LOGIC-BASED FORMULA MANAGEMENT STRATEGIES IN AN ACTUARIAL CONSULTING SYSTEM

Taracad Sivasankaran
Matthias Jarke

March 1984
Revised December 1984

Working Paper Series
Stern #IS-84-44

LOGIC-BASED FORMULA MANAGEMENT STRATEGIES IN AN ACTUARIAL CONSULTING SYSTEM

Abstract

In many decision support systems, multiple decision methods and models must be combined for solving a complex problem. Expertise is required for selecting, adapting and coordinating appropriate models. This paper describes the design and implementation of a knowledge-based model management system called the Actuarial Consulting System (ACS). The ACS supports actuaries in making pricing decisions in the domain of life insurance. Actuarial knowledge is organized using a graph formalism called Formula Derivation Network (FDN), represented in Prolog as a hierarchy of predicates. On the user level, a Problem Analyzer converts a problem specification by the user into a search problem on the stored collection of FDNs. Using different search strategies, including human expert rules, the Surface Planner generates an efficient solution strategy (sequence of models). At the lowest level, a Plan Executor retrieves or requests model data and issues appropriate function calls to a subroutine library.

Keywords; model management, logic-based decision support systems, actuarial science, life insurance, hierarchical knowledge base management, expert systems.

1 INTRODUCTION

A research effort at New York University investigates the integration of artificial intelligence (AI) methods into existing decision support systems (DSS) [Jarke & Vassiliou 84]. One part of the project studies the interaction between expert systems and large existing databases [Vassiliou et al. 84; Jarke et al. 84], another one specific aspects of business expert systems, focusing on expert systems for insurance underwriting [Clifford et al. 85]. This paper describes work on a third subproject that investigates the combination of AI methods with quantitative models, in particular, intelligent model management.

In complex decision situations, it will often be necessary to coordinate the application of multiple decision models for solving a problem. Decision Support Systems need a model management component [Elam et al. 80; Sprague & Carlson 82] that handles the tasks of identifying appropriate models from a problem description, sequencing their application, and instantiating them with the necessary data.

The paper describes the design and implementation of a prototype model management system that supports actuaries in their work. In an insurance company, actuaries are responsible for evaluating the risks of providing insurance for life contingencies, such as death, disability, or retirement. The system -- called the Actuarial Consulting System (ACS) -- structures life insurance problems by organizing appropriate formulas and models, evaluating premiums, and explaining possible solution methods. The focus of the present

paper are the model selection and combination capabilities of the system; details of other features are provided in [Sivasankaran 84].

Several other authors have addressed model management issues. Following [Bonczek et al. 82], three stages of model management systems can be distinguished. In the simplest case, the user must procedurally state an algorithm to solve the problem at hand. In the second stage, the user may select among a set of pre-specified models provided by the system. Several high-level languages have been proposed for this purpose. A good example is Blanning's [1982] relational model management which views a model as a relation between input and output data and implements model sequences by joins between these relations.

The ACS uses a similar, although graph-based high-level model description but actually falls into the third category of [Bonczek et al. 82]: it automatically selects a combination of models, guided by its knowledge base and by a high-level problem specification provided by the user. This approach requires the use of AI techniques. In principle, a pure resolution-based system such as Prolog [Clocksin & Mellish 82] could be used for this task; see also [Bonczek et al. 81] for an exploration of this option.

However, it has been observed that pure resolution systems do not provide sufficient control of the solution planning process [Dolk & Konsynski 83]. For example, it may turn out to be very costly to execute models concurrently with the reasoning process if the results may subsequently have to be discarded due to backtracking.

Consequently, the ACS -- although implemented in Prolog -- employs a more hierarchical control structure. On a surface level, a planner selects applicable models and manipulates them into a feasible and efficient solution plan. This level uses a graph-based knowledge representation that facilitates the search for solution strategies and the evaluation of alternatives. No number-crunching is involved at this level and Prolog's unification capabilities, augmented by cost estimates for additional search space reduction, prove very helpful.

Once a promising plan has been established, the execution level instantiates the required data values and executes the models selected by the planner. This level may need access to databases and mathematical libraries for which Prolog may not be the ideal programming tool; coupling with external systems may become necessary [Jarke & Vassiliou 84]. If not all required data are available, control is returned to the surface planner which may either try an alternative strategy, or invoke a fact acquisition subsystem designed to obtain missing information (or at least directions where to find it) from the user.

The remainder of this paper describes the ACS in more detail. Section 2 briefly reviews the range of actuarial problems to be supported by the ACS. Section 3 presents knowledge representations for actuarial concepts and problem solving strategies. In section 4, the layered architecture of the ACS is described; more details on the main model management component -- the surface planner -- are provided in section 5. Section 6 demonstrates the usage

of the system by a comprehensive example and section 7 reviews the status of the system and outlines extensions currently under design.

2 MODEL MANAGEMENT IN ACTUARIAL SCIENCE

The statistical study of the contingencies of human life, such as death, disability, or retirement forms the foundation of actuarial science. Experts in this field are called actuaries. The actuary must estimate the probabilities of occurrence of contingent events as a basis for calculating premiums, reserves, annuities, etc., for insurance and other financial operations. For the solution of problems involving these contingencies, an actuary requires some quantitative measure of their effects. In problems involving financial calculations, the actuary also requires a set of principles by which probabilistic measurements may be combined with interest functions to produce monetary values [Jordan 75].

The actuarial domain deals with a large number of formulas, equations, and models, many of which are intertwined with one another. At several points of the actuarial problem-solving process, expertise is needed. First, the actuary must comprehend and formulate the problem in terms of insurance concepts, and of the available data like mortality rates, interest rates, commutation function values and health related risk scores. Expertise is also required for choosing, transforming and sequencing an efficient set of applicable formulas. Generally, the actuary must develop an overall solution plan before any actual computation, because many actuarial problems require several formulas to be transformed and combined in a particular sequence.

Knowing the solution strategy beforehand helps avoid cycling and redundant computation. Knowledge is finally needed for deciding whether to access tables of pre-stored data, to compute these values because direct computation is cheaper than accessing the tables or even the only method, and when to override default value table access by user-specified data.

The ACS represents the different actuarial concepts, formulas, and heuristics of problem-solving in a carefully organized knowledge base. The knowledge base must support at least the following functions:

1. Compute the premium for an insurance benefit or mix of benefits
2. Assess feasibility of the benefits
3. Explain the result by showing which models were applied in which sequence
4. Allow the user to modify the reasoning process
5. If a certain problem cannot be solved, point out why. Also ask for values which, if supplied, can solve the problem.

The ACS is not intended to replace an actuary but to assist in life insurance problems by serving as a 'intelligent calculator', i.e., a decision support tool. It was built for expert users and may not be suitable for a user unfamiliar with the basic concepts of life contingencies theory.

3 STRUCTURAL MODEL OF ACTUARIAL DOMAIN KNOWLEDGE

Actuarial theory is usually represented as a conglomeration of interrelated actuarial concepts. Each actuarial concept has a unique notation representing an individual insurance-related idea which can take a numerical value. Two examples are provided, below. This section will discuss the representation of actuarial concepts and their relationships in the ACS.

Concept	Notation	Description	Sample Value
Mortality Rate	q_x	Probability a life aged x will die in one year	.05
Reserve	${}_tV_x$	Net worth of a policy for a life aged x in a pool of premium receipts as at the end of t'th year	$5000

A hierarchy of formalisms called Formula Derivation Net (FDN), Individual Concept Structure (ICS), and Derivation Structure (DS) capture interrelationships among the actuarial concepts. FDN's, ICS's and DS are defined as directed labelled graphs. A FDN consists of a set of interconnected nodes with each node representing an actuarial concept. FDNs are of three types: One-Sided, Mutual, and Collective (Fig. 1).

One-sided link. If a concept is derivable from another one but not vice versa, such a relation is called a one-sided link. For example, in Fig. 1(a) the actuarial concept l_x represents the number of people alive at age x out of a group which started off with l_0 at age zero. ${}_tp_x$ is the probability that a

life aged x will survive t years. l_x cannot be computed from ${}_tp_x$ alone because two values (l_x and l_{x+t}) of the former concept are needed to compute the latter.

Mutual link. If two concepts are derivable from each other, we have a mutual link. For example, in Fig. 1(b), ir and dr are the interest and discount rates respectively. As shown in the figure, if either ir or dr is known, one can find the other using the formula indicated above the arrow. Not all mutual links have to be stored explicitly since the system supports certain simple algebraic formula transformations similar to those in MACSYMA.

Collective link. Here we have a situation where a concept can be expressed in terms of more than one other concept. This is represented by an AND graph [Nilsson 82]. Consider the formula in Fig. 1(c) where A_x represents the present value of \$1 insurance on a life aged x and $a_x^{\cdot\cdot}$ represents the present value of a life annuity payable at the beginning of each year.

---

insert Fig. 1 about here

The different FDNs that compute the same goal concept are combined into an Individual Concept Structure (ICS). The ICS defines the different paths through which a goal concept can be derived. The overlay of all ICSs is a

state-space representation of the stored actuarial knowledge that will be referred to as the Derivation Structure (DS). The DS represents the total static knowledge of a particular implementation of the expert system. A portion of a DS is shown in Fig. 2.

---

insert Fig. 2 here

Each FDN is represented in Prolog using a "can_find" predicate at the surface level and an "evaluate" predicate at the execution level. The general structure of these predicates is as follows:

```
can_find(Goal_category_concept, Computation_procedure_identifier,
        if_known(Required_concepts)).
evaluate(Computation_procedure_identifier, Required_concepts, Result).
```

For example, consider the above FDN for A_x (denoted [cap,a,x] in Prolog). The "can_find" predicate, shown below, stores the knowledge that the concepts A_x, $a_x^{\cdot}$; and dr are interrelated, in particular, that A_x can be solved for if the other two are known. The predicate also keeps an identifier denoting the formula connecting these concepts (0311 in the example below). The "evaluate" predicate represents a procedure which instantiates the concepts numerically and invokes the formula execution. In this simple example, the computation can be easily expressed in Prolog itself and no external function call is necessary.

```
can_find([001,cap,a,X], 0311, if_known ([[012,a,tremma,X], [503,dr]])).

evaluate(0311,Dr,A_tremma_x,Cap_a_x) :-
            member([012,a,tremma,X,is,Val1],Concepts_with_value,Rest1),
            member([503,dr,is,Val2],Concepts_with_value,Rest2),
            Cap_a_x is ( 1 - Dr*A_tremma_x ).
```

Knowledge representations similar to the ones proposed here have been used in expert systems for organic synthesis and geology. Expert systems in the area of synthesis of organic compounds, such as LHASA [Corey & Wipke 69], SECS [Wipke et al. 77] and SYNCHEM [Gelernter et al. 77] use synthesis trees to organize the body of knowledge about chemical reactions. Synthesis routes that create the desired target molecule are viewed as AND/OR branches of the synthesis tree. The tree descends from the goal node representing the compound to be synthesized to the terminal nodes representing the starting chemical compounds. The branches connecting the nodes represent possible chemical reactions.

In PROSPECTOR [Duda et al. 78], an expert system in the field of geology, domain knowledge is represented in a so-called inference network. The nodes represent assertions about entities in the domain. The arcs between the nodes represent either inference rules or provide a context for testing another assertion. The system propagates the user's initial assertions through the inference network and on that basis selects one of its pre-stored geological models to guide its search for discovering what minerals can possibly be identified.

Domain	Name of Data Structure	Nodes	Connecting Links
Organic Synthesis	Synthesis Tree	Compounds	Chemical Reactions
Geology	Inference Network	Assertions	Inference Rules
Insurance	Derivation Structure	Actuarial concepts	Actuarial Formulas

Table 1. Comparison of Knowledge Representations

An interesting distinction between the data structures used in organic synthesis systems and the ACS is that the insurance data structure can automatically insert derived links (see Section 5) between nodes, without requiring the explicit representation of each possible type of link between the different nodes representing the actuarial concepts. A major difference between the data structures in geology and insurance is that in the insurance domain the relationships among the nodes are exact formulas whereas in geology the inference rules have uncertainty factors measures associated with them. On the other hand, the number of possibly interacting actuarial functions seems to be larger than in the very modular PROSPECTOR system.

4 IMPLEMENTATION MODEL OF ACTUARIAL DOMAIN KNOWLEDGE

An actuary when faced with an insurance problem often goes about structuring a solution strategy intuitively. Computerizing this task requires an understanding of the actuarial problem solving process. Rather than relying on a collection of individual expert rules, a general implementation model of this process was developed. This 'model of model management' -- shown in Fig.

3 -- also serves as the control structure for the ACS. It has the following components.

1. Problem Analyzer
2. Surface Planner
3. Plan Executor
4. Database
5. Knowledge Base
6. Blackboard

---

insert Fig. 3 about here

We shall briefly discuss each of these components.

Problem Analyzer. This component accepts a problem statement from the user and attempts to determine what the user is trying to solve for, and what constraints have to be kept in mind while developing a solution. In interpreting a problem statement, the Problem Analyzer searches for a set of key words. The problem statement is broken into three parts: the goal category insurance concept, the type of insurance benefit, and the constraints set by the user. Such interpretation of the problem will be referred to as generating a problem context. Later, it will be seen that the user has the

opportunity to impose further constraints or to restate existing ones during the problem-solving process.

Surface Planner. The task of the Surface Planner is to develop a workable solution strategy for the problem context generated by the Analyzer. Using the set of "can_find" predicates in the knowledge base together with cost estimates for formula execution, it develops an optimal network of derived links from the Derivation Structure that will symbolically solve the problem (Fig. 4). A derived link associates different actuarial concepts transitively through one or more mediating concepts. For example, in Fig. 5, the dotted lines indicate the derived link. Although originally A_x is represented in terms of dr and a_x and each in turn is represented in terms of ir and a_x, it is possible to use these sequential dependencies to derive a new link that directly connects A_x to ir and a_x.

---

insert Fig. 4 and Fig. 5 about here

Plan Executor. This component inherits the solution method developed in the previous step. It then accesses the knowledge base and selects the 'evaluate' predicates corresponding to the formulas to be used. It instantiates the parameters with numeric values and carries out the computations in order to get the result. It can also access data base values if necessary, or request missing data from the user (see section 6.4).

Database. The database consists of numerical values for actuarial concepts and other important factors, such as interest rates. The ACS contains a specialized data dictionary facility to manage those of the values stored within the Prolog knowledge base. External storage of table values, and access to customer data will be provided through a Prolog-database connection [Vassiliou et al. 84; Jarke et al. 84].

Knowledge Base. The knowledge base component (Fig. 6) contains static and dynamic rules used by the previously described subsystems. Static rules identify the types of insurance benefits, actuarial notations and table values of interest, as well as textbook formulas. Dynamic rules deal with the knowledge about developing efficient problem-solving strategies by selecting and manipulating formulas, evaluating alternative solution methods and computing them.

Blackboard. This is a working space which serves as a scratchpad for the Problem Analyzer, the Surface Planner and the Plan Executor. It helps to create data structures, erase them and modify them dynamically during the process of problem solving. The original problem statement, its analyzed version, intermediate steps during a long search and intermediate results can all be stored for future references.

---

insert Fig. 6 about here

---

5 Surface Planning Strategies

Three types of strategies have been incorporated in the Surface Planner: basic breadth-first search, cost-based search, and human expert rules. Prolog's standard depth-first search appears less suitable for most actuarial problems since many problems will have solutions which are only a few steps deep but not immediately obvious.

The basic breadth-first search contains a simple heuristic that attempts first to use formulas in which a partial match between given data and required values exists. The objective of using breadth-first is to limit the total number of formulas to be employed by trying directly applicable formulas upfront. Only when it is realized that no direct formulas exist, the problem is decomposed into layers of subgoals. In other words, different lines of reasoning are examined in parallel at each step in the decomposition of the problem and no commitment is made to any specific strategy right from the beginning.

If the user is unhappy with the proposed solution strategy displayed after the basic breadth-first search, the Planner employs a cost-based search for alternative solution plans. The idea is to find a solution method which would be the cheapest for the Plan Executor to work with. Cost estimates are based on the number of formulas needed in each solution method, the number of input concepts and the amount of computation involved in using each. Thus, a

solution sequence which involves more formulas might be preferred to a shorter sequence with costly computations.

While these two basic methods employ backward chaining, the Surface Planner can also make leaping conclusions like a typical human expert based on certain rules of thumb or experience used by actuaries. This may be called shortcutting the plan development and is simply implemented by adding new FDNs for the expert rules. While the scheme chosen for knowledge representation makes the implementation of these rules easy, the more difficult part is the precise statement of the circumstances under which these shortcuts (often approximations) are applicable. For example, consider the computation of a premium for a pension plan subject to the condition that in the event of death the premiums be returned with interest. An expert actuary can use a 'tricky' factor $a\ddot{}_{x+n} \; / \; s_{n}\dot{}$, obviating a long sequence of computations (see the Appendix for an explanation). Such heuristics have to be used with care only if the problem context warrants them.

6 AN EXAMPLE

Since the user interface has not been the primary concern of this research to date, input is provided to the system following a relatively simple structured English format:

FIND <goal category concept> FOR <type of insurance benefit>

[GIVEN <constrained concept values>].

The key words FIND and FOR are essential while GIVEN is optional

(indicated by the square bracket), depending on whether the user wishes to specify some concepts or constrained values to be used during the problem solving stage. All the possible goal category concepts, types of insurance benefits and constrained concepts are stored with their input patterns in the knowledge base. Slots are provided to store numeric parameter values supplied in the problem specification.

In Prolog, these concepts take the form of predicates. A few sample predicates are shown below. The capital letters indicate instantiable variables. The numbers are concept identifiers provided for control purposes and may be ignored for now.

Goal category concepts :

```
possible_goal([net,single,premium]).
possible_goal([reserve,at,the,end,of,T,years]).
possible_goal([amount,of,paid,up,insurance,at,duration,T]).
```

Types of benefit :

```
possible_benefit([613],[F,dollar,N,year,endowment,payable,
                                at,the,end,of,year,of,death]).
possible_benefit([615],[F,dollar,whole,life,annuity,payable,
    at,the,end,of,year,of,death,with,payments,guaranteed,for,N,years]).
```

Constrained concept values :

```
possible_value([012],[a,tremma,X,is,Val]).
possible_value([501],[interest,rate,is,Val]).
possible_value([609],[commission,C,pc,of,gross,premium]).
```

In the sequel, the solution of a particular actuarial problem concerning a whole life insurance premium will be traced through the components of Problem Analyzer, Surface Planner, and Plan Executor.

6.1 Problem Analyzer

The Problem Analyzer breaks the problem into its three parts. Each is understood by matching the input with the pre-stored 'possible' patterns, and then converted into the appropriate actuarial notations. The Analyzer shows the generated problem context (i.e., the Goal, the type of Benefit, and the Constraint definitions) and stores them on the blackboard for future reference. Suppose, the user submits the following problem (user inputs are underlined).

```
| ?-problem.

|: find the net single premium for a 10000 dollar whole life insurance
payable at the end of year of death given age is 35,a tremma 45 is
8,10 v 35 is 1 and ir is 10 pc.

  Goal = net single premium

  Benefit = 1 10000 dollar whole life insurance payable at the end of
year of death

  Given concepts are
[[601,age,35],[12,a,tremma,45,is,8],[124,10,v,35,is,1],[501,ir,is,10]]
yes
```

6.2 Surface Planner

The Surface Planner first retrieves the problem context from the blackboard and removes numeric values for specific insurance concepts temporarily, in order to conduct a purely symbolic planning process. Then, it identifies the actuarial problem to be solved by combining the goal category concept and the type of insurance benefit. In our example, the Planner combines the goal category concept 'net single premium' with the type of

insurance benefit 'whole life insurance payable at the end of year of death' to form the corresponding uniquely identifiable actuarial goal 'cap_a_x'. For this purpose, the knowledge base contains a class of predicates called 'notation_equivalents' which determine what type of goal categories can be combined with which type of insurance benefits to yield feasible actuarial concepts. The predicate selected for our example is shown below.

```
notation_eqvt([net,single,premium],[F,dollar,whole,life,insurance,payable,
                        at,the,end,of,year,of,death],[001,cap,a,x]).
```

Once the actuarial goal has been precisely identified, the Planner tries to find ways of solving for it using the given concept constraints (without values). It searches through the can_find([001,cap,a,x], ..., if_known(...)) predicates. Each predicate either represents a directly applicable formula for computing cap_a_x or a manipulated form which can be used to compute cap_a_x if the proper algebraic transformations are applied. In our example, the Planner has three choices available in the knowledge base (first three lines below).

```
can_find([001,cap,a,X],0311,if_known([[012,a,tremma,X],[503,dr]])).
can_find([001,cap,a,X],0305,if_known([[075,cap,m,X],[036,cap,d,X]])).
can_find([001,cap,a,X],0312,if_known([[011,a,X],[501,ir]])).
            ...          ...          ...
can_find([011,a,X],0210B,if_known([012,a,tremma,X])).
can_find([012,a,tremma,X],0506B, if_known([[012,a,tremma,Y],[124,T,v,X]])) :-
                              save_problem_context(Z),
                              member([012,a,tremma,Y],Z),
                              member([124,T,v,X],Z), Y is X+T.
```

In the absence of particular "expertise" providing immediate shortcuts,

the Planner tests the applicable rules with the heuristic of focusing the search on predicates where parts of the necessary values are known from the input data. Thus, since ir (interest rate) is known from the input data, the third rule is chosen and a_x is set as a subgoal. The fourth rule above is applied in turn and a new subgoal a_tremma_x is created. After examining several alternatives of finding a_tremma_x (not shown above), the fifth rule is applied successfully. Note that this rule refers to a manipulated form of the textbook formula t_V_x = 1 - a_tremma_x+t / a_tremma_x. The successful paths finally form a complete solution strategy.

```
| ?- soln_plan.

We have to find [1,cap,a,35]
We know values of [501,ir]
We need values of [11,a,35]
 - then we can use formula/s :
cap_a_x = [1-ir.a_x] / 1+ir

 Note that  We can_find [12,a,tremma,35]
              if_known([[12,a,tremma,45],[124,10,v,35]])
 using t_v_x = 1 - a_tremma_x+t / a_tremma_x
 Note that  We can_find [11,a,35] if_known([12,a,tremma,35])
 using  a_tremma_x = 1 + a_x
yes
```

6.3 Plan Executor

The Plan Executor first retrieves the solution strategy developed by the Planner, represented on the blackboard as a list of formula identifiers. Letters are attached to the formula numbers to identify transformation to be applied to the textbook formulas. In our problem, the solution strategy is represented by the predicate, strategy([0506B,0210B,0312]). The numeric part

of the first identifier 0506B indicates to the Plan Executor that the textbook formula is t_V_x = 1 - a_tremma_x+t / a_tremma_x in a manipulated form. The Plan Executor calls the corresponding evaluation predicates one at a time. Each "evaluate" predicate contains or calls the procedure for computing the corresponding formula and can retrieve the numerical inputs either from the problem context or from the data base. Finally, the computed values are combined and the numerical solution to the problem is computed.

6.4 Fact Acquisition

If not enough information is available to solve a problem, the above procedure will notice this either at the Surface Planning or at the Execution level. In this case, the system will ask for additional information. Three cases can be distinguished (Fig. 7). In the first case, the user does not care how the problem is to be solved or where the input data come from. For example, a user may just ask for the premium for a standard policy. In this case (denoted I in Fig. 7), the system will only fail if the goal set by the user cannot be computed from any data available in the database. The fact acquisition subsystem of the ACS [Sivasankaran 84] will make an educated guess which data the user might be able to provide; if that fails again, the Surface Planner will develop an alternative plan and ask for its missing data until either a solution is found or the user decides to give up.

---

insert Fig. 7 about here

In case II, the user specifies which concepts are constrained but wishes to use default values for the constraints. The defaults should be available from the database; if not the system will ask the user for data. However, there is no need for the system to look for alternative strategies without being told so since that would be against the wishes of the user.

Finally, in case III, the user provides at least some of his own data to override default values (e.g., mortality rates) stored in the database. Two possible problems may occur in this case. One the one hand, the user may forget to specify a certain concept or to mention it at all; the above procedures can be used to add the missing information. On the other hand, the problem may be overconstrained, leading to contradictions and leaving the problem unsolvable. For example, the user may put upper limits to the premium payment capability and lower limits to the policy amount that are not compatible. The fact acquisition system will in this case try to point out where the contradiction lies so that the user can correct the input.

7 CONCLUSIONS

The ACS has demonstrated the usefulness of a layered knowledge base architecture for model management even in a logic programming environment. The performance advantages obtained by this kind of architecture increase if the models are more complex than the simple examples shown in the paper. The architecture of the system has also proven a good tool to combine exact

mathematical knowledge (as in the textbook formulas) with human expert problem-solving heuristics.

The current prototype of the ACS has a repertoire of 95 actuarial concepts and 175 formulas which covers approximately 80% of all actuarial concepts applicable to single life policies [Jordan 75]. The about 600 formulas for the multiple-life case are being added to the system. Most of the fact acquisition subsystem described in section 6.4 is also operational. Both this part and the human expert shortcut rules are being expanded, based on experience with using the system. Experiments with a number of textbook and real-world actuarial problems have demonstrated that the system is capable of finding and explaining rather 'clever' solutions to some problems, in some case solutions that the expert posing the problem had not thought of before.

One of the major next steps in this work is to improve the user interface so that it can be used with less training. In particular, we are focusing on the development of an interface for tutoring actuarial students in their preparations for the official actuarial exams. Some initial experiments with the existing prototype have already shown that the ACS can support this process effectively by permitting the student to compare multiple possible solution strategies in terms of their elegance and computational costs. However, the system will need more flexibility in its user interface to become a usable tutoring tool.

Acknowledgments

The authors are grateful to Jim Clifford for many useful suggestions in the early phases of this work. Thanks are also due to the referees whose comments greatly improved the presentation of this material.

Appendix

Explanation of the Factor $a^{\cdot\cdot}_{x+n} \,/\, s^{\cdot}_{n}$

Problem : Find the net annual premium payable for n years for a pension cover of \$1 per annum issued to a life aged x, with the first pension payment n years after date of issue and with the provision that, if the insured dies within the n year period, the net premiums paid are to be returned with compound interest to the end of the year of death.

(i) The mathematical solution is shown below :

$$
\begin{aligned}
P\,(\,N_x - N_{x+n}\,) &= N_{x+n} + P\sum_{t=1}^{n} s^{\cdot}_{t}\, C_{x+t-1} \\
&= N_{x+n} + P/d\,[\;(1+i)^t\, v^{x+t}\, d_{x+t-1} - C_{x+t-1}\;] \\
&= N_{x+n} + P/d\,[\; v^x\,(\,l_x - l_{x+n}\,) - (\,M_x - M_{x+n}\,]\;] \\
&= N_{x+n} + P/d\,[\; D_x - (1+i)^n D_{x+n} - D_x + d\,N_x \\
&\qquad\qquad + D_{x+n} - d\,N_{x+n}\;] \\
&= N_{x+n} + P\,(\,N_x - N_{x+n}\,) - P/d.D_{x+n}\,[\;(1+i)^n - 1\;]
\end{aligned}
$$

Thus, $P\, s^{\cdot}_{n}\, D_{x+n} = N_{x+n}$

$$P = \frac{a^{\cdot\cdot}_{x+n}}{s^{\cdot}_{n}}$$

(ii) Alternative heuristic reasoning :

Imagine the insured survives the n years to age x+n. The same pension benefit if issued at age x+n will cost $a_{x+n}^{\cdot\cdot}$ dollars. Suppose the prospective pensioner while at age x decides to wait till age x+n and then buy the pension coverage at this cost. However, let him create a sinking fund by depositing an amount \$P annually into a bank account, P being so chosen that over n years the annual deposits would accumulate with interest to \$ $a_{x+n}^{\cdot\cdot}$. According to the theory of compound interest, in order to accumulate \$1 over n years with interest, the annual deposit should be $1/s_{n}^{\cdot}$: Hence, to accumulate $a_{x+n}^{\cdot\cdot}$ dollars, \$P has to be $a_{x+n}^{\cdot\cdot} / s_{n}^{\cdot}$:

\$P is the solution to our problem since

1. If such an amount is deposited annually it will accumulate over n years to the price of the pension plan at age x+n which amount can be used then to buy the pension benefit

2. In case he/she dies before reaching age x+n, the annual deposits of \$P made until then can be withdrawn as if they had gone into a bank account.

REFERENCES

1. Barr A. and Feigenbaum E.A. Eds., The Handbook of Artificial Intelligence, HeurisTech Press, William Kaufmann, Stanford 1982

2. Blanning, R., "Language Design for Relational Model Management", in S.K.Chang, Ed., Management and Office Information Systems, Plenum Press 1982

3. Bonczek R., Holsapple C. and Whinston A.B., "A Generalized Decision Support System Using Predicate Calculus and Network Data Base Management", Operations Research 29, 2 (1981)

4. Bonczek R., Holsapple C. and Whinston A.B., "The Evolution from MIS to DSS: Extension of Data Management to Model Management", in M.Ginzberg, E.A.Stohr, W.Reitman, Eds., Decision Support Systems, North-Holland 1982

5. Clifford J., Jarke M., and Lucas H.C., "Designing Expert Systems in a Business Environment", submitted for publication

6. Clocksin W.F., and Mellish C.S., Programming in Prolog, Springer-Verlag 1981

7. Corey E.J and Wipke W.T, "Computer Assisted Design of Complex Organic Synthesis", Science 166, 1969

8. Dolk D.R. and Konsynski B.R., "Knowledge Representation for Model Management Systems", Working Paper, University of Arizona 1983

9. Duda R., Gaschnig J., Hart P., Konolige K., Reboh R., Barrett P., Slocum J., "Development of the PROSPECTOR consultation system for mineral exploration", SRI Projects 5821&6415, SRI International, 1978

10. Elam J.J., Henderson J.C. and Miller L.W., "Model Management Systems: An Approach to Decision Support in Complex Organizations", Proc. First Intl. Conf. on Information Systems, 1980

11. Gelernter H.L., Sanders A.F., Larsen D.L., Agarival K.K., Boivie R.H., Spritzer R.H., Searleman J.E., "Empirical Explorations of SYNCHEM", Science 197, 1977

12. Jarke M., Clifford J. and Vassiliou Y., "An Optimizing Prolog Front-

End to a Relational Query System", Proc. ACM-SIGMOD Intl. Conf. on Management of Data, Boston 1984

13. Jarke M. and Vassiliou, Y., "Coupling Expert Systems with Database Management Systems", in W.Reitman, Ed., Artificial Intelligence Applications for Business, Ablex, Norwood, NJ, 1984

14. Jordan C.W., Life Contingencies, The Society of Actuaries, Chicago 1975

15. Nilsson N., Principles of Artificial Intelligence, Springer 1982

16. Sivasankaran T., Intelligent Model Management in an Actuarial Consulting System, Ph.D. thesis, New York University 1984

17. Sprague R.H. and Carlson E.D., Building Effective Decision Support Systems, Prentice-Hall 1982

18. Vassiliou, Y., Clifford, J., Jarke, M., Access to Specific Declarative Knowledge in Expert System: The Impact of Logic Programming", Decision Support Systems 1, 1 (1984).

19. Wipke W.T, Braun H., Smith G., Choplin F. and Sieber W., "SECS-Simulation and Evaluation of Chemical Synthesis: Strategy and Planning", in Wipke W.T. and House W.J.(Eds.), Computer-assisted Organic Synthesis, American Chemical Society, Washington D.C. 1977

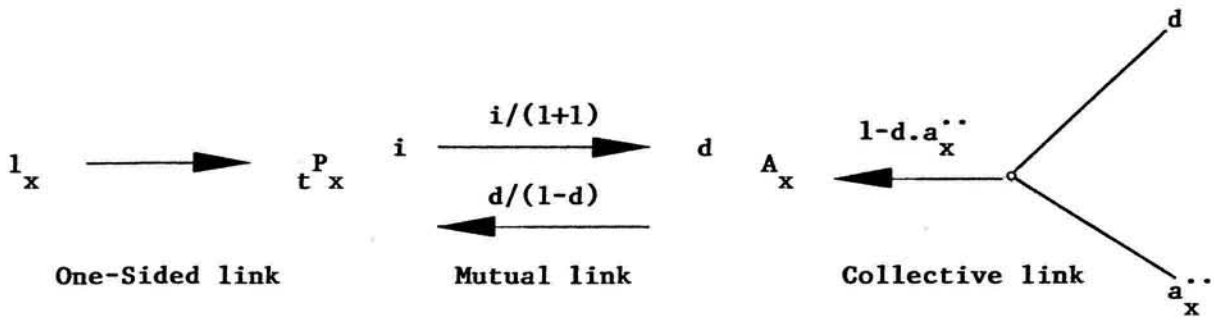

Figure 1: Links in Formula Derivation Networks

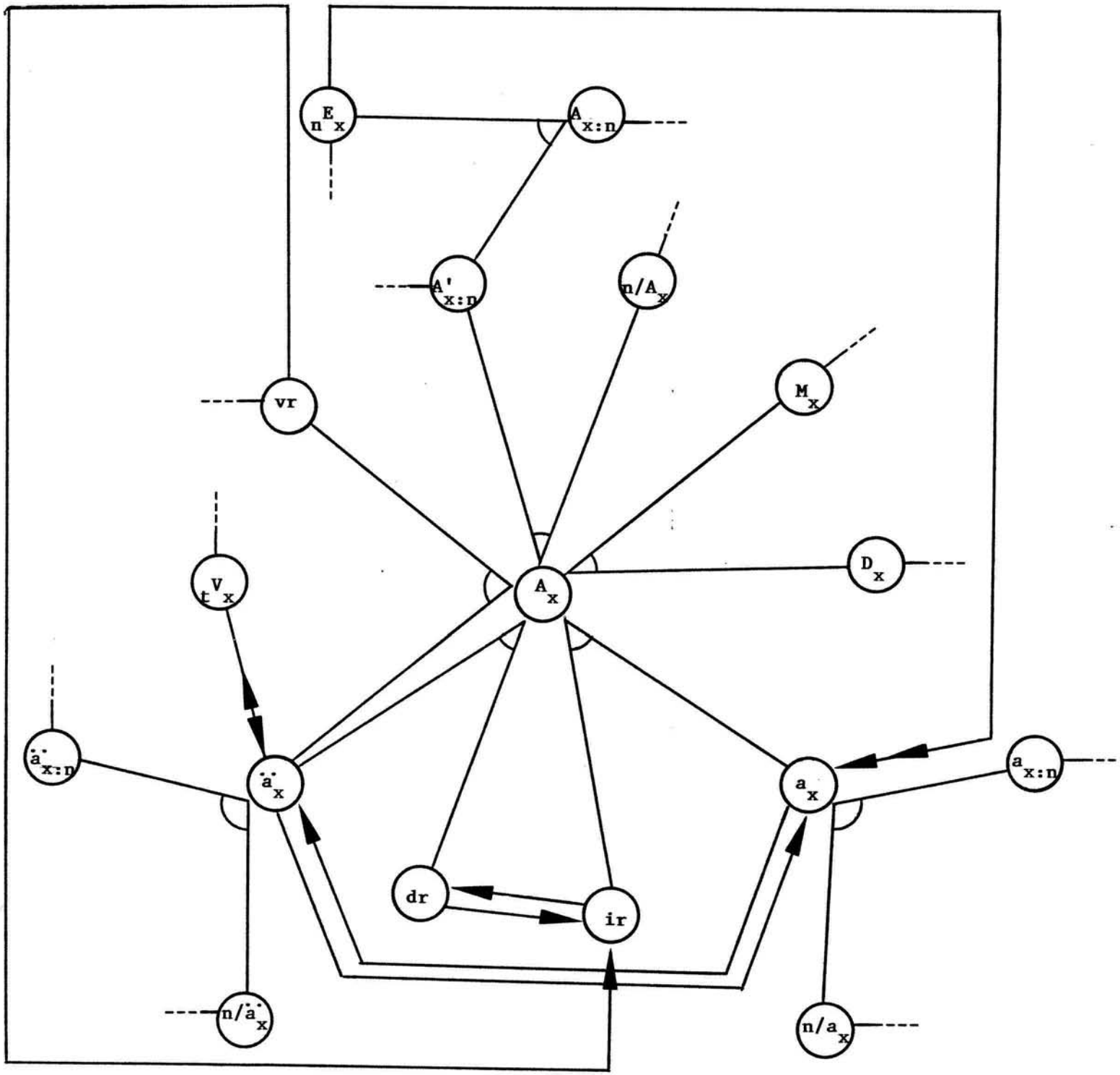

Figure 2: A Simple Derivation Structure

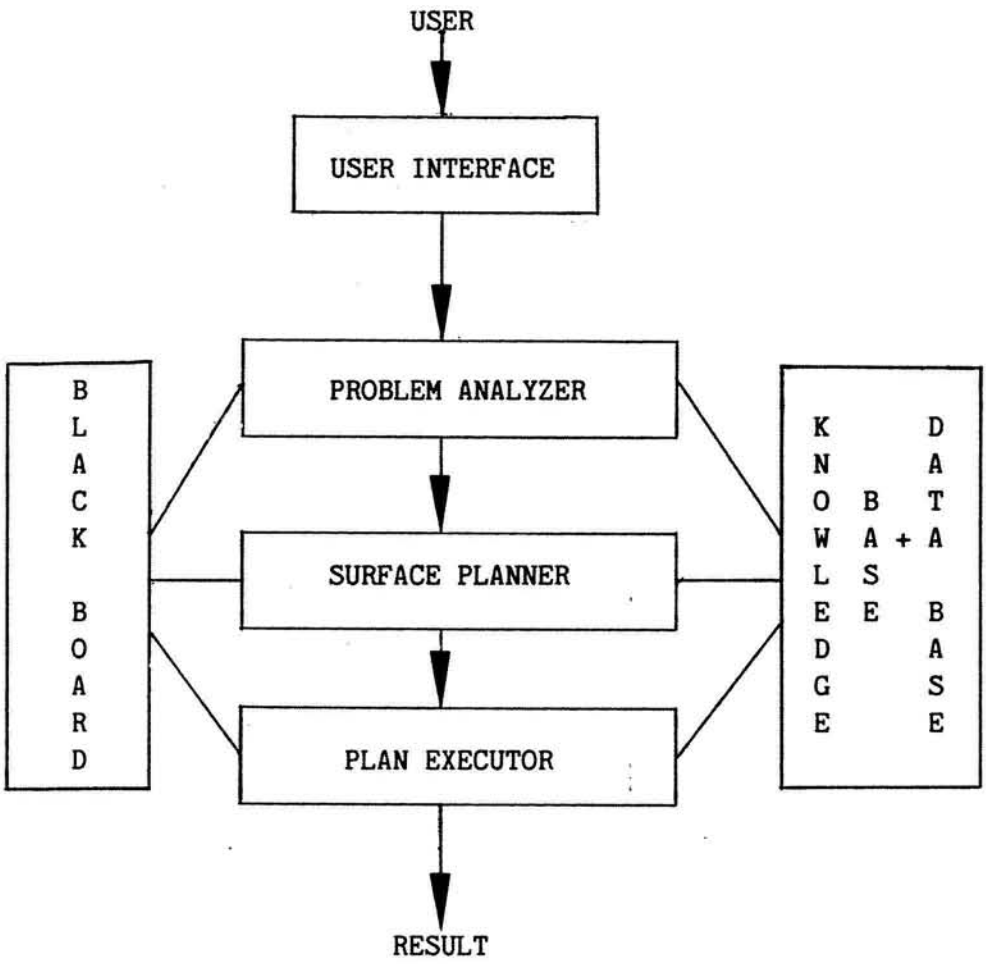

Figure 3

The Model of Model Management

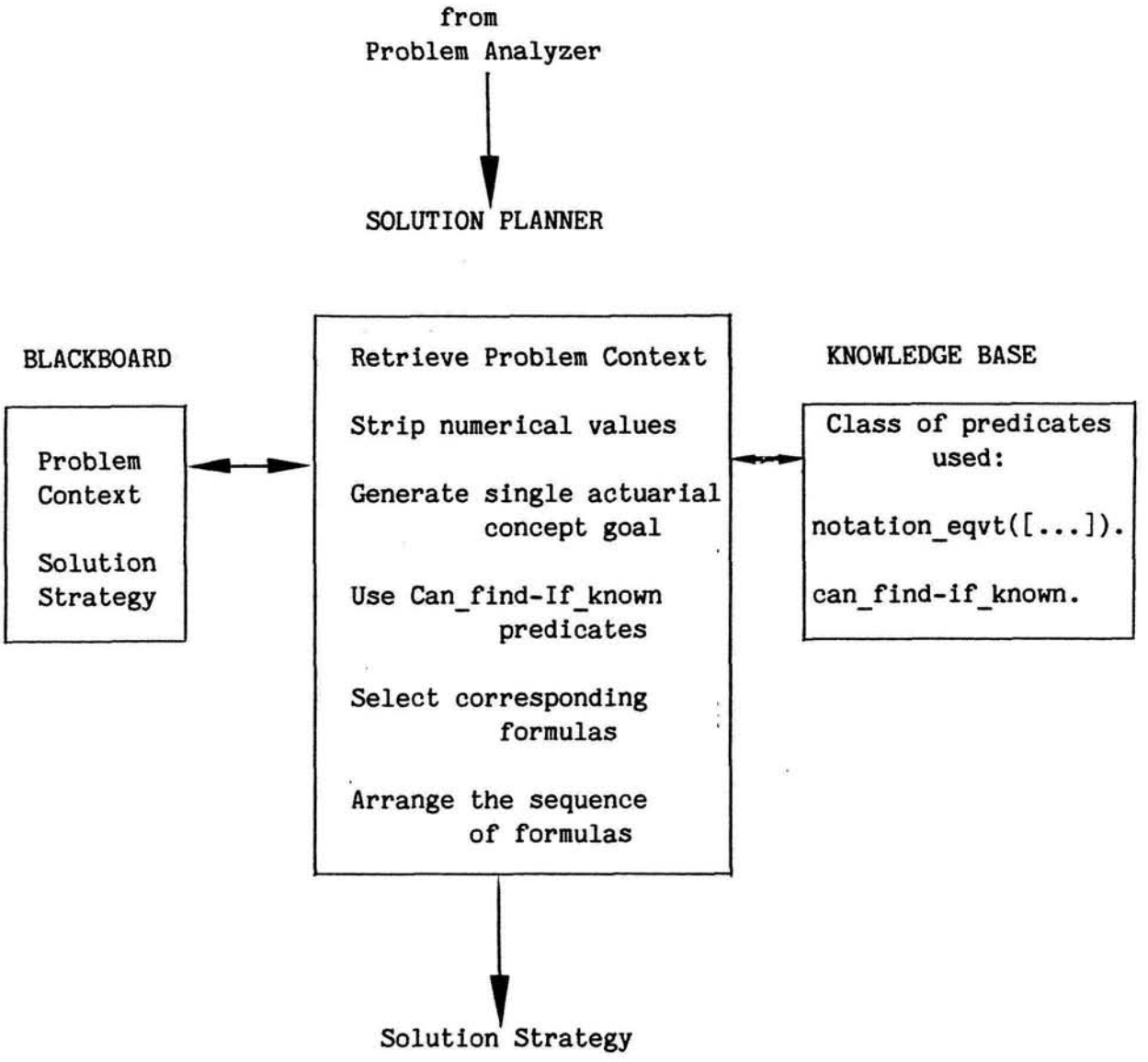

Figure 4

The Solution Planner Component

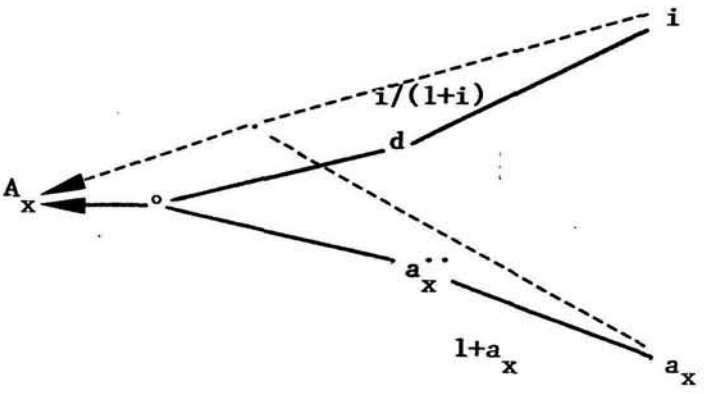

Figure 5: A Derived Link

KNOWLEDGE BASE

STATIC

- Knowledge about the user interface
- Types of goal category concepts
- Types of insurance benefits
- Actuarial notations
- Formulas
- Formula Derivation Nets
- Evaluation/computation procedures
- Data base (table) values

DYNAMIC

- Rules on problem recognition
- Rules on selecting the formulas
- Rules on manipulating the formulas
- Rules on creation and deletion of intermediate strategies/results

Figure 6: Structure of Knowledge Base

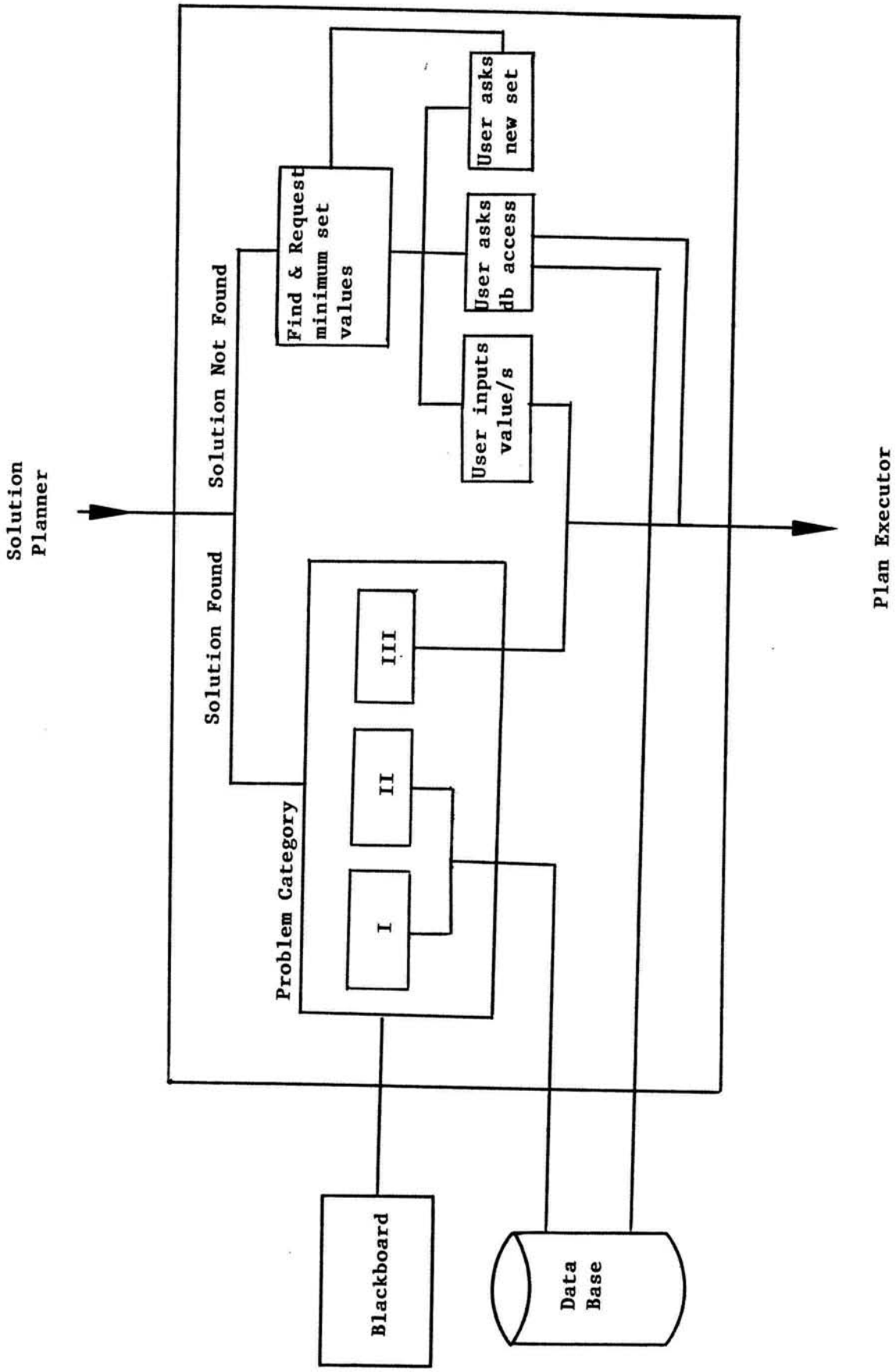

Figure 7: Model of Fact Acquisition