Reasoning in Model Management Systems

Ting-peng Liang

College of Commerce and Business Administration
Bureau of Economic and Business Research
University of Illinois, Urbana-Champaign

BEBR

FACULTY WORKING PAPER NO. 1403

College of Commerce and Business Administration

University of Illinois at Urbana-Champaign

October 1987

Ting-peng Liang, Assistant Professor
Department of Accountancy

Reasoning in Model Management Systems

REASONING IN MODEL MANAGEMENT SYSTEMS

ABSTRACT

Reasoning is crucial to successful development of knowledge-based model management systems. It is particularly important when model integration is needed in a large-scale system. In this case, heuristics must be used to reduce the complexity of model integration. This paper discusses the issues involved in the reasoning process. First, a hierarchy of abstractions that integrates previous contributions in model management is presented. Then, the modeling process is formulated as a planning process by which a set of operators are scheduled to achieve a specified goal. This process involves searches for alternatives that can eliminate the difference between the initial state and the goal state. Various reasoning strategies and heuristic evaluation functions that can be used to improve the efficiency of model integration are discussed.

INTRODUCTION

A model management system (MMS) is a software system that handles all accesses to the decision models stored in a model base. Its primary purpose is to enhance decision performance by providing decision makers with proper models. In order to achieve this goal, functions that support model creation, storage, retrieval, execution, and explanation are essential. Among them, model creation is the most knowledge-intensive function. It needs knowledge of both individual models and the modeling process.

There are two approaches for an MMS to support model creation: user-assisted modeling and automatic modeling. User-assisted modeling allows a decision maker to specify how several smaller models can be combined to become a larger one. The user is responsible for finding a set of appropriate models and determining the best way to integrate them. The MMS serves as a blackboard on which the user examines different alternatives. Because the system performs a limited amount of intellectual activities in this case, its implementation is easier compared to automatic modeling. In general, a good model manipulation language is adequate.

Instead of having the user tell the system what to do and how to do it, automatic modeling requires that the system automatically formulate a decision model for the user. A long-term goal for automatic modeling is to develop an MMS capable of designing decision models based on the problem description provided by the decision maker. This would allow the system to replace human model builders. Given current information technology, however, this goal is, at least in the near future, very difficult to achieve. One major difficulty is that a modeling process usually involves a huge amount of common sense -- a set of knowledge computers cannot yet handle.

At present a feasible goal for automatic modeling is to develop a system that can automatically find a model or a sequence of models already stored in the model base to produce the desired information. In many situations, a complex model is an appropriate integration of several basic models. Given a particular problem, therefore, a good MMS must be able to help the user locate and combine the relevant basic models in the model base. This process is called model integration [Liang, 1986]. The model base provides basic components for modeling and sets up a knowledge boundary, which allow effective use of the stored models without incurring the difficulty of common sense reasoning.

Developing the model integration capability needs both model representation schemes that logically represent each model in the model base and reasoning mechanisms that schedule models. In recent years, several model representation schemes have been developed, such as relational [Blanning, 1982-1986], logic-based [Bonczek, Holsapple & Whinston, 1981; Dutta & Basu, 1984; Kimbrough, 1986; Pan, Pick & Whinston, 1986], frame-based [Dolk & Konsynski, 1982-1986], and graph-based approaches [Elam, Henderson & Miller, 1980; Geoffrion, 1985-87; Liang, 1986-1987]. However, research in the reasoning side is far behind. Although the formalisms underlying those representation schemes may provide basic reasoning capabilities, they are not tailored to the need of model management. For example, a logic-based system can prove that a model exists, but its dependence on an exhaustive search may dramatically reduce its applicability to many problems. Therefore, unless an efficient reasoning mechanism for model management is developed, providing model integration support in MMSs would be very difficult.

In this article, reasoning issues involved in the model integration process are discussed from a planning perspective. Most problem solving processes, as stated by Simon (1981,1983), can be considered as planning processes by which a set of operators can be found and scheduled to eliminate the difference between the initial state and the goal state. This process involves search with the presence of subgoals, operators, macro-operators, and abstraction. Since a model integration process is part of the problem solving process, it can be analogously defined as a process by which available models are selected and scheduled to eliminate the difference between the desired information and the available information. In order to determine and eliminate the difference, the following issues are crucial:

(1) State representation: what information should be included in a state representation?

(2) Reasoning: what mechanisms can be used to eliminate the difference between the goal state and the initial state?

(3) Heuristic functions: how can the efficiency of the process be improved?

In the remainder of the article, they will be discussed by this sequence.

HIERARCHY OF ABSTRACTIONS: A REVIEW OF MODEL REPRESENTATION

To determine how a state should be represented one must first consider what level of abstraction one needs. Here the concept of abstraction is very important. In order to efficiently solve a complex problem, a problem solver must first ignore low level details and concentrate on the essential features of the problem. The details are filled after the problem has been solved at the higher level. Therefore, abstraction formation involves loss of content, which makes the abstraction simpler than its instantiation(s) [Darden, 1987, Korf, 1987]. This idea has been used for problem solving for a long time [Poya, 1957] and adopted by several general-purpose problem solving programs, including General Problem Solver (GPS) [Newell and Simon, 1972] and ABSTRIP [Sacerdoti, 1974].

In the model management arena, Dolk and Konsynski (1984) first adopted the term "abstraction" and presented a model abstraction technique. In fact, despite grounding on different formalisms, most model representation schemes presented in previous research reflect different levels of model abstraction, from user-oriented to execution-oriented. For example, Blanning (1982-86) emphasizes the manipulation of data relations (data level); Liang (1985-87) focuses on the mapping between inputs and outputs (model level); Geoffrion (1985-87) presents a hierarchical framework for structured modeling (structure level); and Dolk and Konsynski (1982-86) concentrate on model specification (specification level). Figure 1 illustrates these five levels of abstraction for the EOQ model that calculates economic order quantity from demand, holding cost, and ordering cost.

INSERT FIGURE 1

----

1. <u>Program level</u>

At the bottom of the hierarchy is a Pascal implementation of the EOQ model. This reflects a highly machine-oriented view of decision models. The advantage of this representation is that the model can be stored in a model base and be readily integrated and compiled for execution. Its major disadvantage, however, is that little information relevant to model management is provided. For example, it provides little input and output information and can be activated by model name only.

2. <u>Specification level</u>

By ignoring some implementation details, Dolk and Konsynski (1984) developed a frame-based model abstraction technique. This technique represents models by their data objects, procedures, and assertions, all expressed in first order predicate logic. An abstraction for the EOQ model is shown at the specification level in Figure 1.

The contents lost at this level of abstraction are implementation details and direct computer executability. However, it gains computer language independence. The same specification may have interfaces to programs in different computer languages. An early work conducted by Elam, Henderson, and Miller (1980) also represented models at a similar level of abstraction but in a graphical form. They adopted the concept of SI-nets in artificial intelligence.

Although this level of model abstraction may be useful to model builders, it still includes too much detail for most users. In addition, some low level operations such as checking data formats may substantially increase the complexity of model integration. These operations need not to be considered until a set of potential model combinations has been identified. Therefore, further abstracting is desirable.

3. <u>Structure</u> <u>level</u>

Geoffrion's framework for structured modeling further eliminates some integrity constraints and data formats. It portrays the fundamental structure of a model by its elemental structure, generic structure, and modular structure.

One feature of the structured modeling is the use of graphical symbols rather than text-based specifications, which makes the functional relationships among various modules very clear. In Geoffrion's framework, nodes stand for modeling elements and arcs stand for calls. A modular structure of the EOQ model, as shown in Figure 1, contains four nodes and three arcs.

A graph-based model structure provides certain insight into a model. For the purpose of model integration, however, representing the detailed structure may not be necessary in many situations. This is particularly true when the model is nondecomposable. For example, the mapping between demand and order quantity in the EOQ model cannot be used independently unless the ordering cost and holding cost are also present.

Therefore, the three arcs of the EOQ model are highly dependent and can be combined.

4. <u>Model level</u>

By considering each model stored in the model base as a mapping between input and output, Liang's approach uses a node to represent a set of data attributes and an arc to represent a set of functions that can be used to convert from one node to another. Since a model is composed of input, output, and a set of functions for converting input to output, it can be represented as a combination of two nodes and one arc. For example, the EOQ is a mapping between two sets, [Demand, O_cost, H_cost] and [Quantity].

Because each model in the model base is considered a single mapping, this approach allows an associated cost or validity value be estimated for each model. It is important when algorithms and heuristics in graph theory are used for model integration [Liang, 1987].

5. <u>Data level</u>

On top of the hierarchy is a relational framework of models. Instead of adopting artificial intelligence techniques, Blanning (1982-86) focuses on model manipulations including join and projection. It reflects a user-oriented view of model management -- the user obtains the desired information without the need to see all the details of calculation. For example, the EOQ model is considered a relation composed of demand, holding cost, ordering cost, and quantity.

The hierarchical view of model abstraction indicates that a model integration process includes the following steps:

(1) identify the desired information;

(2) develop a master plan for building a composite model when there is no single model available for producing the desired information. The master plan determines the sequence by which a set of models should be executed;

(3) determine the model structure and retrieve the corresponding model specifications according to the master plan;

(4) evaluate the feasibility of the master plan by checking details such as data format and model assumptions;

(5) combine selected programs to formulate an executable model, if the master plan is proven feasible. Otherwise, go to step 2 to develop another plan.

For example, if the inventory holding cost is not a constant but determined by a holding cost model with interest expenses and warehouse operation costs as its inputs, then calculating the economic order quantity involves an integration of two models: EOQ and the holding cost model. First, the MMS finds that the table (shown on the top of Figure 2) requested by the user contains information from more than one model. Then, the system develops a master plan for executing the selected models. In this example, the sequence is (1) the holding cost model, and (2) the EOQ model. Based on the master plan, the structure of the composite model can be determined. Figure 2 shows the corresponding representations at the data level, model level, and structure level. Finally, specifications and executable programs can be activated and executed.

INSERT FIGURE 2

REASONING IN MODEL INTEGRATION

The key step in the model integration process is the development of a master plan. Unless a sequence of models is found capable of producing the desired information, there is no need to the check integrity constraint or data format. Therefore, each state in the modeling process can be represented as a set of data attributes. Each basic model stored in the model base is considered an operator that converts one state into another. For instance, the EOQ model is an operator that converts the state, [Demand, O_cost, H_cost], to another state, [Quantity].

By these definitions, all available information constitutes the initial state of modeling and the desired information is the goal state. The process for developing a master plan can be described as a process by which operators are scheduled to convert the initial state to a specified goal state by way of achieving a set of subgoals sequentially.

Two issues are crucial to developing a master plan. First, how can the proper candidate models in the model base be selected? Second, how can the selected models be scheduled efficiently? In this section, various reasoning strategies and heuristics for improving the efficiency of model scheduling will be discussed.

REASONING STRATEGIES

Given the initial state and the desired goal state, the first step to developing a master plan is to choose a proper reasoning strategy. There are three generic strategies that may be employed for model integration: forward reasoning, backward reasoning, and bi-directional reasoning. Each strategy has its advantages and drawbacks.

The forward reasoning strategy requires the system to start from the initial state and search proper candidates based on the available information. Therefore, it is also called data-directed reasoning. This strategy is appropriate when the goal state is complex but the amount of initial information is relatively small. An algorithm that applies this strategy to model integration can be described as follows. The queue produced by this algorithm is the master plan for converting the initial state, INITIAL, to the goal state, GOAL.

```
REPEAT
   1. Find an operator, OP, that can convert a state, IN,
      to another state, OUT, where IN is a subset of the
      initial state, INITIAL;
   2. Add OP to a queue and define a new initial state,
      NEWSTATE = $(\mathrm{INITIAL} \cup \mathrm{OUT}) - \mathrm{IN}$;
   3. Set INITIAL = NEWSTATE
UNTIL $\mathrm{GOAL} \subset \mathrm{INITIAL}$.
```

The backward reasoning strategy is goal-directed, which requires the system to work backward from the goal state trying to prove it from the facts in the data base. It works well when the goal state contains relatively limited amount of outputs while the initial state includes a large amount of input information. A backward model integration process includes the following procedures. The resulting stack represents a master plan.

```
REPEAT
   1. Find an operator, OP, that converts IN to OUT, where
      $\mathrm{OUT} \cap \mathrm{GOAL} \neq \emptyset$;
```

```
        2. Add OP to a stack and define a subgoal state,
           SUBGOAL = (IN U GOAL) - OUT;
        3. Set GOAL = SUBGOAL
    UNTIL GOAL ⊂ INITIAL.
```

Bi-directional reasoning uses forward and backward strategies simultaneously. Its major advantage is that it decomposes a problem into two parts, which can reduce the complexity when the number of nodes at each step grows exponentially with the number of steps that have been taken [Rich, 1983, p.60]. The risk for using this strategy is that the two searches may pass each other, resulting in more work than it would have taken for either one of them.

Because the amount of available information usually is much larger than the amount of the desired information in most modeling situations, backward reasoning is more appropriate for developing a master plan. The other two strategies, however, may be useful to explain the modeling process and help the user understand the causal relationship between inputs and outputs.

Instead of defining the whole set of desired information as the goal state, a modified backward reasoning strategy, called difference elimination, focuses on the difference between the desired state and the initial state. Since part of the desired information may be readily available from the data base, ignoring the information available in the initial state can simplify the goal state and hence reduce the complexity of the reasoning process. In this case, the model integration process can be defined as a process by which the difference between the desired information and the available information can be completely eliminated. Its reasoning process is as follws:

```
REPEAT
   1. Set the goal state as the difference between the
      desired information and the available information,
      GOAL = DESIRED - INITIAL;
   2. Find an operator, OP, that converts IN to OUT, where
      $OUT \cap GOAL \neq \emptyset$;
   3. Add OP to a stack and define a subgoal, SUBGOAL =
      (IN U GOAL) - OUT;
   4. Set GOAL = SUBGOAL - INITIAL;
UNTIL GOAL = [].
```

HEURISTIC SEARCH

Although the difference elimination process may reduce the complexity of the goal state, it still relies on exhaustive search. In order to further improve the efficiency of the search process, two techniques may be used: macro-operators and heuristic evaluation functions.

1. Macro-operators

A macro-operator is a sequence of primitive operators. Because the sequence is predetermined, there is no need for search. In fact, the major purpose of model integration is to design a macro-operator that can be used to solve a particular problem. Then, how can we use macro-operators to improve the efficiency of modeling? A macro-operator is both the result of a modeling process and a tool for modeling. A master plan developed for solving a previous problem can be saved as a macro-operator for later use. When a new problem includes the previously solved problem as a subproblem, the macro-operator can be retrieved and fitted into the master plan directly. From this perspective, a model integration process is also a learning process by which macro-operators are learned.

Using macro-operators involves a tradeoff between model storage costs and modeling costs. On the one hand, the extensive use of macro-operators needs a large amount of computer memory for storage. On the other hand, lack of macro-operators needs extensive search in the modeling process. Therefore, an important issue here is to what extent macro-operators should be used.

In general, there are two criteria for determining proper use of macro-operators: functional dependency and frequency of use. A model is defined functionally dependent on another model if at least one input of the former is among the output of the latter. If there exists a set of functionally dependent models, they may be grouped into a macro-operator.

Determining functional dependency is also important for model scheduling. Model B must be scheduled before model A when model A is functionally dependent on model B. In addition, a set of functionally dependent models may result in a cycle, which traps the system into an infinite loop. In this case, mechanisms for detecting and resolving loops must be applied.

Another factor to be considered is frequency of use. If a particular sequence of operators is used frequently, then it may be appropriate to save them as a macro-operator. Otherwise, it may be unnecessary. For example, if the holding cost model and the EOQ model shown in Figure 2 usually are used together in an organization, then it may be stored as a macro-operator consisting these two models and mapping from [Demand, O_cost, W_cost, I_cost] to [Quantity].

2. Heuristic Search

In addition to macro-operators, heuristic functions may be used to guide a search process. The major purpose of a heuristic function is to estimate how close a particular state is to the goal state so that the system can select the best search direction accordingly. In general, a heuristic evaluation function provides a numerical estimation of the promise of a state, which may depend on the criteria used in the function, the description of the goal, and the information gathered by the search up to that point.

Intuitively there are several criteria that may be used to estimate the promise of a state, such as model applicability, machine capacity, modeling costs, and user preference. Model applicability uses context-dependent measures to evaluate the applicability of a model to a particular problem. For example, when the problem is identified as an inventory problem, then the EOQ model may have a higher value compared to a capital budgeting model.

Machine capacity or modeling cost requires the heuristic evaluation function to assess the anticipated machine capacity or modeling cost for each state and then select the best direction to further explore. It focuses on developing efficient or cost-effective models. User preference is a subjective measure of model applicability. Its goal is to develop a model most preferred by a particular user.

A major drawback of these criteria is that it is difficult to find objective measures. For instance, it is unclear what should be measured when we estimated the applicability of a model. Therefore, a strictly objective criterion is desirable.

One feasible criterion is the distance between the goal state to the resulting subgoal state after applying the model. Since each state represents a set of data attributes, the distance between two states can be defined as the difference between the amounts of items contained in those two states. For example, the distance between state A and state B in Figure 3 is 1 because the arc reduces the number of items in the goal state from 5 to 4.

INSERT FIGURE 3

Because the difference elimination process is basically a backward reasoning process, the system should pursue an operator that results in a state far from the goal state. In other words, the system will select the model that eliminates the largest amount of items. This allows the resulting plan to have a minimum number of basic models.

By taking advantage of the distance-based heuristic evaluation function, the reasoning process can be modified as follows:

```
REPEAT
  1. Set the goal state as the difference between the
     desired information and the available information,
     GOAL = DESIRED - INITIAL;
  2. Find all operators, OPS, each of which results in an
     OUT state, where $OUT \cap GOAL \neq \emptyset$;
  3. Check functional dependencies and cyclicity to
     exclude the operators functionally dependent on
     others and remove loops;
```

```
4. Determine the resulting subgoal for each of the
   remaining operators that converts IN to OUT,

   SUBGOAL = (IN ∪ GOAL) - OUT;

5. Calculate the distance for each of the possible
   subgoals,

   Distance(GOAL,SUBGOAL) = Item(GOAL) - Item(SUBGOAL);

6. Select the operator with the longest distance, OP,
   which converts IN to OUT;

7. Add OP to a stack and define a subgoal, SUBGOAL =
   (IN U GOAL) - OUT;

8. Set GOAL = SUBGOAL - INITIAL;

UNTIL GOAL = [].
```

AN EXAMPLE

In order to illustrate how the difference elimination process develops a master plan for model integration, a prolog implementation and an example is presented in this section. As listed in Appendix 1, the prolog implementation includes the following modules: (1) main planning module for defining goals and subgoals, (2) difference determination module for finding the difference, (3) dependency checking module, and (4) heuristic evaluation function for selecting the best operator.

Assume that we have the following models in our model base:

(1) {m1, [a,b,s,t], [w,x,y]}

(2) {m2, [c,e,r,t],[v,x]}

(3) {m3, [c,d,r], [v,s]}

(4) {m4, [a,c,e], [u]}

(5) {m5, [d,f], [r,y]}

(6) {m6, [a,b,f], [t]}

Further assume that the data base contains data of [a,b,c,d,e,f] and the decision maker needs a model providing [u,v,w,x,y], then the modeling process is as shown in Figure 4. The system first applies model m1 to reduce the difference from [u,v,w,x,y] to [u,v,s,t]. Then, model m3 and m4 are applied to further reduce the difference to [u,t,r] and [t,r] respectively. Finally, model m5 and m6 are used to eliminate the difference, and the master plan is developed as [m6,m5,m4,m3,m1].

INSERT FIGURE 4

CONCLUDING REMARKS

Developing a master plan is a key step to model integration. In this paper, several major issues have been discussed. First, a hierarchy of model abstraction has been described. Then, various reasoning strategies have been presented. They include forward, backward, bi-directional, and a modified backward strategy called difference elimination. Finally, a heuristic evaluation function for improving the reasoning efficiency has been developed.

Since reasoning in model management is very complex, much further research is required before a practical system can be developed. Potential research directions include the following. First, how can various levels of abstraction be integrated into a single system? Efficient algorithms must be developed to link representations at different levels. Second, how can operators and macro-operators be determined? In this article, this issue has been briefly discussed and two criteria have been described: functional dependency and frequency of use. Detailed discussions are needed. Third, how can heuristic evaluation functions be developed and evaluated. A measure of the distance between two states and a distance-based heuristic evaluation function are presented in this article. This, however, should not be the only way to do it. Therefore, much research is needed to explore alternative measures of the distance between states and evaluate various measures to find the most appropriate one.

Appendix 1: An Implementation of the Reasoning Mechanism

```
/* Model base */

arc(m1,[a,b,s,t],[w,x,y]).
arc(m2,[c,e,r,t],[v,x]).
arc(m3,[c,d,r],[v,s]).
arc(m4,[a,c,e],[u]).
arc(m5,[d,f],[r,y]).
arc(m6,[a,b,f],[t]).

/*  A Prolog Implementation  */

model:-write('Output attributes = '), read(GOAL),
       write('Input Attributes = '), read(START),nl,
       write('** Modeling goal = '), write(GOAL),nl,nl,
       modeling(START,GOAL,START,ACTS),nl,
       write('*** The master plan is to execute the following'),
       nl,write('models sequentially: '),nl,nl,
       write(ACTS).

modeling(STARTSTATE,GOAL,STARTSTATE,[]):-
   satisfy(STARTSTATE,GOAL),!.

modeling(STARTSTATE,GOAL,ENDSTATE,ACTS):-
   majordiff(STARTSTATE,GOAL,DIFF),
   findall_acts(STARTSTATE,DIFF,ALLACT),
   write('All possible candidates = '), write(ALLACT),nl,
   check_dependency(ALLACT,RESTACT),
   write('Independent candidate = '),write(RESTACT),nl,
   heuristic(RESTACT,ACT,V),
   write('Selected action by a heuristic = '),write(ACT),nl,
   subgoal(STARTSTATE,DIFF,ACT,START1,SUBGOAL),
   nl,write('** New subgoal = '),write(SUBGOAL),nl,nl,
   modeling(START1,SUBGOAL,_,ACT1),
   append(ACT1,ACT,ACTS).

satisfy(STARTSTATE,GOAL):-
   subset(GOAL,STARTSTATE).

check_dependency([X],[X]).
check_dependency(ALLACT,RESTACT):-
   all_in(ALLACT,TIN),
   del_prereq(ALLACT,TIN,RESTACT).

all_in([],[]).
all_in([a(ACT,V)],TIN):-
   arc(ACT,TIN).
all_in([a(ACT,V)|REST],TIN):-
   arc(ACT,IN,OUT),
   all_in(REST,TIN1),
   union(IN,TIN1,TIN).
```

```
del_prereq([],_,[]).
del_prereq([a(ACT,V)|REST],TIN,RESTACT):-
   arc(ACT,IN,OUT),
   subset(OUT,TIN),
   del_prereq(REST,TIN,RESTACT).
del_prereq([a(ACT,V)|REST],TIN,[a(ACT,V)|RESTACT]):-
   del_prereq(REST,TIN,RESTACT).

/* Determining major difference between START and GOAL */

majordiff(_,[],[]).
majordiff(START,[X|R],Z):-
   member(X,START),!,
   majordiff(START,R,Z).
majordiff(START,[X|R],[X|Z]):-
   majordiff(START,R,Z).

/* Submodels required for bridging the difference between
   the starting state and the goal  */

findall_acts(STARTSTATE,DIFF,ALLACT):-
   arc(ACT,IN,OUT),
   intersection(OUT,DIFF,CONT),
   CONT\=[],
   listlength(CONT,L1),
   majordiff(STARTSTATE,IN,NEG),
   listlength(NEG,L2),
   L is L1 - L2,
   assertz(stack(a(ACT,L))),fail;
   assertz(stack(a(end,[]))),
   collect(ALLACT).

collect(ALLACT):-
   retract(stack(X)),!,
   (X == a(end,[]),!, ALLACT = [];
    ALLACT = [X|REST],collect(REST)).

heuristic([a(ACT,V)],[ACT],V).
heuristic([a(ACT1,V1)|REST],[ACT1],V1):-
   heuristic(REST,[BestACT],VMax),
   V1 >= VMax.
heuristic([a(ACT1,V1)|REST],[BestACT],VMax):-
   heuristic(REST,[BestACT],VMax),
   V1 < VMax.

/* finding subgoals */

subgoal(STARTSTATE,DIFF,[ACT],START1,SUBGOAL):-
   arc(ACT,IN,OUT),
   union(STARTSTATE,OUT,START1),
   union(DIFF,IN,GOAL),
   majordiff(START1,GOAL,SUBGOAL).
```

```
/* Utilities */

append([],L,L).
append([A|X],Y,[A|Z]):- append(X,Y,Z).

member(X,[X|_]).
member(X,[_|Y]):-member(X,Y).

subset([A|X],Y):-member(A,Y),subset(X,Y).
subset([],Y).

intersection([],X,[]).
intersection([A|R],Y,[A|Z]):-
   member(A,Y),!,intersection(R,Y,Z).
intersection([A|R],Y,Z):-intersection(R,Y,Z).

listlength([],0).
listlength([H|R],L):-
   listlength(R,L1),
   L is L1 + 1.

union([],X,X).
union([H|T],Y,Z):-member(H,Y),!,union(T,Y,Z).
union([H|T],Y,[H|Z]):-union(T,Y,Z).
```

REFERENCES

[1] Blanning, R. W. (1982), "A Relational Framework for Model Management in Decision Support Systems", DSS-82 Transactions, pp.16-28.

[2] Blanning, R.W. (1985a), "A Relational Framework for Join Implementation in Model Management Systems," Decision Support Systems, 1:1, pp.69-81.

[3] Blanning, R.W. (1985b), "A Relational Framework for Assertion Management," Decision Support Systems, 1:2, pp.167-172.

[4] Blanning, R.W. (1986), "An Entity-Relationship Approach to Model Management," Decision Support Systems, 2:1, pp.65-72.

[5] Bonczek, R.H., Holsapple, C.W., and Whinston, A.B. (1981a), Foundations of Decision Support Systems, New York: Academic Press.

[6] Darden, L. (1987), "Viewing the History of Science as Compiled Hindsight," AI Magazine, 8:2, pp.33-41.

[7] Dolk, D.R. (1986), "Data as Models: An Approach to Implementing Model Management," Decision Support Systems, 2:1, pp.73-80.

[8] Dolk, D.R. and Konsynski, B.R. (1984), "Knowledge Representation for Model Management Systems," IEEE Transactions on Software Engineering, SE-10:6, pp.619-628.

[9] Dutta, A. and Basu, A.(1984),"An Artificial Intelligence Approach to Model Management in Decision Support Systems," IEEE Computer, 17:9, pp.89-97.

[10] Elam,J.J., Henderson,J.C. and Miller,L.W.(1980), "Model Management Systems: An Approach to Decision Support in Complex Organizations", Proceedings of the First Inter-National Conference on Information System, pp.98-110.

[11] Geoffrion, A.M. (1985), Structured Modeling, monograph, Graduate School of Management, University of California, Los Angeles.

[12] Geoffrion, A.M. (1987), "Introduction to Structured Modeling," Management Science, 33:5, pp.547-588.

[13] Kimbrough, S.O. (1986), "A Graph Representation for Management of Logic Models," Decision Support Systems, 2:1, pp.27-37.

[14] Konsynski, B. and Dolk, D.R. (1982), "Knowledge Abstractions in Model Management", DSS-82 Transactions, pp.187-202.

[15] Korf, R.E. (1985), "Macro-operators: A Weak Method for Learning," Artificial Intelligence, 26, pp.35-77.

[16] Korf, R.E. (1987), "Planning as Search: A Quantitative Approach," Artificial Intelligence, 33, pp.65-88.

[17] Liang, T. P. (1985), "Integrating Model Management with Data Management in Decision Support Systems," Decision Support Systems, 1:3, pp.221-232.

[18] Liang, T. P. (1986), "A Graph-based Approach to Model Management," Proceedings of the Seventh International Conference on Information Systems, San Diego, CA.

[19] Liang, T.P., and Jones, C.V.(1987), "Meta-design Considerations in Developing Model Management Systems," Decision Sciences, (forthcoming).

[20] Muhanna, W.A. and Pick, R.A. (1986), "A Systems Framework for Model Management," Working Paper, University of Wisconsin at Madison (December).

[21] Newell, A. and Simon, H. (1972), Human Problem Solving, Englewood Cliffs, N.J.: Prentice-Hall.

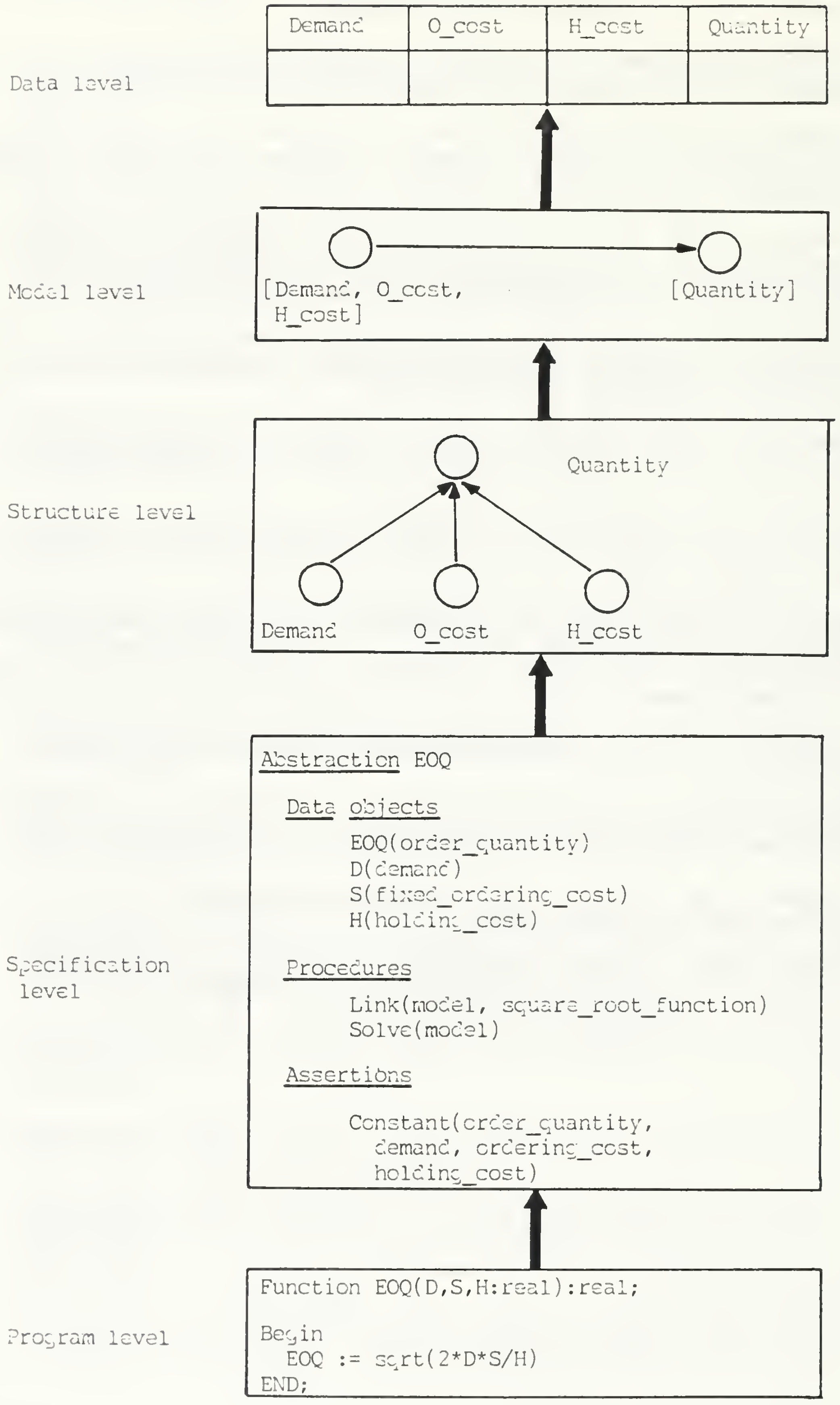
[22] Pan, S., Pick, R.A. and Whinston, A.B. (1986), "A Formal Approach to Decision Support," in S.K. Chang (ed.), Management and Office Information Systems, New York, NY: Plenum Press.

[23] Poya, G. (1957), How to Solve It, Garden City, NY: Doubleday.

[24] Sacerdoti, E. (1974), "Planning in a Hierarchy of Abstraction Spaces," Artificial Intelligence, 5, pp.115-135.

[25] Simon, H.A. (1981), The Science of the Artificial, Cambridge, MA: MIT Press.

[26] Simon, H.A. (1983), "Search and Reasoning in Problem Solving," Artificial Intelligence, 21, pp.7-29.



```
Abstraction EOQ

   Data objects

        EOQ(order_quantity)
        D(demand)
        S(fixed_ordering_cost)
        H(holding_cost)

   Procedures

        Link(model, square_root_function)
        Solve(model)

   Assertions

        Constant(order_quantity,
          demand, ordering_cost,
          holding_cost)
```

```
Function EOQ(D,S,H:real):real;

Begin
  EOQ := sqrt(2*D*S/H)
END;
```

Figure 1. A Hierarchy of Abstractions for EOQ Model

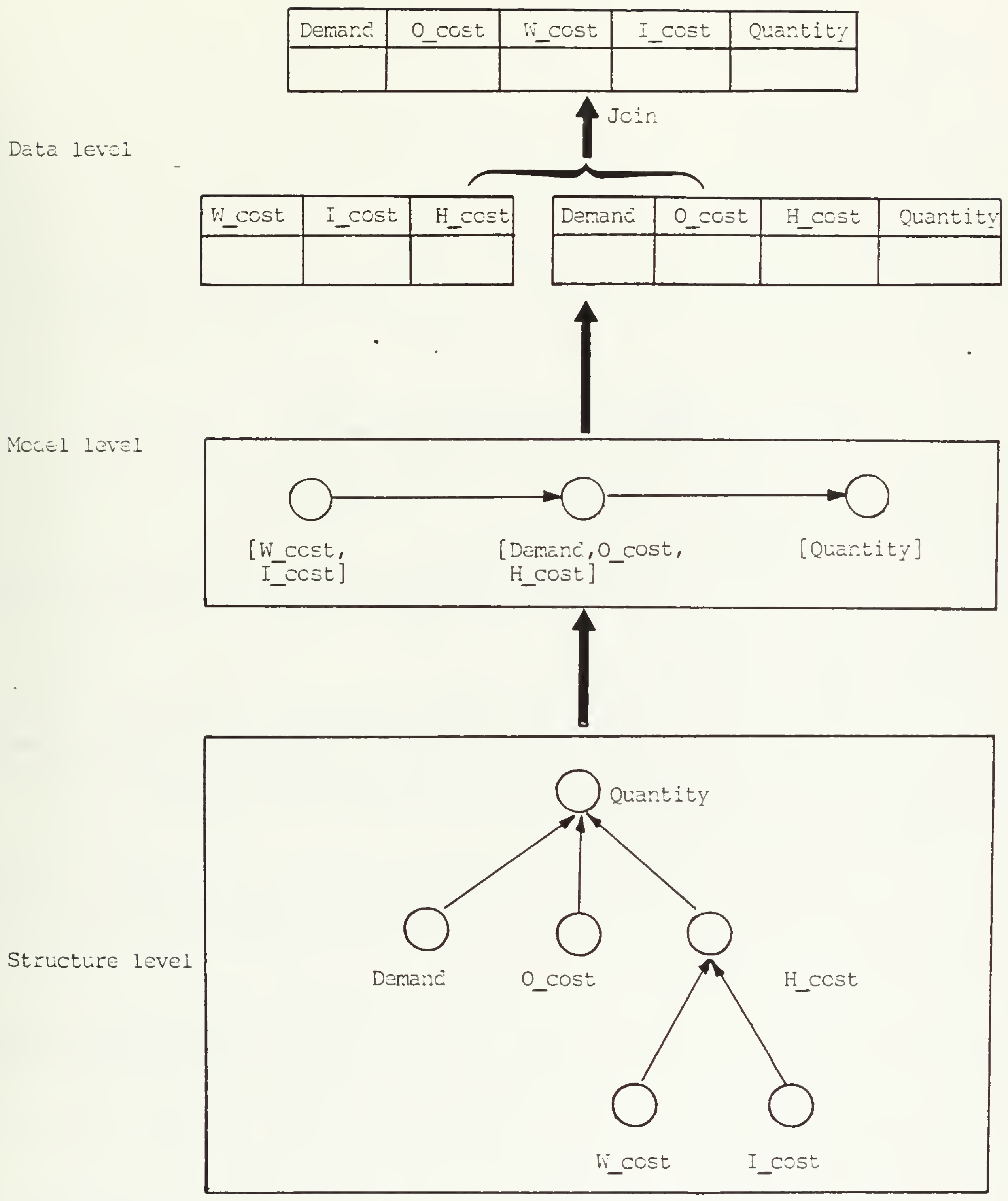
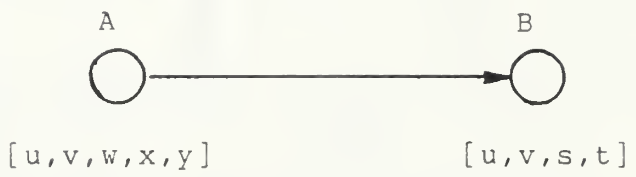


Figure 2. Different Abstractions of a Model Integration Process



Distance(A,B) = 5-4 = 1

Figure 3. Distance between two states

```
utput attributes = [u,v,w,x,y].
nput Attributes = [a,b,c,d,e,f].

* Modeling goal = [u,v,w,x,y]

ll possible candidates = [a(m1,1),a(m2,0),a(m3,0),a(m4,1),a(m5,1)]
ndependent candidate = [a(m1,1),a(m2,0),a(m3,0),a(m4,1),a(m5,1)]
elected action by a heuristic = [m1]

* New subgoal = [u,v,s,t]

ll possible candidates = [a(m2,-1),a(m3,1),a(m4,1),a(m6,1)]
ndependent candidate = [a(m2,-1),a(m3,1),a(m4,1)]
elected action by a heuristic = [m3]

* New subgoal = [u,t,r]

ll possible candidates = [a(m4,1),a(m5,1),a(m6,1)]
ndependent candidate = [a(m4,1),a(m5,1),a(m6,1)]
elected action by a heuristic = [m4]

* New subgoal = [t,r]

ll possible candidates = [a(m5,1),a(m6,1)]
ndependent candidate = [a(m5,1),a(m6,1)]
elected action by a heuristic = [m5]

* New subgoal = [t]

ll possible candidates = [a(m6,1)]
ndependent candidate = [a(m6,1)]
elected action by a heuristic = [m6]

* New subgoal = nil


** The master plan is to execute the following
odels sequentially:

m6,m5,m4,m3,m1]
```

Figure 4. A Sample Session