design and a Technical Description***, UIUCDCS-F-87-962, Department of Computer Science, University of Illinois, Urbana Illinois (DCSUI), 1987.

[3] Davis, R. and King, J., ***An Overview of Production Systems***, in: Elcock and Michie (eds.), Machine Intelligence, 1976.

[4] Buchanan, B.G. and Duda, R.O., ***Principles of Rule-base Expert Systems***, Advances in Computers, **22**, Academic Press, 1983.

[5] Drazovich, R.J., McCune, B.P., and Buchanan, B.G. ***Characteristics of hypothesis formation systems***, Technical Report, Advanced Information and Decision Systems, Mountain View, Ca, Project Number 3003, May 27, 1981.

[6] Baskin, A.B., ***LOGIC NETS: Variable-valued Logic plus Semantic Networks***, Policy Analysis and Information Systems, No. **3**, 1980.

[7] Michalski, R.S., ***Pattern Recognition as Rule-Guided Inductive Inference***, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. **4**, 1980.

[8] Michalski, R.S., ***Pattern Recognition as Knowledge-Guided Computer Induction***, Report No. 927, DCSUI, 1978.

[9] Michalski, R.S., Carbonell, J.G., and Mitchell, T., (eds.), ***Machine Learning, An Artificial Intelligence Approach, Vol II.***, Morgan Kaufman Publishers, Inc., Los Altos, CA, 1986.

[10] Michalski, R.S., Mozetic, I., Hong, J., and Lavrac, N., ***The AQ15 Inductive Learning System: An Overview and Experiments***, UIUCDCS-R-86-1260, ISG 86-20, DCSUI, 1986.

[11] Winston, P.H., Binford, T.O., Katz, B. and Lowry, M., *Learning Physical Descriptions from Functional Definitions, Examples and Precedents*, in: Proceedings of AAAI-83, Washington, D.C., 1983.

[12] Carbonell, J.G., *Derivational Analogy in Problem Solving and Knowledge Acquisition*, in: Proceedings of the International Machine Learning Workshop, University of Illinois Allerton House, Urbana, Illinois, 1983.

[13] Burstein, M.H., *Concept Formation by Incremental Analogical Reasoning and Debugging*, in: Proceedings of the International Machine Learning Workshop, University of Illinois Allerton House, Urbana, Illinois, 1983.

[14] Lenat, D.B., *Case Study 2: The Eurisko Program; Heuristics Used to Develop New Heuristics*, in: MACHINE LEARNING: An Artificial Intelligence Approach, R.S. Michalski, J. Carbonell, and T. Mitchell (eds.) TIOGA Publishing Co., Palo Alto, CA, 1983.

[15] Stepp, R.E. and Michalski, R.S., *Conceptual Clustering: Inventing Goal-Oriented Classifications of Structured Objects*, in: Machine Learning: An Artificial Intelligence Approach, Volume II, Morgan Kaufman Publishers, Inc., Los Altos, CA, 1986.

[16] Michalski, R.S., *AQVAL/1 - Computer Implementation of a Variable-valued Logic System VL_1*, in: Proceedings of the First International Joint Conference on Pattern Recognition, Washington, DC, 1973.

[17] Larson, J., *INDUCE-1: An Interactive Inductive Inference Program in VL_{21} Logic System*, Report No. UIUCDCS-R-77-876, DCSUI, 1977.

[18] Quinlan, J.R. *Discovering Rules from Large Collections of Examples: A Case Study*, in: Expert Systems in the Micro-Electronic Age, Michie, D. (ed.), Edinburgh University Press, Edinburgh, 1979.

[19] Michalski, R.S. and Larson, J.B., *Selection of Most Representative Training Examples and Incremental Generation of VL_1 Hypothesis: The Underlying Methodology and the Descriptions of Programs ESEL and AQ11*, Report No. 877, DCSUI, 1978.

[20] Michalski, R.S. and Chilausky, R.L., *Knowledge Acquisition by Encoding Expert Rules versus Computer Induction From Examples: A case study involving soybean pathology*, International Journal for Man-Machine Studies, No. 12, 1980.

[21] Dieterich, T., and Michalski, R.S., *Discovering Patterns in Sequences of Events*, Artificial Intelligence Journal, Vol25, No.2, 1985.

[22] Baim, P.W., *Automated Acquisition of Decision Rules: The Problem of Attribute Construction and Selection*, Masters Thesis, DCSUI, 1984.

[23] Cramm, S., *ESEL/2: A Program for Selecting the Most Relevant Training Events for Inductive Learning*, ISG 83-1, UIUCDCS-F-83-901, DCSUI 1983.

[24] Riedesel, J.D., *Consistency and Completeness: An Exercise in Knowledge Base Validation*, Masters Thesis, DCSUI, 1988.

[25] Michalski, R.S., *Knowledge Acquisition Through Conceptual Clustering: A Theoretical Framework and an Algorithm for Partitioning Data into Conjunctive Concepts*, International Journal of Policy Analysis and Information Systems, Vol. 4, No. 3, 1980.

[26] Jacob, R.J.K., and Froscher, J.N., *Developing a Software Engineering Methodology for Knowledge-Based Systems*, Technical Report NRL REport 90919, Naval Research Laboratory, 1986.

[27] Collins, A. and Michalski, R.S., *THE LOGIC OF PLAUSIBLE REASONING: A Preliminary Report*, Report No. UIUCDCS-F-85-951, DCSUI, 1986.

[28] Reinke, R.E., *Knowledge Acquisition and Refinement Tools for the ADVISE Meta-Expert System*, Masters Thesis, DCSUI, 1984.

[29] Michalski, R.S., et. al., *PLANT/ds: An Expert System for the Diagnosis of Soybean Diseases*, European Conference on Artificial Intelligence, 1982.

[30] Boulanger, A.B., *The Expert System PLANT/cd: A Case Study in Applying the General Purpose Inference System ADVISE to Predicting Black Cutworm Damage in Corn*, Masters Thesis, DCSUI, 1983.

[31] Rodewald, L.E., *BABY: An Expert System for Patient Monitoring in a Newborn Intensive Care Unit*, Masters Thesis, DCSUI, 1984.

[32] Spackman, K.A., *QUIN: Integration of Inferential Operators within a Relational Database*, ISG 83-13, UIUCDCS-R-83-917, DCSUI, 1983.

[33] Michalski, R.S., Baskin, A.B., and Spackman, K.A., *A Logic-based Approach to Conceptual Database Analysis*, in: Proceedings of Sixth Annual Symposium on Computer Applications in Medical Care, Washington, D.C., 1982.