# PROBLEM SOLVING APPROACH IN DATA MANAGEMENT

William D. Haseraan
Carnegie-Melion University
Pittsburgh, Pennsylvania

Andrew B. Whinston
Purdue University
Lafayette, Indiana

## Abstract

This paper looks at an artificial intelligence control program required for a system which accepts queries to a data base which consists of both data and program modules. Using a problem solving approach, the data base structure and the program modules specify the initial world model. The operators consist of functions which look through sets in the data base and the actual program modules themselves. The goal is the query which requests data and/or processes. The control program determines the path which maps through the data base and the program modules in a manner which satisfies the predicate calculus constraints and the goal. Real examples of how this system is being used for a large scale data base is included.

## Introduction

This paper presents a problem in the data management area and demonstrates how that problem can be solved using an Artificial Intelligence Technique known as Problem Solving. The problem is that of determining a data path through a data base which consists of data and programs in order to determine the desired answer to a user's request. The data structure is viewed as the initial state, the queries provide the goal, and the various functions in the data base provide the operators which are used to map from the initial state to the goal. This paper will primarily be concerned with describing this problem and the development of its solution and then discussing the particular algorithm involved in the solution. The work presented here was a development made in the data management area and is not simply a new application developed for an Artificial Intelligence Algorithm. The following, then, is a description of the problem.

One of the many concerns facing organizations, both in the private and public sector, deals with the need to analyze as well as report information from their data bases. There is an increasing demand for these organizations to perform such planning functions involving forecasting, simulation, and optimizations. The GPIAN [1] research project at .Purdue University is concerned with developing a framework for a generalized planning system which will help to satisfy that demand. The system being developed will provide the planner with an environment consisting of data, programs for generating reports using this data, and a collection of models (programs) which can easily be used to analyze the data. The interface between the user and this environment will be English-like interactive query language. The following is a brief discussion of the major components of the CPIAN system. (Figure 1)

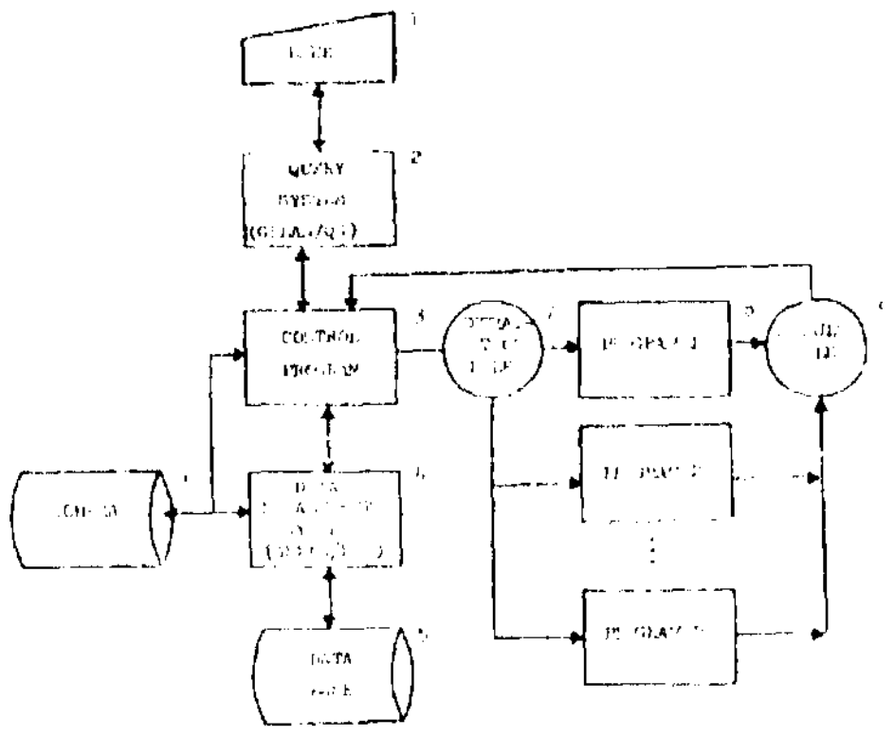


**Figure 1. GPIAN Framework**

The user (1) interfaces to the system through an interactive query language known as the GPIAN/QA [2]. The query system (2) accepts the user's request as stated in the query language, parses and verifies the syntax, and stores the query in a standard matrix form. All of the tables used in the parsing are actually stored in the data base to permit each application to have its own set of keywords and operators. A more thorough discussion of the query language can be found in user's manual [2] and paper by Haseman and Whtnston [3].

After the query has been parsed, the control program (3) analyzes the query and determines what action will be required to answer the request. These actions not only include generating calls to the data base through the Data Manipulation Language (DML), but also include the execution of functions (programs) which will operate on the data retrieved via the DML calls. This process will be discussed in more detail in the next section of the paper, while the following is a description of the rest of the system.

The core of the system is a Data Management System (3, A, 5), which is called the GPLAN/DMS |"4] [5] system. The DMS was developed using the specifications of the CODASYL DBTG Report [6] and will support data structures which range from sequential files to complex network data structures. The data structure which is supported by the query language is a hierarchical data structure. The schema consists of a group of tables

which describe the logical structure of the data base while the physical data is stored in the data base. The system generates the schema based on a user's description of the data structure using the Data Description Language (DDL). The basic unit of data is known as an item-type, where a group of item-types forms a record-type. The user can also define set-types which generate an owner-membership relation between two record-types. The DML commands are used by the control program to search through the various set relationships to determine the desired data.

The final component of the system is a collection of functions (programs) which will perform various operations on the data involved. In general, we can have any collection of application programs. Current implementation has the following: report generators, plotting, simulation, and optimizations. These functions receive their input from files generated from the data base by control program and may return the output to the user as well as store results back into the data base.

Using this introduction to the GPIAN system as background information, the following, then, is a description of the problem of how to design a control program to intermix DML calls on the data base and functions on this data to answer more involved queries. This can be viewed as a path finding problem.

## Problem

The previous approach used to answer a query involves determining the shortest path through the data structure which includes all the item-types which were requested in the query. For example, assuming the data structure as shown in Figure 2a, the following query could be requested

LIST REACH-NUMBER AND FLOW-RATE.

The algorithm would determine that the correct path would involve $S_1$ and S3. Assuming each reach contains three measurements of flow, the following response would be obtained:

| REACH-NUMBER | FLW-RATE |
|---|---|
| I | 110 |
| 1 | 114. |
| 1 | 125 |
| 2 | 205. |
| 2 | 230. |
| 2 | 270. |

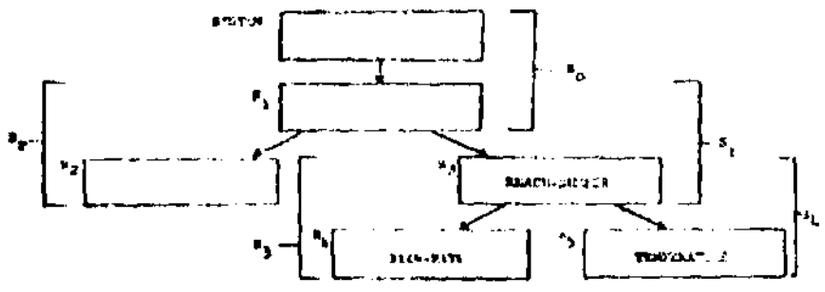
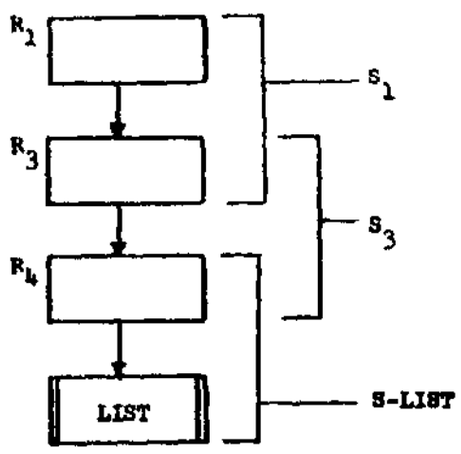


Figure 2a. Example Set Structure

| | | |
|---|---|---|
| RECORD | $R_1$ | |
| RECORD | $R_2$ | |
| RECORD | $R_3$ | |
| ITEM | REACH-NUMBER | INTEGER |
| RECORD | $R_4$ | |
| ITEM | FLOW-RATE | REAL |
| RECORD | $R_5$ | |
| ITEM | TEMPERATURE | REAL |
| SET | $S_o$ | |
| OWNER | SYSTEM | |
| MEMBER | $R_1$ | |
| SET | $S_1$ | |
| OWNER | $R_1$ | |
| MEMBER | $R_2$ | |
| SET | $S_2$ | |
| OWNER | $R_1$ | |
| MEMBER | $R_2$ | |
| SET | $S_3$ | |
| OWNER | $R_3$ | |
| MEMBER | $R_4$ | |
| SET | $S_4$ | |
| OWNER | $R_3$ | |
| MEMBER | $R_5$ | |

Figure 2b. DDL for Above Example

and the derived data path would appear as follows




As can be seen, if each set has only one owner record-type and one member record-type, the data path can be completely determined by the sets involved. Extending the concept of currency as discussed in the CODASYL Report, the following is a definition of a current set:

> A current set is a set which is used to describe a data path required for answering a query.

One of the problems with this approach of answering a query is that functions can only be attached at the end of a data path; in other words, all the data along the path is collected and then passed to the function. The following query cannot be answered using the above approach

without a specialized average function:

LIST REACH-NUMBER AND AVERAGE (FLOW-RATE)

This query essentially requires that a function be Inserted into the data structure, and therefore requires a restructuring of the data base. A second type of query which cannot be handled with the above approach is:

LIST REACH-NUMBER, FLOW-RATE , AND TEMPERATURE

since this request involves splitting a path at the logical level. Assuming that the user specifies that there exists a relationship between FLOW-RATE and TEMPERATURE (say, a one-to-one relationship), the system should be able to answer this query.

It was these two requirements which lead to the development of a control program using A.I. techniques. The following, then, is a description of the formulation of the problem in terms of the initial state, the goal, and the operators available to transform the initial state into the goal. It is assumed that the reader is familiar with predicate calculus as discussed by Nilsson [7] and with theorem proving algorithms such as discussed by Fikes and Nilsson [8].

## Initial State

The purpose of the Initial world model is to describe the initial state of the system in terms of a group of well-formed formulas (wffs). These wffs provide a description of what the theorem proving system is given as known's or truths. As a particular solution is generated, various operators will change or modify this initial model to represent the current state at that point in the solution. This is done in terms of generating a list of those wffs which should be removed from the initial state, and those which should be added to the list.

In terms of the specific problem addressed in this paper, the initial world model contains a description of the initial data structure as described in the data description language. The wffs include definitions of which variables are item-types, record-types, and set-types, as well as a description of the item-types which compose the various record-types, and various record-types which form the various set relationships. Additional wffs can be defined which will provide a description of the relationships such as was discussed previously with the TEMPERATURE and FLOW-RATE problem. These relationships can either be defined as an addition to the DDL, or interactively by the user.

The wffs which would describe the data structure shown in Figure 2 are as follows:

TYPE (R , RECORD)

TYPE (R , RECORD)

TYPE (R , RECORD)

TYPE ($R_4$, RECORD)

TYPE ($R_5$, RECORD)

TYPE (REACH-NUMBER, ITEM)

TYPE (FLOW-RATE, ITEM)

TYPE (TEMPERATURE, ITEM)

TYPE (SO, SET)

TYPE ($S_1$, SET)

TYPE ($S_2$, SET)

TYPE ($S_3$, SET)

TYPE ($S_4$, SET)



TYPE (REL, RELATI0NSHIP)

BELONG (REACH-NUMBER, $R_3$)

BELONG (FLOW-RATE, R )

BELONG (TEMPERATURE, R )

CONNECT ($S_o$, SYSTEM, $R_1$

CONNECT (S , R , $R_3$)

CONNECT ($S_2$, R , $R_2$)

CONNECT ($S_3$, $R_3$, $R_4$)

CONNECT ($S_4$ $R_{3_3}$ $R_{4_4}$)

CONNECT (REL, $R_4$, $R_5$)

STATUS (S , ON)

STATUS ($S_1$ OFF)

STATUS ($S_{„}$, OFF)

STATUS ($S_3$, OFF)

STATUS ($S_4$, OFF)

STATUS (REL, OFF)

where the following wffs are defined as:

TYPE - define the type of a variable

BELONG - define which record-type contains an item-type

CONNECT - define which records form a set or relationship

STATUS - define whether the set or relationship is currently connected (determines path through data base)

Two additional wffs are required for determining uniqueness. The first states that a particular variable cannot be more than one type:

(V V V X V Y) [TYPE(V,X) A (X Y)]

TYPE (V,Y)]

The second states that an item-type can only belong to one record-type:

(V V V x V Y) {[BELONG (V,X) A (X # Y)] =~

BELONG (V,Y)}

## Specifying the Goal

The goal is specified by the user in terms of the query language. Although the query can contain arithmetic and logical operators, the portion of the query which is critical to the control program is the item-types involved and any functions which are specified to operate on those item-types. The goal essentially becomes a list of item-types and functions which must appear

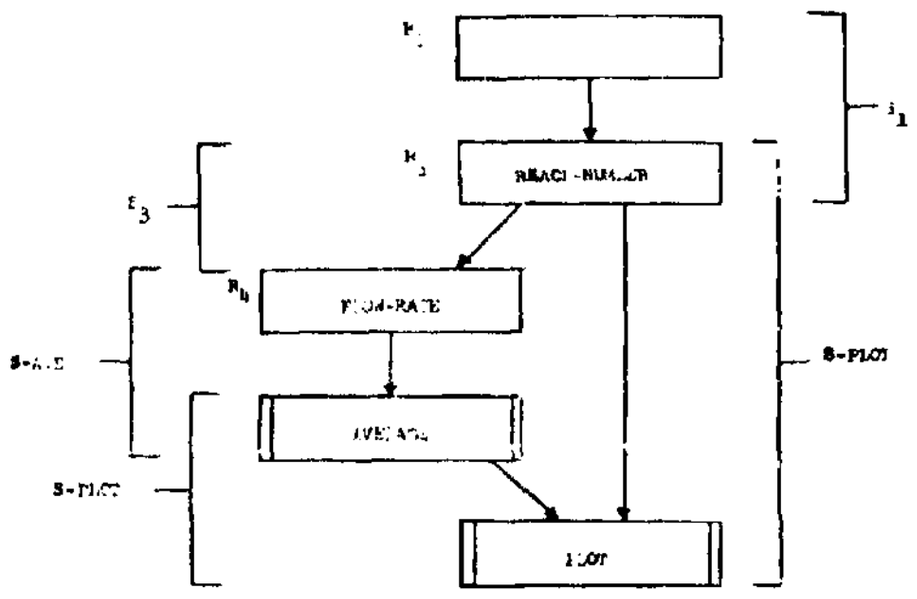



**Figure 3. Data Path with Program and Function**

along a single data path in order for the query to be valid.

For the following query:

PLOT THE REACH-NUMBER VERSUS THE AVERAGE (FLOW-RATE)

the goal would be to determine a data pth which contained REACH-NUMBER, FLOW-RATE, AVERAGE, PLOT. One path (not necessarily unique) which would satisfy this goal is shown in Figure 3. Since many models can produce specific item-types as outputs, the function names may be implied by the goal, rather than specifically stated. An example might be:

LIST LEAST-COST-SOLUTION

where this query would imply the execution of an optimization program whose output is a variable called "LEAST-COST-SOLUTION." As the technique is further developed, it is hoped that goals can be implied from ordinary English-like questions. It should be noted that this definition of goal is substantially different from the concepts used in many of the so-called "question and answering" systems. The goal, in our context, is to determine the data path which will satisfy the desired query. Once this data path is determined, a separate algorithm will actually go out and access the data and execute the functions along that path for the different record occurrences.

## Operators

The operators are used to transform the initial state to the final state, which satisfies the goal. These operators can be classified into two groups. The first group is used to connect and disconnect various set relationships in the data base to reach those item-types of interest. The second group relates to those functions which are stored in the data base and are used to operate on the data items. The format which will be used to describe the operators are as follows:

a) Name and Parameters.

b) Preconditions: This expression must be evaluated as being true for existing state before this operator can be applied.

c) Delete List: These wffs should be deleted from the current state of the system when this operator is applied.

d) Add List: These wffs should be added to the current state of the system when this operator is applied.

The following predicate will be referred by the operators:

a) **available (P) ≡ (∀ X) (∀ Y) (∃ Z)**
**[TYPE (P, ITEM) ∧ BELONG (P,X) ∧ CONNECT (Y,Z,X) ∧ STATUS (Y,ON)]**

The following function is used to determine the unique X for a given P.

b) **record (P) = X such that (∃ X) BELONG (P,X) is true**

The predicate available determines if item-type P is available; in other words, if item-type P currently lies along a path which is "ON". The record function determines which record-type X owns the item-type P. The following, then, are two operators which connect and disconnect the set structures;

1) **link (P): Link together two records to form a set**

**Precondition:** STATUS (P,OFF) ∧ (∃ W)(∃ X) (∃ Y)(∃ Z)
[CONNECT (P,X,Y) ∧ CONNECT (Z,W,X) ∧ STATUS (Z,ON) ∧ TYPE (Z,SET) ∧ [~(∀ ∧)(∀U)[CONNECT (V,W,U) ∧ STATUS (V,ON) ∧ TYPE (V,SET) ∧U ≠ X]]
**Delete List:** STATUS (P,OFF)
**Add List:** STATUS (P,ON)

2) **unlink (P): unlink two records from a set**

**Precondition:** STATUS (P,ON) ∧ (∃ X)(∃ Y) [CONNECT (P,X,Y) ∧ ~ (∀ W)(∀ Z) ⌈CONNECT (Z,Y,W) ∧ STATUS (Z,ON)]
**Delete List:** STATUS (P,ON)
**Add List:** STATUS (P,OFF)

The link operator is used to link together two records in a set or link together two records which form a relationship. Before a record can own a set or a relationship, it must Itself be owned by another record through a set relationship. Records which are owned through a relationship cannot in turn own other records. The unlink operator is used to break the set and relationship links. This operator will not break a link in the middle of the chain; in other words, it will only remove records from the bottom of the path.

The following operators are two examples of the kinds of functions which can be defined to operate on any data items:

3) ave (P): Average the value of P

Precondition: AVAILABLE (P)

Delete List: STATUS (S-AVE,OFF)

Add List: CONNECT (S-AVE,RECORD (P),R-AVE) STATUS (S-AVE, ON)

4) plot ($P_1$,$P_2$): PLOT $P_1$ VERSUS $P_2$

Precondition AVAILABLE ($P_1$ A

AVAILABLE: (P2)

Delete List: STATUS (S-PLOT,OFF)

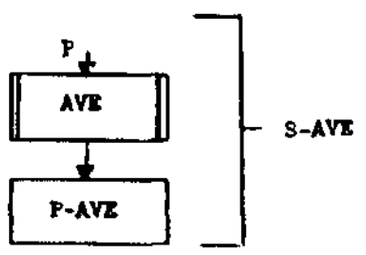
Add List: STATUS (S-PLOT,ON)

CONNECT (S-PLOT, RECORD ($P_1$),R-PLOT)

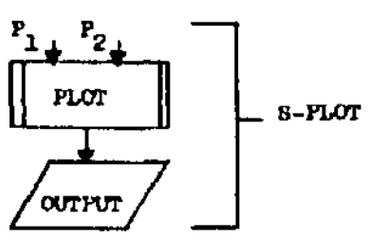
CONNECT (S-PLOT, RECORD ($P_2$),R_PLOT)

where the above functions assume the following information which was initial in the initial world description:

TYPE (S-AVE,FUNCTION)
STATUS (S-AVE,OFF)
TYPE (R-AVE, RECORD)
TYPE (P-AVE,ITEM)
BELONG (P-AVE,R-AVE)




TYPE (S-PLOT,FUNCTION)
STATUS (S-PLOT,OFF)




The third operator describes the average function which inputs one item-type and generates a new item-type called P-AVE. When this function is used, a new record is actually created in the data base. The fourth function is PLOT which inputs two Item-types and generates a plot as its output. Since the result of the PLOT function is only output, no new data structures are created by the PLOT function.

The two operators just discussed are general functions which are designed to operate on any data items. Another class of operators can be defined which requires specific data items rather than any data item. An example of this type might be:

5) Temp A Temperature Analysis Routine

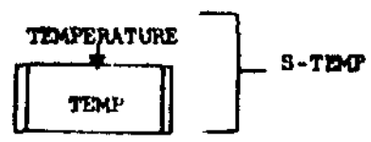
Preconditions: available (TEMPERATURE;

Delete List : STATUS (S-TEMP,OFF)

Add List: STATUS (S-TEMP,ON)

where the following would be included in the initial state:

TYPE (S-TEMP,FUNCTION)
STATUS (S-TEMP,OFF)




This function could be requested by the simple query:

TEMP

## Examples

The following examples will show how the previously defined operators may be used to determine the logical data path through the data and the programs stored in the data base. The first example is the query mentioned previously:

PLOT REACH-NUMBER VERSUS AVERAGE (FLCW-RATE).

One example of the sequence of operators which might be applied to the Initial world model to solve this problem would be as follows:

| Function | Add List | Delete List |
|---|---|---|
| link (S ) | STATUS (S ,0N) | STATUS ($S_1$ ,OFF) |
| link ($S_4$) | STATUS ($S_4$,ON) | STATUS ($S_4$,OFF) |
| unlink ($S_4$) | STATUS ($S_4$,OFF) | STATUS ($S_4$,ON) |
| link ($S_3$) | STATUS (S ,0N) | STATUS (S ,OFF) |
| ave (FLOW-RATE) | CONNECT (S-AVE,R4, R-AVE)<br>STATUS (S-AVE, ON) | STATUS (AVE,OFF) |
| PLOT (REACH-NUMBER,R-AVE) | CONNECT (S-PLOT,R , R-PLOT)<br>CONNECT (S-PLOT,R-AVE, R-PLOT)<br>STATUS (S-PLOT, ON) | STATUS (S-PLOT,OFF) |

The logical structure this query would generate is shown in Figure 3. Once this path was determined, the data would be accessed and the functions (AVE and PLOT) would operate on the collected data.

The second example would involve the following query:

PLOT FLCW-RATE AND TEMPERATURE

The query cannot be answered using the set relationships alone since these two item-types do not lie along the same data path. If the user were to define a relationship as given in the initial world model, then the logical data path for the query would be generate'd as follows:

| Function | Add List | Delete List |
|---|---|---|
| link ($S_1$) | STATUS (S .ON) | STATUS (S^OFF) |
| link ($S_3$) | STATUS ($S_3$,ON) | STATUS ($S_3$,OFF) |
| link (REL) | STATUS (REL,ON) | STATUS (REL.OFF) |

where the generated data structure is shown in Figure 4a.

The third example involves requesting *a* function whish will operate on a specific set of data. For example:

EXECUTE TEMP

The functions which would be required are:

| Function | Add List | Delete List |
|---|---|---|
| link ($S_1$) | STATUS ($S_1$ON) | STATUS ($S_1$OFF) |
| link ($S_4$) | STATUS ($S_4$0N) | STATUS ($S_4$OFF) |
| TEMP | STATUS (TEMP ,ON) | STATUS (TEMP, OFF) |

where the logical data structure for this query is shown in Figure 4b.

## Conclusion

The approach which was presented offers one possible solution to the complex problem of dynamically restructuring a data base in order to respond to a user's query. The problem solving technique provides the unique capability of determining if the goal (query) can be obtained given the initial world model. There still clearly remains several areas which need to be explored in more detail. One area involves trying to recognize "classes of queries" such that for simple requests, the traditional data path algorithm can be used, and for more involved requests, the problem solving approach should be used. Another area involves the possibility of saving queries and their paths for future reference, thus providing a learning capability to the system. The final area is concerned with placing more preconditions on the various functions to prevent such things as trying to average a group of names, and to perform other syntactical checks.

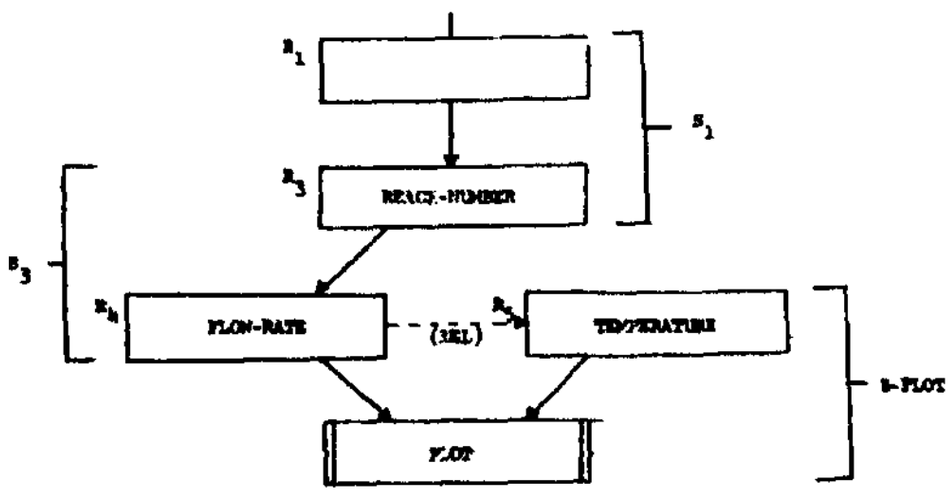



**Figure 4a. Path with Specific Function**

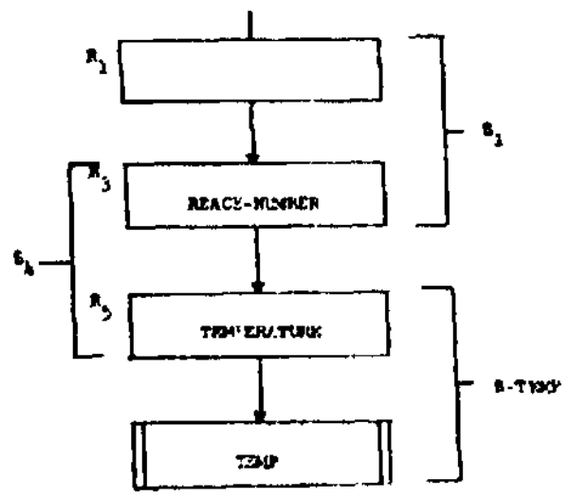



**Figure 4b. Path with Specific Function**

## Acknowledgement

This work was supported in part by Grant Number 65377-55 from the Office of Computing Activities of the National Science Foundation. The authors are Indebted to Professors Dave Workman and Bob Bonczek of Purdue University and Jack Buchanan of Carnegie-Mellon University.

## Bibliography

1. J. F. Nunamaker, D. Swenson, and A. B. Whinston, "Specifications for the Development of a Generalized Data Base Planning System," Proceedings of the Natural Computer Conference, New York City, AFIPS Press, June 1973.

2. W. D. Haseman, A. Z. Lieberman, A. B. Whinston, Generalized Planning System/Query System (GPIAN/QS), Users Manual, Krannert School of Industrial Administration, November 1974.

3. W. D. Haseman and A. B. Whinston, "Natural Query Language for Hierarchical Data Structures," Krannert School of Industrial Administration, November 1974.

4. W. D. Haseman, A. Z. Lieberman, and A. B. Whinston, "Generalized Planning System/Data Management System (GPLAN/DMS). Users Manual." Krannert Graduate School of Industrial Administration, December 1973.

5. W. D. Haseman, J. F. Nunamaker, and A. B. Whinston, "A Fortran Implementation of the CODASYL Data Base Task Group Report," Proceedings of the Fifth Annual Pittsburgh Conference on Modeling and Simulation, April 1974.

6. CODASYL Committee, Data Base Task Group Report. Association for Computing Machinery, April 1971.

7. N. J. NIlsson, Problem Solving Methods in Artificial Intelligence. McGraw-Hill Book Company, New York, New York, 1971.

8. R. E. FIkes and N. J. NIlsson, "Strips: A New Approach to the Application of Theorem Proving to Problem Solving," Artiflcal Intelligence 2 (1971), 189-208.