CATS

A Software Aid for

Analog/hybrid Programming

by

Ratilal Raichand Shah

Submitted to the School for Graduate Studies
in partial fulfillment for the requirements of
the degree of Master of Applied Science.

Department of Electrical Engineering
Faculty of Science and Engineering
University of Ottawa
Ottawa, Ontario

August 1976

ACKNOWLEDGMENTS

I wish to extend my thanks to a number of people who have helped directly or indirectly to shape and guide this thesis.

In particular to Dr. L.G. Birta, as my supervisor, for his everpresent counseling, patience, invaluable suggestions and encouragement during the course of this project.

To Drs. J. Raymond and T.I. Oren for their helpful suggestions while this work was carried out.

To Larry Towns, Mike Hull and Richard Perron for their assistance in programming.

I am grateful to Mrs. Claudette Henderson for her secretarial assistance in typing and helping to organize this document. Her full co-operation and care are especially acknowledged.

The financial support of the National Research Council through grant A8109 is gratefully acknowledged. Without it this work would not have been undertaken.

I wish to express my appreciation to the entire faculty and staff of the Department of Computer Science, Department of Electrical Engineering and the Computer Centre, University of Ottawa, for their kindness and moral support.

Finally I must acknowledge that without the loving support of my family I could not have done this work.

Table of Contents

List of Figures

List of Tables

ABSTRACT

The need for programming aids in the preparation and verification of analog/hybrid computer programs has led to the development of various software packages in recent years. Each of these has its distinctive features and limitations. This thesis describes a new software system of this general type; i.e. a "hybrid compiler", called CATS (CSMP - Analog Translator Software).

The CATS program is a FORTRAN based system which is related to an earlier hybrid compiler called PATCH, inasmuch as both programs utilize the syntax of S/360 CSMP (Continuous System Modelling Program) as the means of defining the problem being considered. The CATS package, however, differs from PATCH in several significant respects. These differences include both implementational aspects and overall capabilities.

The thesis describes the design principles underlying the CATS software as well as various aspects of its implementation. Examples are given to illustrate the capabilities of the program. Various limitations imposed on the user are also outlined.

CHAPTER 1

Introduction

1.1 Preliminary Remarks

Despite the inherent advantages in speed and cost per solution of the analog/hybrid (a/h) computer, its use as a problem solving tool has not been wide-spread. This restricted use is principally a result of the complex and time consuming programming procedures required in using the machine.

To deal with this problem, various software packages have been developed in recent years to assist the a/h computer user in program preparation and verification; e.g. APACHE[1], PATCH[2], ACTRAN[3], APSE[4]. These packages have the common feature of generating a/h patching information from a problem specification expressed in some high level language. These packages differ in overall capabilities (inherent restrictions) and in the nature of the source language.

A new software package of this general type is described in this thesis. This package is called CATS (CSMP - Analog Translator Software) where the chosen acronym reflects the fact that the package utilizes the S/360 CSMP (Continuous System Modelling Program [5]) language syntax as its source language.

Implemented around 1966, the APACHE system represents the first major effort in this general area (often referred to as hybrid compilation). One of its particular limitations relates to its inability to handle the parallel logic capabilities available on modern a/h computers.

PATCH is a more recently developed software package and shares the feature of using CSMP as its high level source language. The operation of PATCH however, relies heavily on the execution of the CSMP system and this in itself, represents a major drawback.

The choice of CSMP as a source language for problem specification for the CATS system was based on the following considerations:

1) Because of its wide range of system macros and functions, the CSMP syntax provides an especially powerful vehicle for describing the dynamics of continuous system.
2) CSMP is widely used and easy to learn and understand.
3) Many of the system macros provided in CSMP have natural hardware counterparts in a modern a/h computer.

In addition, by adhering strictly to the CSMP syntax rules, one has the possibility of having the source program executed by the CSMP processor. The resultant output can prove useful in verifying the correctness of the mathematical model or as check solutions for later a/h results.

The main output generated by CATS consists of a table containing all information relevant to preparing a magnitude and time scaled a/h program for the problem described in the source CSMP statements. In particular, the following are provided: maximum values of all problem variables, interconnections amongst the devices generating these variables and all potentiometer coefficients.

Of the several available high level languages, the FORTRAN language was selected for implementing the CATS program. This

choice was based primarily on the wide usage and relative simplicity of FORTRAN.

1.2 Relationship to PATCH

Because of their common feature of using CSMP syntax as the source language, the CATS and PATCH programs are closely related. However, a fundamental implementational difference exists inasmuch as CATS operates independently of the CSMP processor. This is not the case with PATCH whose operation can be viewed as an extension to the CSMP processor. This feature imposes several restrictions on the way in which the CSMP code for describing the problem can be written. Some of these restrictions are outlined below:

1) All outputs of amplifiers, multipliers, limiters, etc. must be explicitly named on the left side of structure statements. Thus the structure statement

```
W = X*Y*Z
```

is invalid. The necessary specification is

```
V = X*Y
W = V*Z
```

2) All constants within structure statements must be defined in a data statement. Thus rather than writing

```
X = 3.*Y+4.*Z
```

the user is obliged to write

```
CONST     THREE=3, FOUR=4
      X = THREE*Y+FOUR*Z
```

3) An arithmetic expression is not allowed in the argument of the INTGRL statement. Thus the statement

```
X = INTGRL(TWO,X+Y)
```

must be written as:

```
Z = X+Y
X = INTGRL(TWO,Z)
```

The CATS program is free of all the above restrictions.

The handling of complex structure statements; e.g. X = Y+LIMIT(A,B*C,M+N)/Z represents a major phase in the CATS processing. This phase involves decomposing the statement into a series of basic substatements each of which is associated with a unique a/h device. This procedure gives rise to new (system generated) variables whose names have the form Sdd where d denotes a decimal digit. The user is therefore obliged to avoid using such names in his own source code.

In its present form CATS is unable to handle a totally arbitrary CSMP source program. A summary of current restrictions is given in Appendix 3.

1.3 Summary of the Contents of the Thesis

The main data structures and general implementational aspects of the CATS program are described in detail in Chapter 2. Definitions of central concepts are also introduced. This chapter therefore provides an overview of the general organization of CATS and insight into its overall operation.

Chapter 3 describes how CATS processes the CSMP program which

specifies the simulation problem being studied. The chapter outlines how the program determines and stores all the necessary data contained in the source statements. Coding conventions for storing this data are also described.

Chapter 4 deals with the problem of decomposing complex structure statements referred to in the previous section.

An essential task of CATS consists of computing the maximum values of the variables associated with the problem. This is done by executing a FORTRAN subroutine created by the CATS program. The statements in this subroutine are derived from the source CSMP program. For proper execution of this subroutine its statements must be properly ordered and this requires a sorting operation. This operation is discussed in Chapter 5.

Chapter 6 discusses the generation of patching and scaling information for an a/h computer program. In particular, the procedure for computing the maximum values of problem variables via a CATS-generated subroutine is described. The formulation of the a/h wiring information must take into account the sign inversion associated with many of the devices on the a/h computer. At the same time, some attempt must be made to avoid excessive and redundant use of inverters in the final patching specifications. These matters are also dealt with in this chapter.

Chapter 7 demonstrates the capability of CATS by presenting several examples.

Finally, in Appendix 1 further details of the organization and implementation of CATS are provided, together with the JCL needed to execute a CATS job. In Appendix 2 a summary of the

subroutines in the CATS program and their interdependencies are given. Current restrictions in the use of CATS are outlined in Appendix 3.

CHAPTER 2

Data Structures and the Organization of CATS

2.1 Introduction

This chapter describes the main data structures that are used in the implementation of the CATS program. This description therefore provides an overview of the general organization of the program and insight into its overall operation. It should be noted that the storage allocation to the various data structures (i.e. their size) is user specificable and is normally tailored to the size of the source CSMP program being treated. This is achieved through the specification of certain size parameters as outlined in Appendix 1.

The data structures are grouped into three broad categories as follows:

(i) data structures for storing CSMP structure statements; i.e. the statements describing the mathematical model under consideration;

(ii) data structures for storing identifiers and their corresponding values;

(iii) data structures required for the generation of the patching data for the a/h program.

Each of these categories is dealt with in turn in the sequel.

2.2 Data Structures for CSMP Structure Statements

The SC Vector

All structure statements of the source CSMP program except those contained in a NOSORT segment or within user-defined MACRO's and/or PROCEDURES, are stored in the SC vector. Initially each such card image is stored directly as a character string (however blanks are deleted). At a subsequent stage each structure statement is translated into a numeric code sequence which replaces the initial character string description. Prior to its storage in the SC vector, a header is attached to each structure statement. This header contains four 'tags' which describe certain attributes of the structure statement. A structure statement, together with its header, is referred to in the sequel as an 'augmented structure statement'. Each of these tags is described below.

P-tag

The P-tag is a pointer which gives the location in SC of the beginning of the augmented structure statement which precedes the one in question. In the case of the first augmented structure statement, the P-tag is set to zero.

S-tag

The S-tag is a pointer which gives the location in SC of the beginning of the augmented structure statement which succeeds the one in question. In the case of the last augmented structure statement, the S-tag is set to zero.

T-tag

The T-tag describes the type of the structure statement in question. The T-tag for structure statements within the Initial segment, is assigned the value 0. The T-tag for structure statements within the Dynamic segment, is initially assigned the value 1. Later, when these statements are decomposed into their 'component parts', a more descriptive T-tag coding is used, as summarized in Table 4.5.

L-tag

The L-tag contains the length of the structure statement in question. Initially when each structure statement is stored as a sequence of non-blank characters, as contained on the source card, the L-tag is assigned the number of characters which make up the structure statement. After the coding operation which converts the character string into a numerical code sequence, the L-tag is reassigned the number of such codes.

2.3 Data Structures for Identifiers

All the identifiers on the left-hand side of data statements and structure statements are stored, together with their values.

The PNAME Vector

An identifier on the left-hand side of the '=' sign in a data statement and/or within the Initial segment is referred to as a 'pname'; these pnames are stored in a vector called PNAME. Pnames

are placed into PNAME through a hashing technique.

The VNAME Vector

An identifier on the left-hand side of the '=' sign which is not a pname is referred to as a 'vname', these vnames are stored in a vector called VNAME. Vnames are also placed into VNAME through a hashing technique.

The PVAL Vector

The PVAL vector is a parallel vector to PNAME, which stores the values of pnames in corresponding positions; i.e. if a pname is stored in PNAME(i) then PVAL(i) will be the value of this pname.

The VMAX Vector

The VMAX vector is a parallel vector to VNAME, which stores the maximum values of vnames in corresponding positions, i.e. if a vname is stored in VNAME(i), then VMAX(i) will be the maximum values of this vname.

2.4 Data Structures for Patching

The following data structures build up the data that will subsequently contain the necessary information for the a/h program.

The LABEL Vector

The sorting operation (see Chapter 5) establishes a 'natural ordering' for the structure statements in the Dynamic segment, and

consequently it is possible to speak of the J^{th} statement in the sequence. The vname appearing on the left-hand side of the J^{th} statement necessarily appears somewhere in the VNAME vector. LABEL(J) contains a pointer to the location in VNAME which stores the vname on the left-hand side of the J^{th} structure statement; i.e. the vname which represents the 'output' of the J^{th} structure statement (or the J^{th} 'device').

The NODTYP Vector

The NODTYP vector is a parallel vector to the VNAME vector, and its entries indicate the type of the device which generates the corresponding vname. Thus, if the output of the J^{th} structure statement is V, whose location in VNAME is K_V, then

$$LABEL(J)=K_V$$

$$NODTYP(K_V)=T$$

where T is a numeric code designating the device type associated with the output variable (vname) V. In general, T will be identical to the contents of the T-tag of the header associated with the J^{th} structure statement.

The NODIN Vector

The vnames appearing on the right-hand side of each structure statement in the Dynamic segment are viewed as 'inputs' to the device represented by that structure statement. The pnames appearing on the right-hand side of these statements usually appear as constants which either multiply or divide a vname. In some cases, however, a pname may appear independently. Such a circumstance is

handled by adopting the point of view that such a pname is in fact a multiplicative constant associated with a pseudo-vname. This pseudo-vname is called either PREF if the algebraic sign preceding the pname is (+), or NREF if the algebraic sign preceding the pname is (-). The value of both these pseudo-vnames is 1. With this convention, an 'independently' appearing pname on the right-hand side of a structure statement is viewed a legitimate 'input' to the device in question.

The NODIN vector is a parallel vector to VNAME and its entries contain the number of inputs to the device which generates the associated vname. Continuing with the notation introduced above, if

$$NODIN(K_V)=N$$

then the device generating the vname stored in $VNAME(K_V)$ has N inputs.

The NODPTR Vector

The NODPTR vector is also a parallel vector to VNAME and its entries contain pointers to locations in the NODSRC vector. As described below the two vectors NODPTR and NODSRC together with NODIN and LABEL provide the mechanism for identifying the vnames which serve as inputs to any particular device.

The NODSRC Vector

The NODSRC vector normally contains pointers to locations in the LABEL vector and is used to determine the vnames which serve as inputs to a particular device. To illustrate, suppose the vname

V is stored in location K_V of VNAME and suppose

$$NODIN(K_V)=N$$

$$NODPTR(K_V)=M$$

$$NODSRC(M)=R_1$$

$$NODSRC(M+1)=R_2$$

$$\vdots$$

$$NODSRC(M+N-1)=R_N$$

then the vnames which are required in the generation of V are those referenced by $LABEL(R_1)$, $LABEL(R_2)$ ... $LABEL(R_N)$. It may however occur that one of the R_K is 0 or -1. This circumstance implies an input from PREF or NREF respectively.

The MROW Vector

The MROW vector is a parallel vector to the LABEL and designates the polarity of the associated vnames. The entries are either 0 or 1 indicating respectively that the output polarity is either positive or negative.

The BRGAIN Vector

The BRGAIN vector is a parallel vector to the NODSRC vector, and its entries assign a gain to each input. Each such gain is a pname and consequently the entries in BRGAIN are pointers to locations in the PNAME vector. Continuing with the example introduced above, suppose

$$BRGAIN(M)=S_1$$

$$BRGAIN(M+1)=S_2$$

$$\vdots$$

$$BRGAIN(M+N-1)=S_N$$

This would imply that the gains associated with the inputs are in locations S_1, S_2, ... S_N of the PNAME vector. As noted earlier, the parallel vector, PVAL, contains the numeric values of these pnames.

The MDFLAG Vector

The MDFLAG vector, is a parallel vector to BRGAIN, which establishes whether the gain associated with an input is in a "multiply mode" or a "divisor mode". The entries in MDFLAG are either 1 or 0 indicating respectively that the gain occurs in a multiply or divisor mode.

The GAIN Vector

The GAIN vector is a parallel vector to the NODSRC vector. The magnitude of each entry is the net input gain required for use with the corresponding device input variable. Initially the algebraic signs of these entries correspond to the algebraic signs preceding the input variables in the dynamic structure statements. At a later stage when the proper polarities of all the device outputs are determined (see Chapter 6), necessary alterations on GAIN will result in all entries having positive signs.

Continuing with the example introduced above, suppose the initial entries in GAIN are as follows:

$$GAIN(M)=c_1$$

$$GAIN(M+1)=-c_2$$

$$GAIN(M+2)=c_3 \qquad (c_i>0)$$

$$\vdots$$

$$GAIN(M+N-1)=-c_N$$

This would imply that the inputs whose names are referenced by LABEL(R_1), LABEL(R_2), ... LABEL(R_N), enter the device in question through gains whose magnitudes are c_1, c_2, ..., c_N. Furthermore, the example indicates that the vnames referenced by LABEL(R_2) and LABEL(R_N) are preceded by a negative sign in the source statement in question.

The NODOUT Vector

The NODOUT is a parallel vector to the VNAME vector and stores the number of devices that require the output of the device in question; e.g. if

NODOUT(K_V)=L

then the implication is that the vname V serves as an input to L devices.

The PTRGO Vector

The PTRGO vector is a parallel vector to NODOUT and its entries are pointers to locations in the NODGO vector (see below). As described below, the PTRGO and NODGO vectors together with NODOUT and LABEL provide the mechanism for determining which devices require a particular vname as an input.

The NODGO Vector

The NODGO vector contains pointers to locations in the LABEL vector and is used to determine the devices which require a particular vname as an input. To illustrate, suppose the vname V is stored in location K_V of VNAME and suppose

NODOUT(K_V)=L

PTRGO(K_V)=J

NODGO(J)=R_1

NODGO(J+1)=R_2

⋮

NODGO(J+L-1)=R_L

then the devices which require V as an input are those whose output labels are referenced by LABEL(R_1), LABEL(R_2), ..., LABEL(R_L).

Figure 2.1 shows some of the feature of these various data structures and their interactions.

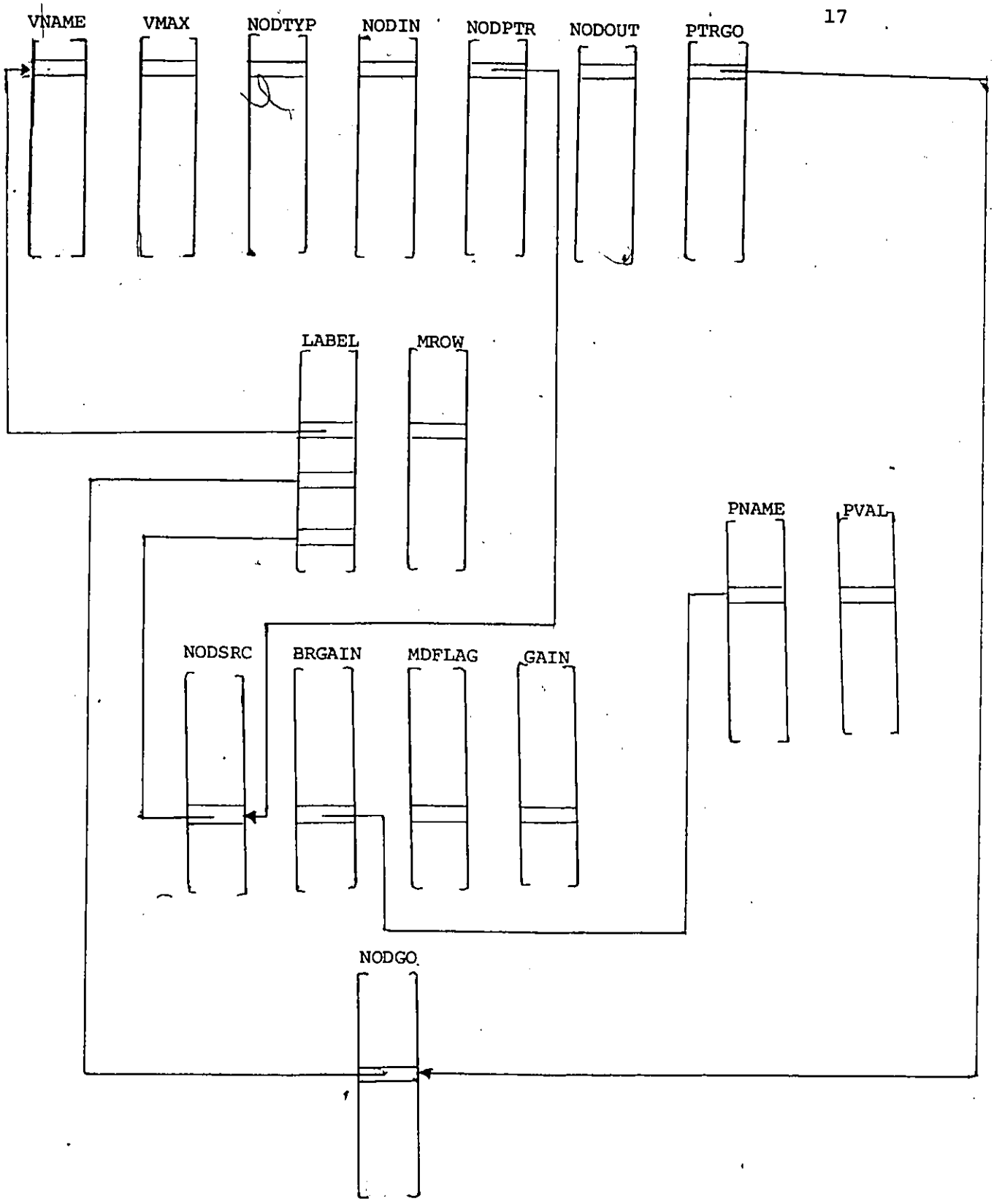


Figure 2.1

Interactions Amongst the Data Structures

CHAPTER 3

Initial Processing of the Source Statements

3.1 Introduction

The structure of the simulation problem and the data for the variables are defined by the structure and data statements of the CSMP program. All valid structure, control and data statements are processed by the CATS program. Some of these statements are however not relevant to the a/h translation process or are considered to be outside the scope of the present work. Such statements are therefore ignored in subsequent processing.

This chapter outlines how the CATS program determines and stores all the necessary data associated with the problem. The coding convention for storing the structure statements is also outlined.

3.2 Processing of the CSMP program data

During this initial phase of the program, the user's input which is a source CSMP program (consistent with all CSMP syntax rules) is read in as data. Each card is read and processed successively until the first 'END' card is encountered. Cards following the first 'END' card are not processed.

All structure statements relevant to defining the model are stored in the SC vector except for those statements which are (a) between the MACRO-ENDMAC labels, (b) between the PROCEDURE-ENDPRO lables and (c) in a NOSORT segment. The non-blank characters

in each structure statement are stored in the SC vector. When a statement is continued on a following card, the '...' are surpressed and the following card is concatenated with its predecessor. As each statement is stored in SC, appropriate values are assigned to the 4 tags attached to its head. At this stage structure statements from the Initial segment are designated as type 0 whereas the structure statements from the Dynamic segment are designated as type 1. These values are assigned to the augmented statement T-tag. The L-tag contains the total number of characters in the statement stored in SC. The P-tag points to the location in SC of the beginning of the preceding augmented statement and the S-tag points to the location in SC of the beginning of the succeeding augmented statement. For each structure statement the identifier on the left-hand side of '=' is stored in PNAME or VNAME vector depending on whether the structure statement is from the Initial or Dynamic segment. A hashing technique is used for storing these entries in PNAME and VNAME.

OVERLAY and TABLE data statements are discarded. Each identifier on the left-hand side of '=' on PARAM, CONST, INCON and FUNCTION cards is stored in PNAME. The corresponding value of the identifier is stored in the corresponding position in the parallel vector PVAL. A special case is a FUNCTION data statement. The name identifying the function is stored in PNAME. The values of the 'X' variable are stored in successive locations of a vector called Z and the values of the 'Y' variable are stored in corresponding positions of a parallel vector called F. The location

in PVAL corresponding to the function name is assigned a value of the form $\alpha.\beta$. The integral part of this number (namely α) is a pointer to the first location in Z associated with the function. The fractional part (namely β) is a pointer to the last location in Z.

The only translation control statement processed is RENAME. On encountering such a statement, the reserved identifiers 'TIME' and 'FINTIM' are altered in VNAME if these identifiers are assigned new names via this statement.

The TIMER statement is the only execution control statement processed. On encountering such a statement the value assigned to FINTIM (or its altered name if specified in a RENAME statement) is assigned as the value in VMAX(1).

All output control statements are discarded by CATS.

It should be noted that by convention the names TIME and FINTIM are assigned the first two locations of VNAME and are thus not subject to the hashing technique. Similarly the first three locations in PNAME are reserved for the system generated pnames S100, S101 and S102 whose values are stored in the first three locations of PVAL as 0.0, 1.0 and 2.0 respectively.

3.3 Coding the Structure Statement

After reading the input data, the SC vector contains all the relevant statements stored in their character form and PNAME and VNAME contain all the identifiers. Furthermore the values of the pnames are stored in corresponding locations in PVAL. Since the

storage of these statements in a character-form is wasteful of storage space, each statement is re-generated in SC as an equivalent numeric code string.

The coding process is based on the fact that each statement to be coded is composed of elements from three different categories of language entities; namely, (i) extended operators (the usual arithmetic operators of +, -, *, / and ** together with =,), (and,); (ii) function names (e.g. ABS, SQRT, INTGRL, LIMIT, etc.); and (iii) identifiers (namely pnames and vnames). It should be noted that each constant appearing in the source CSMP program is replaced by a system generated pname whose value is the constant. Table 3.1 provides a summary of the specific coding convention used for categories (i) and (ii) above.

The code used for pnames and vnames is a negative number related to their respective locations in the PNAME and VNAME vectors. Specifically, if PN is a pname stored in location K_p of PNAME, then all occurrences of PN are replaced by the number $-(K_p+1000)$. Similarly if VN is a vname stored in location K_V of VNAME then all occurrences of VN are replaced by the number $-K_V$.

This coding operation on the data stored in the SC vector necessitates a modification of the L-tag in the header of each statement. In particular, the L-tag is assigned a new value given by the total number of numeric codes in the coded statement. This change also requires corresponding changes in the P and S tags.

Two pointers, PTRFST and PTRD, are assigned values corresponding to the location in SC of the first Initial statement and

EXTENDED OPERATOR	CODE
=	104
(	105
)	106
,	111
+	112
-	113
*	121
/	122
** (↑)	131*

Table 3.1(a)

Coding Convention for Extended Operators

* In the sequel ↑ represents the exponentiation operator.

FUNCTION NAME	CODE
ABS	2
SQRT	4
SIN	5
COS	6
NLFGEN	7
AFGEN	7
NOT	8
TAN	14
EXP	15
ALOG	16
ALOG10	17
TANH	18
ATAN	19
IABS	20
COMPAR	23
AND	24
NAND	25
IOR	26
NOR	27
EOR	28
INSW	30
INTGRL	32
LIMIT	33
MODINT	34
REALPL	97
CMPXPL	98
LEDLAG	99

Table 3.1(b)

Coding Convention for Function Names

first Dynamic statement respectively.

A further task performed during this phase, is concerned with the relocation of improperly located Initial segment statements. During the program preparation, the user may inadvertently place within the Dynamic segment a statement of the form name = RHS, where the expression RHS does not contain any vname references. Such an assignment should more properly be relocated within the Initial segment of the program, since it is necessarily an Initialization statement. This relocation operation also involves deleting the identifier 'name', from the VNAME vector and inserting it into the PNAME vector. Related coding changes are also required in the SC vector.

CHAPTER 4

Decomposition of the Source Statements

4.1 Introduction

A source statement in the Dynamic segment may be 'too complex' to be directly realizable by a single a/h computing device. In such circumstances it is necessary to decompose the statement into an equivalent sequence of substatements, each of which has the desired property of corresponding to a single a/h computing device. The central concern of this chapter is with describing the method used in CATS for dealing with this problem.

To establish the foundation for this discussion, the following section provides the definitions of the three basic types of expressions which can be directly implemented by a single a/h computing device. Each of the substatements resulting from the decomposition procedure produces a structure having the form V=rhs where rhs is one of the three basic types of expressions defined in section 4.2.

Prior to applying the decomposition procedure, it is essential to transform each occurrence of the CSMP macros REALPL, LEDLAG and CMPXPL into an equivalent INTGRL form. These transformations are described in section 4.3.

The basic decomposition procedure is then outlined. It may, however, give rise to substatements containing an exponential operator (↑) having a negative exponent which is a circumstance that does not have a direct a/h computer counterpart. Thus a special step must be taken to transform such occurrences into an

alternate form that is directly realizable. This process is described in section 4.6.

An example is considered in some detail to illustrate the various stages of the decomposition procedure.

4.2 Definitions

In the following P, P_1 and P_2 denote pnames and V, V_1 and V_2 denote vnames.

Canonical expression

A canonical expression is an expression which has the form

$$\theta_0 \; S_1 \; \theta_1 \; S_2 \; \theta_2 \; \cdots\cdots \; \theta_{n-1} \; S_n$$

where:

θ_0 is either a null operator or a unary + or -

θ_i is either + or - (binary)

S_i is a "basic expression"; i.e. S_i has one of the following forms:

(i) P

(ii) V

(iii) P*V or V*P

(iv) V/P

Non-linear expression

A non-linear expression is an expression which has one of the following forms:

(i) $P*V_1\theta V_2$

(ii) $V_1*P\theta V_2$

(iii) P/V (v) $V_1 * V_2$

(iv) V↑2 or V↑0.5 (vi) V_1 / V_2

Where θ is one of the binary operators * or /.

Functional expression

A functional expression is an expression which has one of the following forms:

(i) INTGRL(P,Ce)

(ii) MODINT(P,{P,V},{P,V},Ce)

(iii) LIMIT(P_1,P_2,Ce)

(iv) COMPAR({P,V},{P,V})

(v) INSW({P,V},{P,V},{P,V})

(vi) NLFGEN(name,V)

(vii) SQRT(V)

(viii) SIN(V)

(ix) COS(V)

(x) AND({P,V},{P,V})

(xi) NAND({P,V},{P,V})

(xii) IOR({P,V},{P,V})

(xiii) EOR({P,V},{P,V})

(xiv) NOR({P,V},{P,V})

(xv) NOT(V)

In the above Ce denotes a canonical expression and {P,V} indicates a choice of the argument between a pname and a vname.

In general, the creation of the substatements from a source statement leads to the generation of system pnames and vnames of the form Sldd and Sdd respectively, where d is a decimal digit.

Thus identifiers with these formats should not appear in the source CSMP program.

4.3 Transformation of the CSMP Macros

Before the process of decomposition of the source statements can be undertaken, it is necessary to transform all the occurrences of the CSMP macros REALPL, LEDLAG and CMPXPL into an equivalent INTGRL form, which utilizes only the INTGRL macro. The nature of these transformations is summarized in Table 4.1. In this Table, Sdd represents a system vname and the argument X may be either a pname, a vname or an expression.

Note that as each such macro-statement is transformed, it is deleted from the SC vector and its corresponding INTGRL form is inserted into the first available space in SC. Recall that the statements are stored in SC in a numerically coded form.

4.4 The Decomposition Procedure

Each source statement from the Dynamic segment is passed through the decomposition procedure whose purpose is to decompose where necessary, the statement into an equivalent sequence of substatements each having 'simple' structure and each having the property of being implementable by a single a/h computing device. The decomposition procedure is based on a procedure described by Knuth [6].

In its preliminary phase, this procedure gives rise to the generation of temporary labels of the form Ti where i is a decimal

CSMP Macro-statement	Equivalent 'INTGRL' Statement(s)
Y=REALPL(IC,P,X)	Y=INTGRL(IC,(X-Y)/P)
Y=LEDLAG(P_1,P_2,X)	Sdd=INTGRL(0,(Sdd-X)/P_2) Y=-(P_1*X+(P_1-P_2)*Sdd)/P_2
Y=CMPXPL(IC_1,IC_2,P_1,P_2,X)	Sdd=INTGRL(IC_2,X-2*P_1*P_2*Sdd-P_2**2*Y) Y=INTGRL(IC_1,Sdd)

Table 4.1

Transformations of the CSMP Macros

digit. These temporary labels may subsequently be replaced by system generated vnames or pnames (as appropriate) or alternatively they may cease to have relevance due to a substitution process.

In the procedure each statement is viewed as consisting of elementary components or 'items'. These items are either operands or operators. The latter are divided into three categories as summarized in Table 4.2. Furthermore the operators have an assigned priority as summarized in Table 4.3. In Table 4.3, the special operators @ and ; are distinctive inasmuch as they are inserted during the course of the analysis to indicate the left and right boundaries respectively of the source statement being examined.

The input to the procedure consists of a source statement from the Dynamic segment which is scanned from left to right. Let S be the item currently being analyzed, then all the items to the left of S are considered to be the contents of a stack where the first item to the left of S is the top element of the stack e.g.

STACK	S	INPUT
@X=Y+LIMIT(A,B	*	C,M+N)/Z;

A flowchart of the procedure is given in Figure 4.1 and a brief description follows:

Start by setting the counter I to 1 (I counts the number of substatements generated) and set S to @, the first item in the input string (Box 1).

Test the item S (Box 2). If S is a binary operator, a right parenthesis or a semicolon and if the priority of the second item from the top of stack (an operator) is greater than the priority of S (Box 3), then some action is initiated. Otherwise the item

Operator Category	Operator Members
binary	+ - * / ↑ ,
unary	unary - and function name e.g. INTGRL, COS ... etc.
special	( ) = @ ;

Table 4.2

Categories of Operators

Priority	Operators
0	@ = ( ) ;
1	+ - ,
2	* /
3	↑
4	unary

Table 4.3

Priorities of Operators

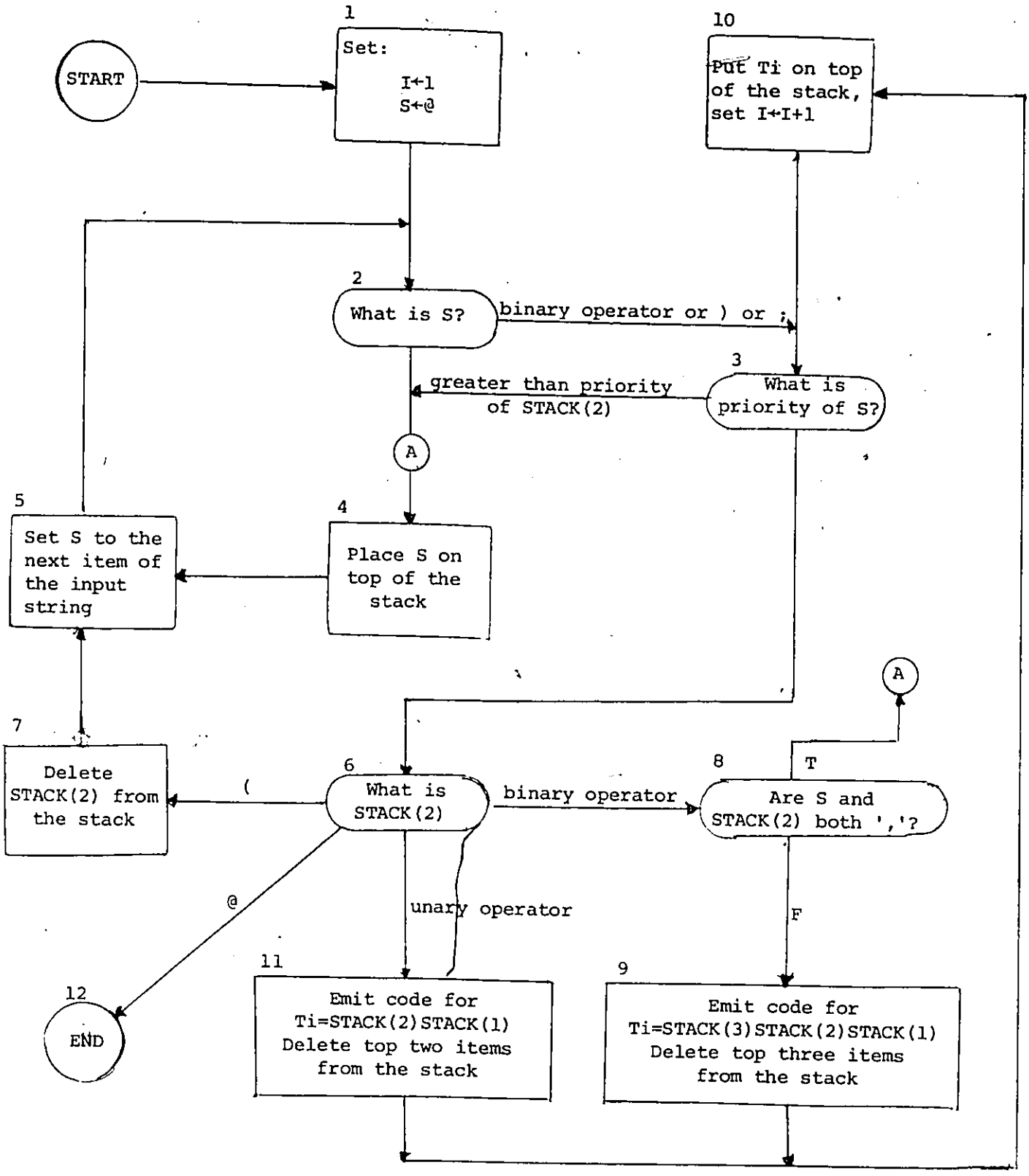


Figure 4.1

Flowchart of the Decomposition Procedure

S is placed on the top of the stack (Box 4) and the next item to the right of S is examined (Box 5).

Depending on the operator that is the second item from the top of the stack (Box 6), one of the following actions is initiated:

If the second item from the top of the stack is a left parenthesis then it is deleted from the middle of the stack (Box 7) and the next item of the input string is examined (Box 5). Note that S is deleted and not placed on the top of the stack.

If the second item from the top of the stack is a binary operator θ then the top three items of the stack are necessarily of the form:

$$X\theta Y$$

and thus a substatement of the form:

$$Ti=X\theta Y$$

is generated (i is an integer corresponding to the counter I). The top three items of the stack are deleted (Box 9). The temporary label Ti is placed on the top of the stack as an operand and I is incremented by 1 (Box 10) and a priority test is made (Box 3). Note that S has not been altered. However one exception to the generation of substatement with a binary operator is when both the items in S and the second item of the stack are the binary operators ','. In such a case S is saved on the top of the stack and the next item is examined (Box 8).

If the second item from the top of the stack is a unary operator θ, then the top two items of the stack are necessarily of the form

$$\theta X$$

and a substatement of the form

Ti=θX

is generated. The top two items of the stack are deleted (Box 11). The temporary label Ti is placed on top of the stack as an operand and I is incremented by 1 (Box 10) and a priority test is made (Box 3). Again S is not altered.

If the second item from the top of the stack is the operator @, then the entire input string has been processed and the procedure ends (Box 12).

An example illustrating the action of this algorithm on the input string X=Y+LIMIT(A,B*C,M+N)/Z is given in Figure 4.2.

4.5 Creation of POST Matrix

The substatements generated in the decomposition procedure are sequentially stored (in a numeric code) in the rows of a matrix called POST. Each row of the POST represents a substatement with the following format:

Column 1: contains the temporary label Ti, where i corresponds to the row number.

Column 2: contains the operator of the substatement.

Column 3: contains the left operand of the substatement.

Column 4: contains the right operand of the substatement.

Whenever the operator is unary, then the 3rd column is assigned the value of 0 to denote the absence of the second operand. Furthermore in the first column of the last row of POST, the temporary label Ti is replaced by the vname that existed on the left-

STACK	S	INPUT	
	@	X=Y+LIMIT(A,B*C,M+N)/Z;	
@	X	=Y+LIMIT(A,B*C,M+N)/Z;	
@X	=	Y+LIMIT(A,B*C,M+N)/Z;	
@X=	Y	+LIMIT(A,B*C,M+N)/Z;	
@X=Y	+	LIMIT(A,B*C,M+N)/Z;	
@X=Y+	LIMIT	(A,B*C,M+N)/Z;	
@X=Y+LIMIT	(	A,B*C,M+N)/Z;	
@X=Y+LIMIT(	A	,B*C,M+N)/Z;	
@X=Y+LIMIT(A	,	B*C,M+N)/Z;	
@X=Y+LIMIT(A,	B	*C,M+N)/Z;	
@X=Y+LIMIT(A,B	*	C,M+N)/Z;	
@X=Y+LIMIT(A,B*	C	,M+N)/Z;	
@X=Y+LIMIT(A,B*C	,	M+N)/Z;	T1=B*C
@X=Y+LIMIT(A,T1	,	M+N)/Z;	
@X=Y+LIMIT(A,T1,	M	+N)/Z;	
@X=Y+LIMIT(A,T1,M	+	N)/Z;	
@X=Y+LIMIT(A,T1,M	N	)/Z;	
@X=Y+LIMIT(A,T1,M N	)	/Z;	T2=M+N
@X=Y+LIMIT(A,T1,T2	)	/Z;	T3=T1,T2
@X=Y+LIMIT(A,T3	)	/Z;	T4=A,T3
@X=Y+LIMIT(T4	)	/Z;	
@X=Y+LIMITT4	/	Z;	T5=LIMIT T4
@X=Y+T5	/	Z;	
@X=Y+T5/	Z	;	
@X=Y+T5/Z	;		T6=T5/Z
@X=Y+T6	;		T7=Y+T6
@X=T7	;		T8=X=T7
@T8	;		

Figure 4.2

Snapshots of the Decomposition Algorithm Applied to X=Y+LIMIT(A,B*C,M+N)/Z

hand side of the original statement.

As an example, the POST matrix after processing the source statement

X=Y+LIMIT(A,B*C,M+N)/Z

is shown in Figure 4.3.

4.6 Processing an Expression with an Exponential Operator having Negative Exponent

The expression V↑(-P) (where P is a pname with a value of either 2.0 or 0.5 and V is a vname) is not directly implementable by a single a/h computing device. In order to accommodate this expression, its occurrences must be manipulated to produce an equivalent expression involving the implementable form V↑P.

The equivalent form produced for the structure V↑(-P) depends on the nature of the expression in which this structure occurs specifically on the operators which preceed and follow the structure V↑(-P) in the source expression. For example, if the structure occurs as V↑(-P)*Vl, then an equivalent acceptable representation is Vl/V↑P.

Recall that each generated substatement occupies a unique row of the POST matrix. Thus an expression containing the exponential of the type being considered will be represented by two rows of the POST matrix. For example, the structure .....V↑(-P)..... will give rise to the following entries in the POST matrix:

T1	*	B	C
T2	+	M	N
T3	,	T1	T2
T4	,	A	T3
T5	LIMIT	T4	0
T6	/	T5	Z
X	+	Y	T6

Figure 4.3

The POST Matrix Resulting from

the Source Statement

X=Y+LIMIT(A,B*C,M+N)/Z

$$\begin{bmatrix} & & \vdots & \\ Ti & - & P & 0 \\ Tj & \uparrow & V & Ti \\ & & \vdots & \end{bmatrix}$$

where Ti and Tj are temporary labels and j=i+1. The manipulations in question are performed directly on the entries of the POST matrix as created during the decomposition procedure. Eight possible cases can occur and in Table 4.4 the transformations which take place in each case are summarized. In this Table, θ represents a null operator, θ_1 represents one of the binary operators + or - and θ_2 represents one of the binary operators *, + or -. Ti, Tj and Tk temporary labels assigned to the subexpressions in the i^{th}, j^{th} and k^{th} rows of the POST matrix respectively. The second and fourth columns of this Table corresponds to the entries within the POST matrix.

4.7 Substatement Creation from POST Matrix

At this stage, most rows of the POST matrix represent substatements that can be implemented by a single a/h computing device. One of the exceptional cases is when the subexpression (i.e. the last three columns of a row of the POST matrix) contains the binary operator ','. In this situation the subexpression is in reality, a portion of a function argument list. Hence such a row of POST matrix is not a valid operation. Furthermore the label for this subexpression; e.g. Ti (as contained in column 1) must necessarily be referenced as an operand in some other row of the POST matrix. Thus this situation can be handled simply by replacing the reference

	Portion of the source expression	Corresponding Generated subexpression	Equivalent form of the expression	Corresponding Generated subexpression
(i)	$\theta_0 V\dagger(-P)\theta_0$	-P V†Ti	1/V†P	V†P 1/Ti
(ii)	...V†(-P)*V1...	-P V†Ti Tj*V1	V1/V†P	V†P V1/Tj
(iii)	...V†(-P)/V1...	-P V†Ti Tj/V1	1/(V†P*V1)	V†P Ti*V1 1/Tj
(iv)	...V†(-P)θ_1V1...	-P V†Ti Tjθ_1V1	(1/V†P)θ_1V1	V†P 1/Ti Tjθ_1V1
(v)	...V1*V†(-P)/V2...	-P V†Ti V1*Tj Tk/V2	V1/(V†P*V2)	V†P Tj*V2 V1/Tk
(vi)	...V1*V†(-P)θ_2V2...	-P V†Ti V1*Tj Tkθ_2V2	(V1/V†P)θ_2V2	V†P V1/Tj Tkθ_2V2
(vii)	...V1/V†(-P)...	-P V†Ti V1/Tj	V1*V†P	V†P V1*Tj
(viii)	...V1θ_1V†(-P)...	-P V†Ti V1θ_1Tj	V1θ_1(1/V†P)	V†P 1/Ti V1θ_1Tj

Table 4.4

Transformations Used in Handling V†(-P)

to Ti with the subexpression containing the operator ','.

There is another circumstance in which a row of the POST matrix should not give rise to a separate statement. This occurs when the row corresponds to a term in a canonical expression (see section 4.2) and therefore corresponds to one of the inputs to a multi-input a/h device e.g. a summer, an integrator etc. For example, consider a source statement of the form:

X=X1+X2-X3+X4

The decomposition procedure would generate the following equivalent sequence of substatements:

T1=X1+X2

T2=T1-X3

Y=T2+X4

This set of substatements would give rise to 3 summers, each with 2 inputs, which is clearly an undesirable result. The proper result should be a single summer with 4 inputs..

Bearing in mind the above two issues, the following problem must be considered: Under what circumstances does a row of the POST matrix give rise to a new statement having a system name (rather than a temporary label) on its left hand side. Listed below are the rules for dealing with the problem; specifically they are the rules governing the conditions under which a subexpression (i.e. a row of POST) is given a system name and thus becomes a new statement:

(i) when the subexpression has a unary operator

(ii) when the subexpression has the operator †

(iii) when the subexpression has the operator * and both its operands are either pnames or vnames

(iv) when the subexpression has the operator / and the second operand is a vname

(v) when the subexpression has one of the binary operators + or - and (a) the subexpression is referenced by another subexpression having either a unary operator or one of the operators +, * or /, or (b) both the operands of the subexpression are pnames

(vi) when the subexpression has the operator ',' and (a) its first operand refers to a subexpression having one of the binary operators + or -, or (b) its second operand refers to a subexpression having one of the binary operators + or - and is not a portion of an argument list of one of the functions LIMIT, MODINT or INTGRL. (This is because the above functions can have a canonical expression as their last argument).

Any subexpression that does not fall under one of the above rules does not give rise to a new statement and hence the temporary label, Ti, in column 1 is not replaced by a system name. Instead the whole subexpression replaces the occurrence of Ti, as an operand, in some other row of POST. In this context it should be recalled that:

(i) each temporary label, Ti, is referenced once and only once as an operand of some other subexpression (i.e. row of POST)

(ii) each temporary label is always referenced by a subexpression lying beneath it in the POST matrix

(iii) each subexpression containing the temporary label Ti as an operand is linked to its associated subexpression through the subscript, i, which serves as a row pointer

The rows of the POST matrix are processed sequentially beginning with the first row. Whenever a temporary label is to be replaced by a system name, this replacement name is a vname if one of the operands is a vname; otherwise it is a pname. Whenever a replacement takes place, corresponding change must be made at the other occurrence of the temporary label.

Continuing with the example introduced in section 4.4 and applying the rules given above, the modified POST matrix becomes:

S00	*	B	C
T2	+	M	N
T3	,	S00	T2
T4	,	A	T3
S01	LIMIT	T4	0
S02	/	S01	Z
X	+	Y	S02

where it is assumed all the names are vnames.

The final stage in the substatement creation procedure proceeds by working upwards from the bottom of the POST matrix. Whenever a row is encountered with a vname or a pname in column 1 and with no temporary label as an operand, then a new statement can be directly generated. Alternatively whenever a temporary label occurs as an operand then a substitution process is used to eliminate the reference by working upwards in the POST matrix.

The occurrence of the ABS function requires special attention because its a/h realization requires three distinct components. Accordingly, to properly reflect this requirement, the occurrence of Y=ABS(X) in a row of POST, gives rise to the following three statements:

```
Sdd=COMPAR(X,0)
Sdd'=INSW(Sdd,0,X)
Y=2*Sdd'-X
```

Applying the procedure to the example given in (section 4.4) the following four substatements are created:

```
X=Y+S02                 (1)
S02=S01/Z               (2)
S01=LIMIT(A,T3)
   =LIMIT(A,S00,T2)     } substitution process
   =LIMIT(A,S00,M+N)    (3)
S00=B*C                 (4)
```

Each newly created substatement is stored at the first available space in the SC vector. The T-tag associated with the substatement represents the type of a/h device required for its realization as given in Table 4.5. The T-tags associated with the above four statements are 31, 22, 33 and 21 (a summer, a divider, a limiter and a multiplier respectively). The original source statement is deleted from the SC vector.

Type of A/H Device	T-tag
Squarer	3
Square Rooter	4
Sine Generator	5
Cosine Generator	6
Function Generator	7
Logical Complement (NOT)	8
Multiplier	21
Divider	22
Comparator	23
And Gate	24
Nand Gate	25
Ior Gate	26
Nor Gate	27
Eor Gate	28
Switch Function	30
Summer	31
Integrator	32
Limiter	33
Mode-controlled Integrator	34
Inverter	90

Table 4.5 Coding Convention for a/h Devices

CHAPTER 5

Sorting

5.1 Introduction

The next phase in the procedure is concerned with sorting a particular subset of the structure statements in the source CSMP program into a 'natural order'. This sorting operation is necessary because these structure statements are executed within the CATS program as a FORTRAN subroutine to compute the values of pnames as specified in the Initial segment and the maximum values of vnames as specified in the Dynamic segment. The proper evaluation of a FORTRAN assignment statement requires that all variables in the expression be previously defined. The purpose of the sorting operation is to ensure this condition. The notion of 'natural order' is treated more precisely after introducing some basic conventions.

The sorting operation extends over all the structure statements in the source program except these which have the form:

```
Y=INTGRL(IC,X)
```

or

```
Y=MODINT(IC,X1,X2,X)
```

The statements that are excluded from the sorting operation are clearly those which specify the state variables of the problem.

5.2 Natural Order

To facilitate the discussion, let the n-vector $X=(X_1,X_2,\ldots,X_n)$ denote the vector of names (pnames and vnames) which appear on

the left-hand side of structure statements in the source program and which are not state variables. Each such name X_k is assigned a value via a unique assignment statement which has the form:

$$X_k=\phi_k(X_1,X_2,\ldots,X_n)=\phi_k(X) \qquad 1\leq k\leq n$$

In general, however, ϕ_k explicitly depends on a subset of the components of X i.e. ϕ_k may explicitly depend only on n_k $(0\leq n_k\leq n)$* of the entries of X. Let X^k be the n_k-vector

$$(X^k_{\alpha_1}, X^k_{\alpha_2}, \ldots, X^k_{\alpha_{n_k}})^{**}$$

such that $\phi_k(X^k)\equiv\phi_k(X)$.

Suppose the n statements to be sorted (the sort-set) are written in some particular sequence. This sequence is in a natural order if the statement defining X_k is preceded by the n_k statements that define the components of the vector X^k. The object of the sorting procedure is to determine such a natural order for the n statements in the sort-set.

5.3 The Dependency Matrix

The Dependency matrix, D, characterizes the interactions among the statements in the sort-set. It is an n×n matrix whose entries are either 0 or 1. The statements in the sort-set have

---

* Note that since n_k will, for some k, equal zero X^k will in these circumstances be the null vector, denoted by θ.

** It is assumed that the relative order of the components in X^k is the same as in X i.e. $\alpha_i<\alpha_j$ if $i<j$

an initial order corresponding to the logical order in the SC vector*. It is this order that establishes the correspondence between the names and the components of the n-vector X.

The k^{th} row of D corresponds to the k^{th} statement of the sort-set, namely to the statement

$$X_k = \phi_k(X^k)$$

The entries in the k^{th} row of D are defined as follows:

$$D_{kj} = \begin{cases} 1 & \text{if } X_j \epsilon X^k \\ \text{otherwise} \end{cases}$$

To illustrate the creation of the Dependency matrix consider the following sequence of the source structure statements.

```
P2=P1*C
Y3=Y1+Y2+Y4
Y4=Y1*Y2
Y1=INTGRL(Y1Z,Y1+Y2+Y3)
Y5=Y4+Y2
Y2=INTGRL(Y2Z,Y1+Y2*Y4)
P1=A*B
```

If it is assumed that the logical order in which these statements are stored in the SC vector is as shown above, then the X vector of the earlier discussion is X=(P2,X3,Y4,Y5,P1). The 5×5 Dependency matrix then is as follows:

$$D = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

---

* This is the order established by the P and S tags associated with the statements.

5.4 Associated Parallel Vectors

The sorting procedure makes use of three parallel n vectors called ADR, NAMPOS and SEQ. The functions of each of those is briefly summarized below:

ADR Vector

Each entry in the ADR vector is a pointer to the starting location, in SC vector, of a statement in the sort-set. ADR(k) stores the starting location in SC of the k^{th} statement in the sort-set namely the statement $X_k=\phi_k(X^k)$.

NAMPOS Vector

The NAMPOS vector is initialized so that NAMPOS(j)=j, j=1,2,...,n. The entries in NAMPOS are manipulated during the sorting procedure. When NAMPOS(k)=j the implication is that the statement $X_k=\phi_k(X^k)$ from the sort-set is at the j^{th} position in the current ordering.

SEQ Vector

The SEQ vector acts as an inverse to the NAMPOS vector. The manipulations of the entries of SEQ are made to ensure that

SEQ(NAMPOS(k))=k

When SEQ(j)=k, the implication is that the j^{th} position in the current ordering is occupied by the statement $X_k=\phi_k(X^k)$.

5.5 The Sorting Operation

The sorting operation makes use of the Dependency matrix D which describes the interactions among the statements in the sort-set and the parallel vectors ADR, NAMPOS and SEQ. A brief

description of the sorting operation is given below together with a specific example. A flowchart of the procedure is given in Figure 5.1.

After initializing the Dependency matrix D and the vectors ADR, NAMPOS and SEQ the integer I is set to 1. The integer K, located at SEQ(I) defines a statement of the sort-set, namely $X_k=\phi_k(X^k)$ represented in the k^{th} row of D. The components of the vector X^k correspond to the non-zero entries in the k^{th} row of D.

Associated with each component of X^k there is a statement in the sort-set which has some position in the current ordering of the sort-set. Set MAX to the 'maximum position'. This position is determined by scanning the contents of NAMPOS. MAX has the property that if the statement $X_k=\phi_k(X^k)$ is moved into the MAX^{th} position in the sort-set then it will be properly positioned.

If MAX is less than K, then the implication is that the statement $X_k=\phi_k(X^k)$ is currently properly positioned (i.e. it is already preceded by the n_k statements corresponding to the entries in the vector X^k). The integer I is incremented by 1 and the statement indexed by SEQ(I) is processed next.

If MAX is greater than K, then the statement $X_k=\phi_k(X^k)$ has to be repositioned so as to follow the n_k statements corresponding to the entries of the vector X^k. The repositioning is done as follows.

All entries in SEQ between K and MAX are moved up by 1 and into MAX^{th} location is inserted the value of K. Corresponding updating is done in the inverse vector NAMPOS. The k^{th} position of the sort-set is now occupied by some other statement due to

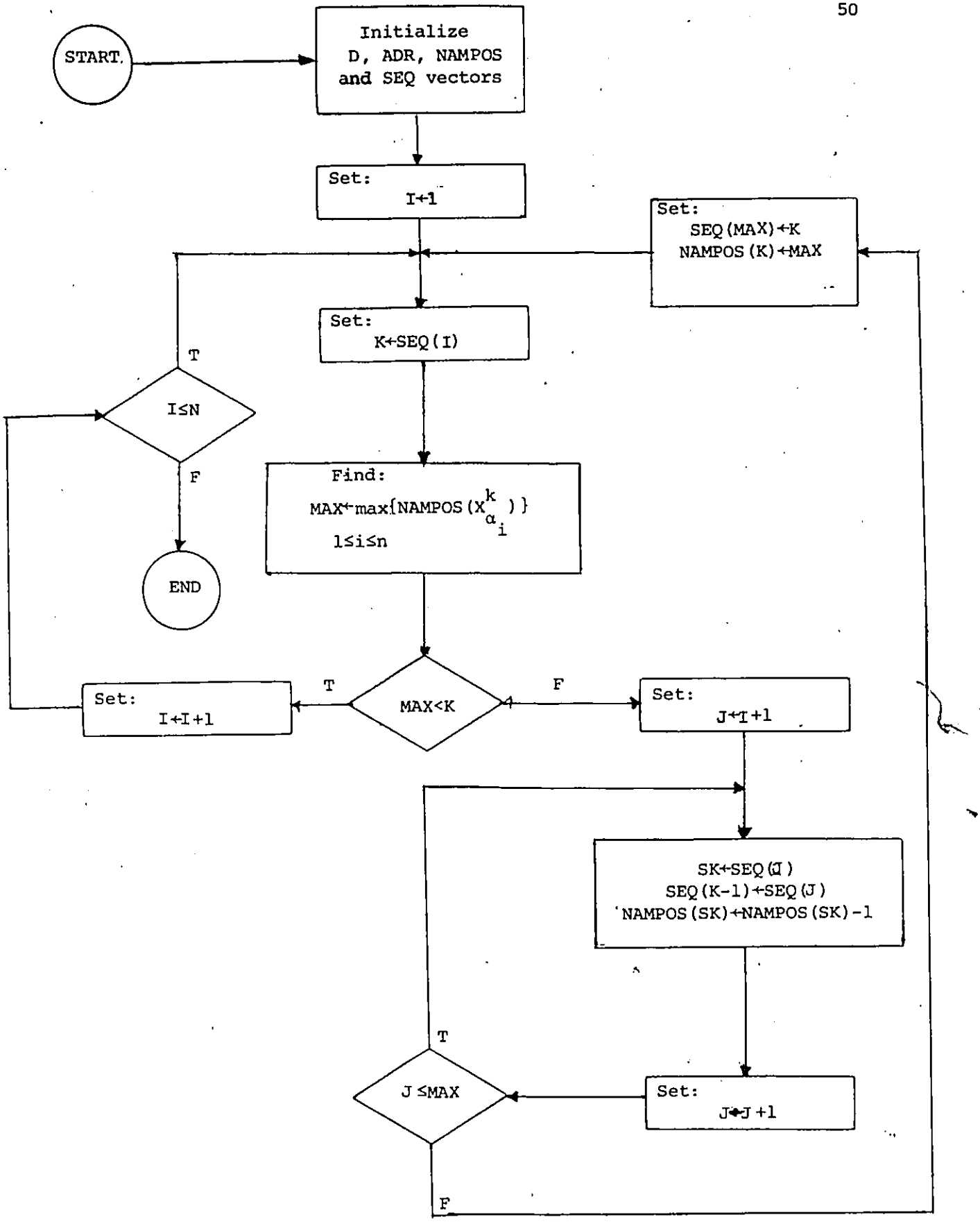


Figure 5.1 Flowchart of the Sorting Procedure

the above shifting. Thus I is not incremented and the statement indexed by SEQ(I) is processed next.

The sort-set of the example introduced in section 5.3 is used to illustrate the sorting operation in Figure 5.2.

5.6 Reordering of the Statements in the SC Vector

At the completion of the sorting operation a natural order for the sort-set is given by the contents of the SEQ vector. In general, statements from the Initial segment will be interspersed within the ordered sort-set. This set of statements forms a natural group which is independent of all other statements in the sort-set. In order to make it possible to execute these statements only once, it is necessary to move these statements to form a 'block' at the beginning of the ordered set.

The SEQ vector is processed twice.. Once to reposition the initial statements and later to reposition the remaining statements, namely the dynamic statements. After repositioning an initial statement, the corresponding entry in SEQ vector is assigned the value of 0 to denote that the statement has already been repositioned in SC vector. Recall that the starting addresses of the statements of the sort-set are stored in ADR.

The repositioning of a statement is done via manipulation of the P and S tags associated with the statement (i.e. there is no physical movement of the statement). To reposition a statement, the address of its predecessor is required. The predecessor will always be the last statement that was repositioned. However

Iteration #	Index I	Statement being Processed	Current Contents of SEQ	Current Contents of NAMPOS	Action Performed
1	1	$X_1=\phi_1(X^1)\equiv\phi_1(X_5)$	1 2 3 4 5	1 2 3 4 5	reposition X_1 in 5th position
2	1	$X_2=\phi_2(X^2)\equiv\phi_2(X_3)$	2 3 4 5 1	5 1 2 3 4	reposition X_3 in 2nd position
3	1	$X_3=\phi_3(X^3)\equiv\phi_3(\theta)$	3 2 4 5 1	5 2 1 3 4	none - because of null argument
4	2	$X_2=\phi_2(X^2)\equiv\phi_2(X_3)$	3 2 4 5 1	5 2 1 3 4	none - because X_2 properly positioned
5	3	$X_4=\phi_4(X^4)\equiv\phi_4(X_3)$	3 2 4 5 1	5 2 1 3 4	none - because X_4 properly positioned
6	4	$X_5=\phi_5(X^5)\equiv\phi_5(\theta)$	3 2 4 5 1	5 2 1 3 4	none - because of null argument
7	5	$X_1=\phi_1(X^1)\equiv\phi_1(X_5)$	3 2 4 5 1	5 2 1 3 4	none - because X_1 properly positioned

Figure 5.2

Snapshots of the Sorting Procedure Applied to

X=(P2,Y3,Y4,Y5,P1)

since the first statement in the sort-set does not have a predecessor, a special procedure is required.

If the first statement of the sort-set is already the first statement in SC, then the next statement of the sort-set is repositioned. If the first statement of the sort-set is not the first statement in SC, then it is made to follow the first statement and then these two statements are 'switched' so that the first statement of the sort-set becomes the first statement in SC and the statement that was previously the first statement in SC becomes the second statement in SC.

Once the first statement of the sort-set is repositioned the repositioning of other statements is trivial as shown in the Figure 5.3, where X_1 is the predecessor of X_2 the statement to be repositioned. P_1, P_2, S_1 and S_2 are the predecessors and successors of X_1 and X_2 respectively.

After application of the sorting and reordering procedures to the example problem introduced earlier the statements would appear in the following logical order within the SC vector.

```
P1=A*B
P2=P1*C
Y4=Y1*Y2
Y3=Y1+Y2+Y4
Y5=Y4+Y2
Y1=INTGRL(Y1Z,Y1+Y2+Y3)
Y2=INTGRL(Y2Z,Y1+Y2*Y4)
```

Note that as a result of these procedures, the state-variable statements within the program always 'sink-down' (in logical order) to the bottom of the program. The relative order of the state variable statements remain invariant.

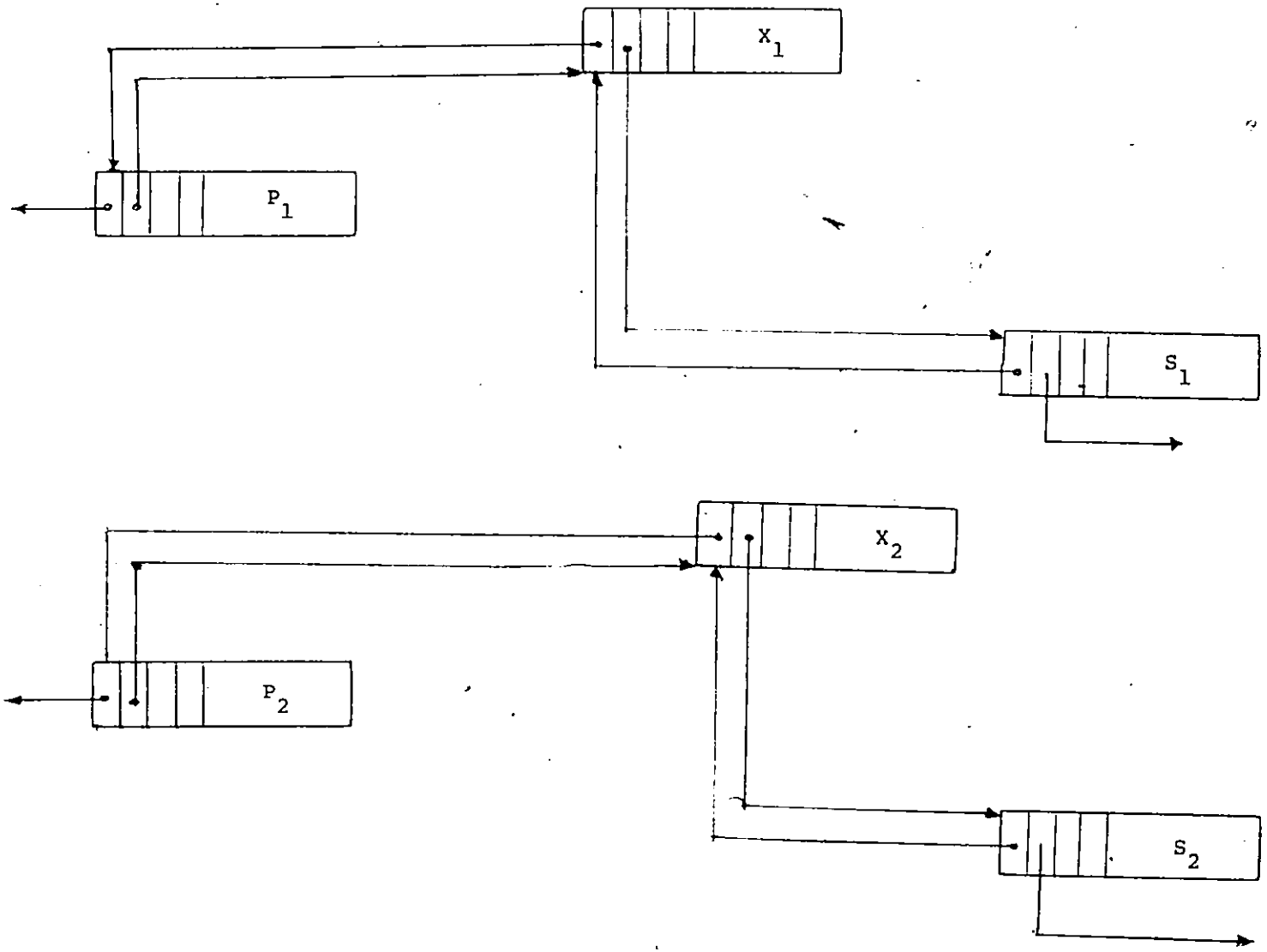


Figure 5.3a

Configuration before repositioning X_2 to logically follow X_1

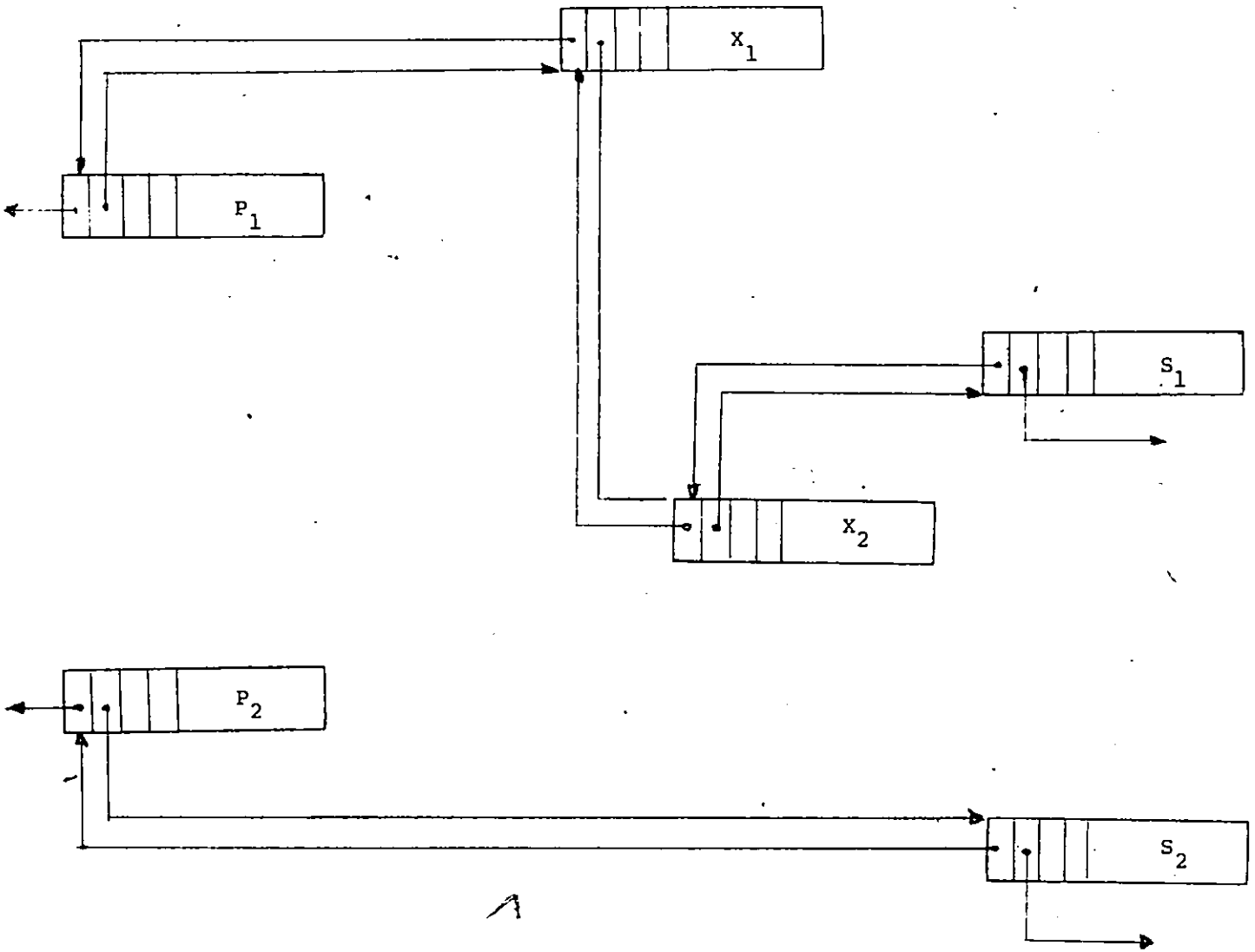


Figure 5.3b

Configuration after repositioning X_2 to logically follow X_1

CHAPTER 6

Magnitude Scaling and a/h Program Preparation

6.1 Introduction

One especially tedious aspect of the use of the a/h computer relates to the magnitude scaling requirement. This requirement originates because the computing devices in the machine are constrained to operate within fixed output limits (nominally taken to be ±1). In general the problem variables will swing outside these limits during the problem solution and hence, scaled versions of the problem variables must be created. This process, which is referred to as "magnitude scaling", requires an estimate of the maximum (absolute) value which the various problem variables will attain in the course of the problem solution.

A second peripheral matter which must be borne in mind when preparing an a/h computer circuit, is the "sign inversion" associated with some of the devices in the machine. For example, devices such as the summer and the integrator introduce an "extraneous" sign inversion when performing their prescribed operations. Such inversions must, of course, be duly taken into account in synthesizing the circuit. This procedure relies heavily on the "inverter" device which is available to generate the opposite polarity of any analog signal. Needless and excessive use of such devices must be avoided however in the interests of circuit efficiency.

This chapter describes how CATS deals with the above two matters in synthesizing its final a/h circuit specification for

the problem being treated.

6.2 Generating Maximum Value Estimates

The maximum value estimates for the magnitude scaling procedure are generated from data obtained by solving the model equations specified in the source CSMP program; and monitoring the maximum (absolute) value attained by each of the problem variables (vnames). This data is further processed to produce the desired maximum value estimates, as described below.

The central mechanism in this procedure is a predictor-corrector based routine for the solution of differential equations. This routine interacts with a secondary subroutine called RHS which procides the value of the derivative vector as required. The subroutine RHS is created by CATS from the source CSMP statements.

The proper functioning of RHS depends on the resolution of two peripheral matters: (i) The correct ordering (sequencing) of all statements that do not define the derivative of a state variable and (ii) The determination of the values of all pnames defined within the Initial segment of the source program. The first of these matters was discussed in detail in Chapter 5.

The second matter is handled by organizing RHS so that the statements in the Initial segment of the source program appear in a segment of RHS which is executed once at the beginning of the integration procedure (i.e. when the problem independent variable equals zero). The values for the pnames generated at

this step are then appropriately stored in the PVAL vector for subsequent use.

Having properly organized the RHS subroutine, the differential equation solving routine is executed, and the maximum excursion of each of the problem variables is established. In some cases the maximum value estimate used in the magnitude scaling operation is then derived by rounding upward the fourth significant digit so that it is either a 0 or a 5. For example, 155.431 and 0.0032163 become respectively 155.5 and 0.00322. In the other cases, the maximum value estimate for a device output is determined in alternate way using information about the inputs and/or parameters of the device. Table 6.1 summarizes, for each of the devices, how the maximum value estimate (Y_m) is determined for subsequent use in the magnitude scaling procedure.

6.3 Magnitude Scaling Procedure

For proper operation, all devices on the a/h computer must be programmed to ensure that their output excursions during the problem solution do not exceed prescribed limits. These limits are usually taken to be ±1. This requirement gives rise to the procedure known as magnitude scaling which is an essential operation in a/h programming.

In effect this operation involves the creation of a normalized version of each of the problem variable (device outputs) and the introduction of a corresponding gain adjustment to preserve the integrity of the original equations. The operation uses the

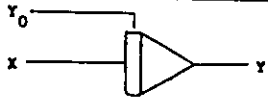
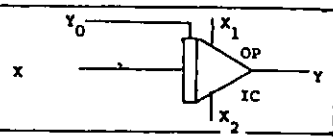
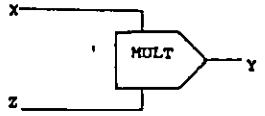
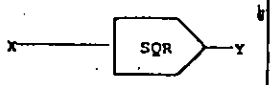
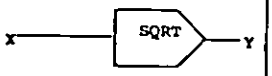
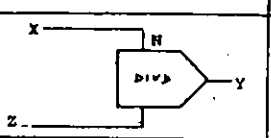
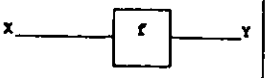
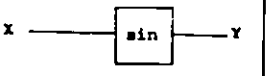
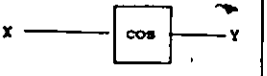
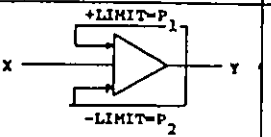
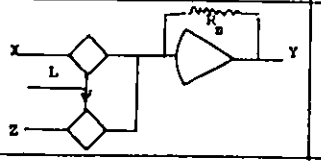
Analog Device	Representation	Input-Output Relation	Maximum Value Determination*
summer		$Y=-(X+Z)$	$Y_m=\overline{Y}_m$
integrator		$Y=-(\int_0^t X dt+Y_0)$	$Y_m=\overline{Y}_m$
mode controlled integration		$Y=\begin{cases}-(\int_0^t X dt+Y_0) & \text{for } X_1>0, \text{ and } X_2\\ -Y_0 & \text{for } X_1\leq 0, \text{ and } X_2>0\\ \text{last output} & \text{for } X_1\leq 0 \text{ and } X_2\leq 0\end{cases}$	$Y_m=\overline{Y}_m$
multiplier		$Y=X*Z$	$Y_m=X_m*Z_m$
squarer		$Y=X^2$	$Y_m=X_m^2$
square rooter		$Y=+\sqrt{X}$	$Y_m=+\sqrt{X_m}$
divider		$Y=X/Z$	$Y_m=\overline{Y}_m$
function generator		$Y=f(X)$	$Y_m=\overline{Y}_m$
sine		$Y=\sin(X)$	$Y_m=1$
cosine		$Y=\cos(X)$	$Y_m=1$
limiter		$Y=\begin{cases}P_1 & \text{for } X<-P_1\\ -X & \text{for } -P_1\leq X-P_2\\ P_2 & \text{for } X>-P_2\end{cases}$	$Y_m=\max(\lvert P_1\rvert,\lvert P_2\rvert)$
switch		$Y=\begin{cases}-X & \text{for } L=0\\ -Z & \text{for } L=1\end{cases}$	$Y_m=\max(X_m,Y_m)$

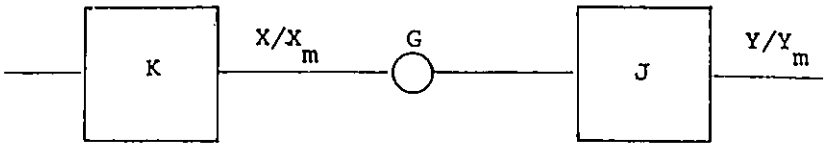
Table 6.1

Determination of Maximum Values

$\overline{Y}_m$ denotes the computed maximum, modified by the rounding procedure

maximum value estimates for the variables.

The general configuration to be treated concerns a device K whose output is X/X_m which is in turn an input to a device J whose output is Y/Y_m e.g.



The auxiliary gain, G, introduced between these devices to preserve the original equations is dependent on the specific devices K and J. In many cases G=1; namely when device J is any of the following: multiplier, squarer, square rooter and function generator. When J is not one of these devices, the necessary value for G is given by the table entries in Table 6.2.

The direct application of the magnitude scaling procedure is based on the G values in Table 6.2 can lead to several awkward and/or unrealisable* a/h circuit configurations. A 'housekeeping' operation is therefore necessary to eliminate these situations.

The situation requiring attention are listed in Table 6.3. The right-hand column shows the necessary operations to be performed.

The CATS program assumes that all angular displacements are measured in radians and that a particular hardware device is used for performing the sine and cosine operations. The last row of Table 6.3 reflects the step required to make these two assumptions compatible.

---

* potentiometer coefficients greater than unity

Device J / Device K	Linear	Divider (num)	Divider (denom)	Comparator	Switch	Sine/cosine
linear	X_m/Y_m	X_m/Y_m	X_m	X_m	X_m/Y_m	X_m
multiplier	X_m/Y_m	X_m/Y_m	X_m	X_m	X_m/Y_m	X_m
squarer	X_m/Y_m	X_m/Y_m	X_m	X_m	X_m/Y_m	X_m
square rooter	X_m/Y_m	X_m/Y_m	X_m	X_m	X_m/Y_m	X_m
divider	X_m/Y_m	X_m/Y_m	X_m	X_m	X_m/Y_m	X_m
switch	X_m/Y_m	X_m/Y_m	X_m	X_m	X_m/Y_m	X_m
function generator	X_m/Y_m	X_m/Y_m	X_m	X_m	X_m/Y_m	X_m
sine/ cosine	$1/Y_m$	X_m/Y_m	1	1	$1/Y_m$	X_m

Table 6.2

Auxilary Gain, G, for Magnitude Scaling

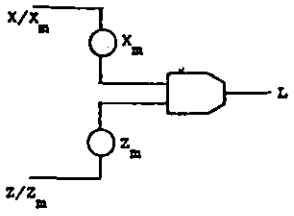
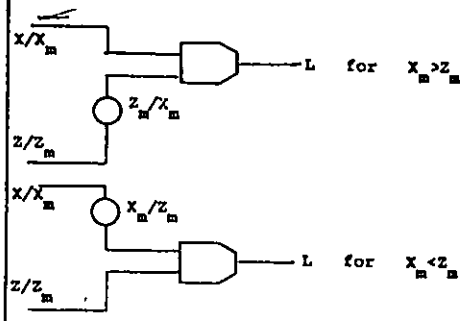
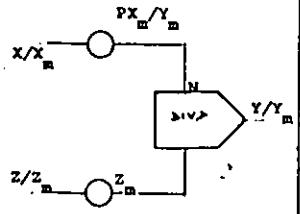
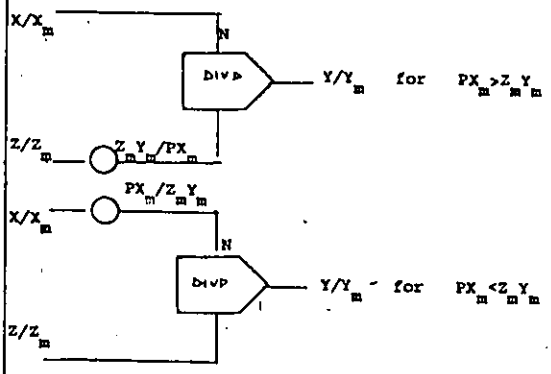
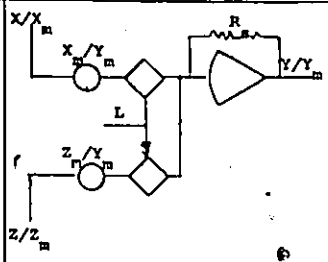
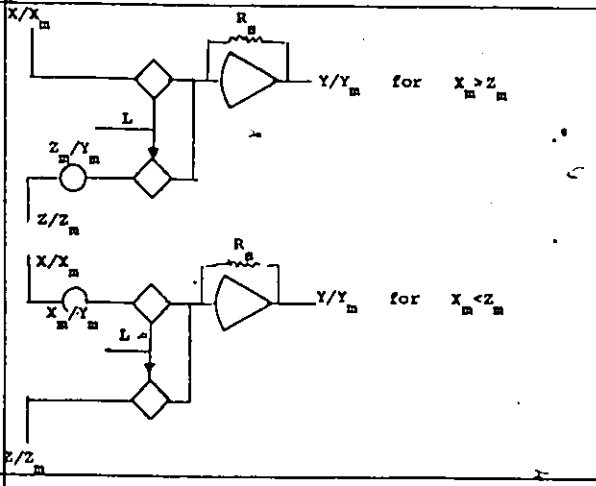
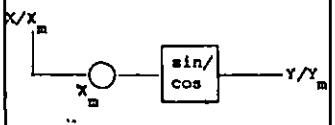
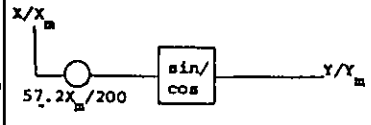
Device	Configuration after magnitude scaling	Configuration after 'housekeeping'
Comparator		
Divider		
Switch		
Sin/cosine		

Table 6.3

'Housekeeping' Operation for Magnitude Scaling

6.4 Determining the Polarities of the Output Variables

Many of the computing devices in an a/h computer have an intrinsic sign inversion property which must be taken into account in the programming procedure. Some devices, furthermore, simultaneously require both polarities of their input variable(s), while others intrinsically make available both polarities of their output. These device properties complicate the generation of a proper, and reasonably efficient, a/h program.

One straight-forward approach to the problem is to make "wholesale" use of the inverter device* to ensure that all variables are available in their "primary" (rather than inverted) form. Although this simplifies the conceptual view of the final circuit, it can lead to an unnecessarily complex program whose accuracy is degraded due to the use of many redundant devices (i.e. inverters).

Thus the following general problem must be handled: Given the need to realize the operation specified by

$$x_k = \phi_k(x_1, x_2, \ldots, x_n)$$

is it best to arrange the program so that x_k or $-x_k$ is generated by the device associated with the operator ϕ_k? The natural criterion for making this decision is the minimization of the total number of inverters required in the final program.

---

* The inverter is the most basic active analog computer device. It does not, however, perform a computationally useful task. It is a single input device whose output label is derived by simply reversing the algebraic sign associated with the input.

The problem has been considered in detail in [7] and an algorithm for its solution is outlined there. This algorithm is incorporated in CATS for handling the polarity assignment problem. In view of the availability of this reference, further discussion of the problem and its solution will not be undertaken in this thesis.

The output of the algorithm, as implemented in CATS, consists of the polarities of the outputs. These are stored in a vector called MROW which is parallel to the LABEL vector (see section 2.4). The polarity of the output of the J^{th} device is determined by the J^{th} entry in MROW, whose value is between 0 and 4, defined as follows:

0,2 - the output is generated with a positive polarity

1,3 - the output is generated with a negative polarity

Furthermore the values 2 and 3 imply that some other device needs the opposite polarity of this output and therefore an invertor is required to generate the appropriate polarity. The values 2 and 3 are updated respectively to 0 and 1 after inserting the inverters to generate the correct polarity.

At this stage all the information required for programming an a/h computer has been established. This information is stored in the vectors, associated with the output table (Chapter 2).

6.5 The a/h Program Output

The principle output generated by CATS is a table which contains the information for the a/h program. The rows of this

table are blocked into groups where each horizontal block is associated with a single a/h device. The table has thirteen columns and a summary of the data provided in each column is given below.

Column 1: Contains a unique identification number assigned to the a/h device.

Column 2: Contains the label for the device output. This label will either be one of the vnames in the source CSMP program or a system generated name of the form Sdd where d is a decimal digit. The name may appear unsigned which indicates that it is generated with positive polarity. Alternately, the vname may be preceded by a - sign to indicate that it is generated with negative polarity.

Column 3: Contains the maximum value estimate for the device output.

Column 4: Contains the specification of the type of a/h device.

Column 5: Contains the component numbers of the devices which provide inputs to the device in question.

Column 6: Contains the output labels associated with the devices referred to in column 5.

Column 7: Each input entering a device is assumed to have an associated "gain" which normally is a multiplicative constant (pname) specified in the source program. Such pnames are given in column 7 for each input to the device in question. If the pname is assigned a negative value in a CSMP data statement, then a negative sign

precedes the pname. If the pname appears as a divisor, rather than as a multiplicative constant, then a / precedes the pname. If no explicit pname appears in the source program, then a multiplicative constant of value +1 is assumed and the system pname S101 accordingly appears in column 7.

Column 8: Contains the values of the pnames appearing in column 7. These values are appropriately altered to be consistent with any "modifier" (a - or /) which may precede the pname in column 7.

Column 9: Contains a secondary gain factor associated with each input arising from magnitude scaling requirements.

Column 10: Contains the "net input gain" for each input to the device. This value is the product of the entries in columns 8 and 9.

Column 11: Whenever the net input gain (entry in column 10) is not 0 or 1, then a potentiometer is required in the input path in question. Column 11 contains the unique identification number for each required potentiometer.

Column 12: Contains the component number(s) of the device(s) which require(s) the output of the device in question, as an input.

Column 13: Contains the output labels associated with the devices referred to in column 12.

It should be noted that the first input to an integrator (or mode controlled integrator) represents the initial condition input.

The second and third inputs to the mode controlled integrator are the logical input (OP and IC respectively). The first and second inputs to a limiter are its lower and upper limits respectively and finally the first input to the SW (switch) is its control input.

Specific examples of the form of the a/h program output can be found in Chapter 7.

CHAPTER 7

Examples

7.1 Introduction

In this chapter three examples are presented to illustrate the capability of the CATS program. These examples have all been taken from the literature and are designated as follows:

(1) Automobile Suspension System [8]

(2) Body Water Regulatory System [9]

(3) Chased Target Problem [10]

For each example, the mathematical equations governing the system are given, followed by the CSMP program representing the system. The a/h circuit diagram is drawn from the patching information outputted by the CATS program.

7.2 Example 1: Automobile Suspension System

Consider the system in Figure 7.1 which is a simplified model of one wheel of an automobile suspension system. The differential equations of the system are derived by equating the forces acting upon the masses involved in the system. These equations are

$$M_1\ddot{X}_1 + D_1(\dot{X}_1-\dot{X}_2) + K_1(X_1-X_2) + F_a = 0$$

$$M_2\ddot{X}_2 + D_1(\dot{X}_2-\dot{X}_1) + K_1(X_2-X_1) + K_2(X_2-X_3) = 0$$

with initial conditions

$$X_1(0) = \dot{X}_1(0) = X_2(0) = \dot{X}_2(0) = 0$$

where

M_1 = one fourth mass of the automobile = 25 slugs

M_2 = mass of one wheel = 2 slugs

K_1 = linear spring constant of automobile = 1000 lb/ft

K_2 = linear spring constant of tire = 4500 lb/ft

D_1 = damping constant of shock absorber = 100 lb/ft/sec

X_1 = displacement of body of automobile (feet)

X_2 = displacement of wheel (feet)

X_3 = function describing road profile defined as follows:

$$X_3 = \begin{cases} 1/12 \text{ ft.} & \text{if } t \le 0.5 \\ 0 & \text{otherwise} \end{cases}$$

F_a = an externally applied force defined as follows:

$$F_a = \begin{cases} 5t & \text{if } t \le 1 \\ 0.2M_1 & \text{if } t > 1 \end{cases}$$

Furthermore a quality of ride index is introduced into the problem and this index has the following form:

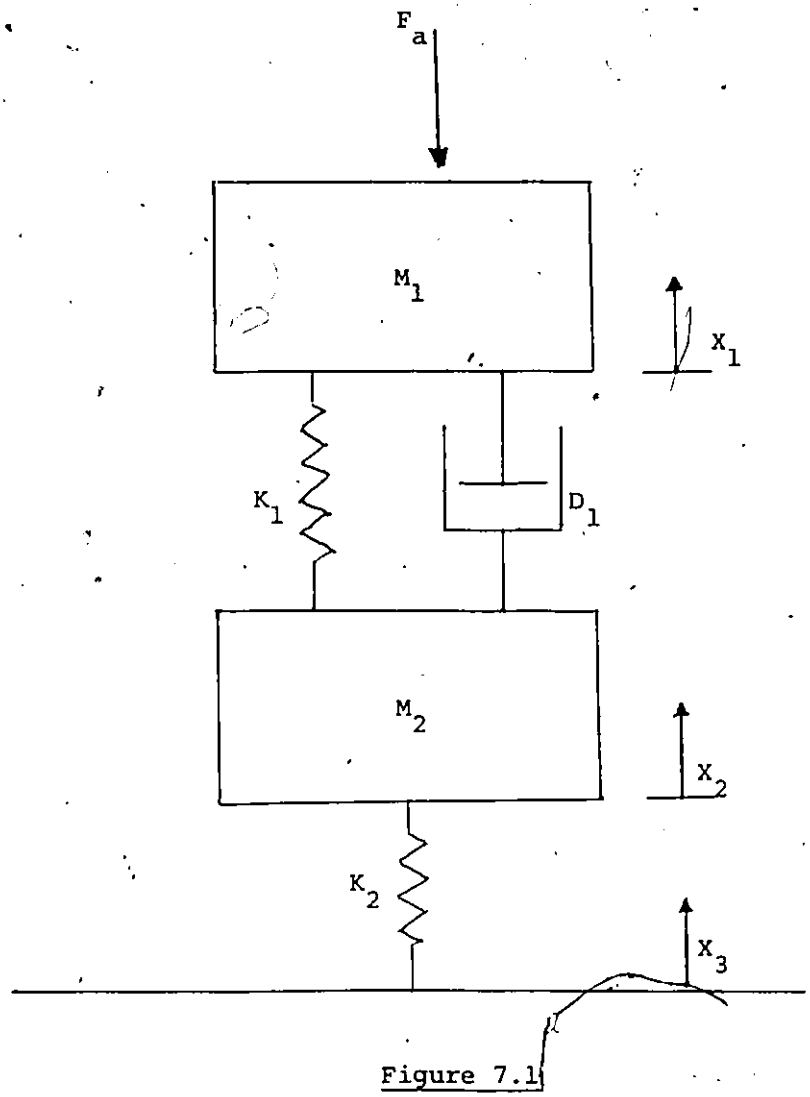


Figure 7.1

Simplified Model of Automobile Suspension System

$$I_r = \int_0^T (x_1)^2 t\,dt; \quad T=10$$

A CSMP program corresponding to this system is given in Figure 7.2. The resulting scaling and patching information generated by CATS is given in Figure 7.3. The a/h computer diagram drawn from the patching data is shown in Figure 7.4.

```
*         EXAMPLE 1 - AUTOMOBILE SUSPENSION SYSTEM

X1=INTGRL(0.,X1D)
X1D=INTGRL(0.,D1*X2D/M1-D1*X1D/M1-K1*X1/M1+K1*X2/M1-FA/M1)
X2=INTGRL(0.,X2D)
X2D=INTGRL(0.,D1*X1D/M2-D1*X2D/M2-(K1+K2)*X2/M2+K1*X1/M2+K2*X3/M2)
IRD=TIME*X1**2
IR=INTGRL(0.,IRD)
L=COMPAR(T1,TIME)
X3=INSW(L,0,A)
FA=LIMIT(0.,0.2*M1,5*TIME)
TIMER  FINTIM=10,DELT=0.01,OUTDEL=0.1
METHOD RKSFX
PARAM  M1=25,M2=2,K1=1000,K2=4500,D1=100,A=0.0833,T1=0.05
PRTPLT  X1,X1D,X2,X2D,IRD,IR,L,X3,FA
END
```

Figure 7.2

CSMP program for Example 1

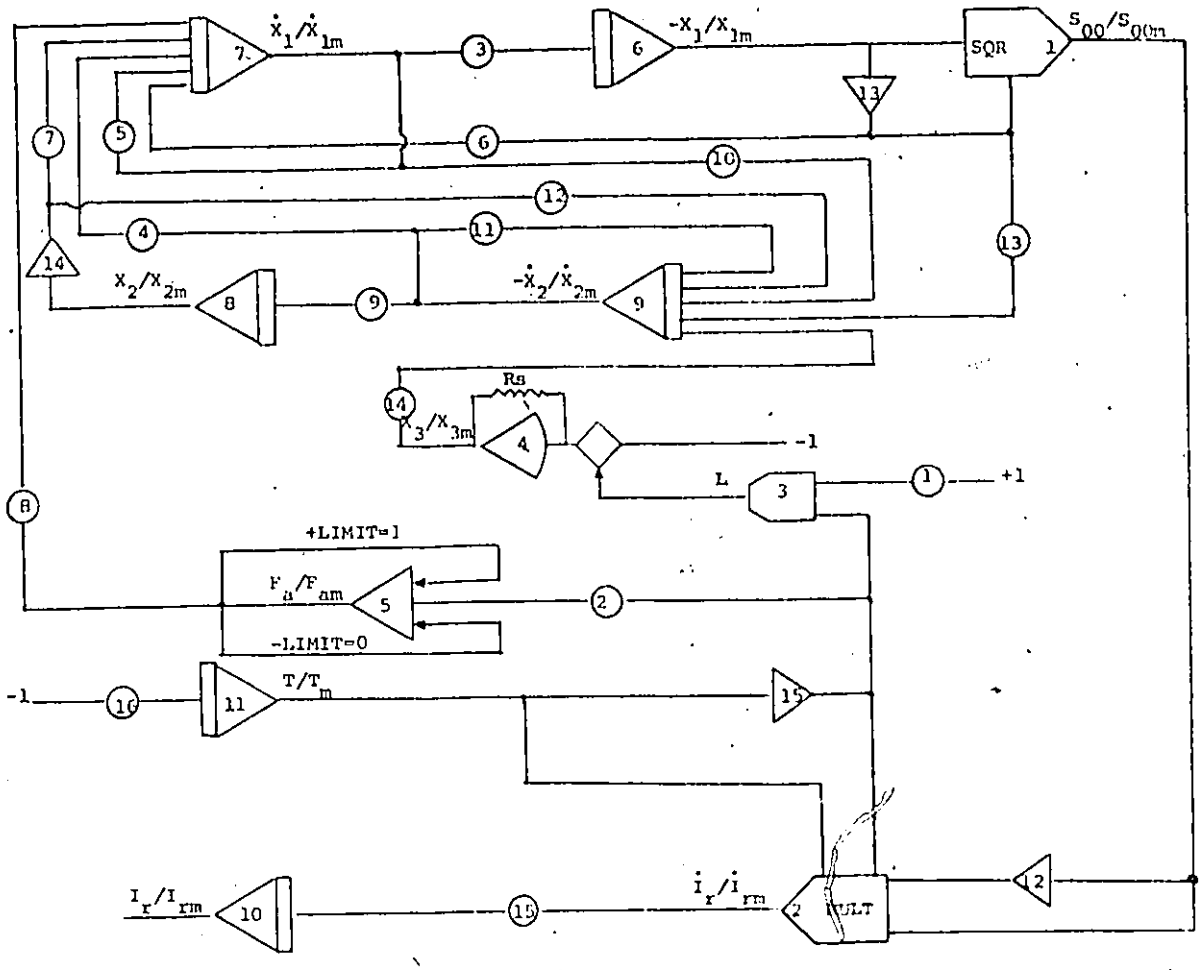
TABLE FOR ANALOG CIRCUIT

OUTPUT COMP #	OUTPUT LABEL	MAXIMUM VALUE	DEVICE TYPE	INPUT(S) COMP #	INPUT(S) LABEL	PROBLEM PARM	GAIN VALUE	SCALE FACTOR	NET INPUT GAIN	PGT #	TARGET(S) COMP #	TARGET(S) LABEL
1	S00	0.6250E-03	SQR	6	-X1	S101	0.1000E 01	0.1000E 01	0.1000E 01		2	-IRD
				13	X1	S101	0.1000E 01	0.1000E 01	0.1000E 01		12	-S00
2	-IRD	0.6250E-02	MULT	11	TIME	S101	0.1000E 01	0.1000E 01	0.1000E 01		10	IR
				1	S00	S101	0.1000E 01	0.1000E 01	0.1000E 01			
				15	-TIME	S101	0.1000E 01	0.1000E 01	0.1000E 01			
				12	-S00	S101	0.1000E 01	0.1000E 01	0.1000E 01			
3	L	0.1000E 01	COMP		PREF	T1	0.5000E-01	0.1000E 00	0.5000E-02	1	4	X3
				15	-TIME	S101	0.1000E 01	0.1000E 01	0.1000E 01			
4	X3	0.8330E-01	SW	3	L	S101	0.1000E 01	0.1000E 01	0.1000E 01		9	-X2D
					NREF	A	0.8330E-01	0.1200E 02	0.1000E 01			
5	FA	0.5000E 01	LIMIT		NREF	S100	0.0	0.0	0.0		7	X1D
					PREF	S106	0.5000E 01	0.2000E 00	0.1000E 01			
				15	-TIME	S104	0.5000E 01	0.2000E 01	0.1000E 02	2		
6	-X1	0.2500E-01	INT		PREF	S100	0.0	0.0	0.0		1	S00
				7	X1D	S101	0.1000E 01	0.1600E 02	0.1600E 02	3	13	X1
7	X1D	0.4000E 00	INT		NREF	S100	0.0	0.0	0.0		6	-X1
				9	-X2D	S107	0.4000E 01	0.7500E 01	0.3000E 02	4	7	X1D
				7	X1D	S108	0.4000E 01	0.1000E 01	0.4000E 01	5	9	-X2D
				13	X1	S109	0.4000E 02	0.6250E-01	0.2500E 01	6		
				14	-X2	S110	0.4000E 02	0.2500E 00	0.1000E 02	7		
				5	FA	/M1	0.4000E-01	0.1250E 02	0.5000E 00	8		
8	X2	0.1000E 00	INT		NREF	S100	0.0	0.0	0.0		14	-X2
				9	-X2D	S101	0.1000E 01	0.3000E 02	0.3000E 02	9		
9	-X2D	0.3000E 01	INT		PREF	S100	0.0	0.0	0.0		7	X1D
				7	X1D	S111	0.5000E 02	0.1333E 00	0.6667E 01	10	8	X2
				9	-X2D	S112	0.5000E 02	0.1000E 01	0.5000E 02	11	9	-X2D
				14	-X2	S113	0.2750E 04	0.3333E-01	0.9167E 02	12		
				13	X1	S114	0.5000E 03	0.8333E-02	0.4167E 01	13		
				4	X3	S115	0.2250E 04	0.2777E-01	0.6247E 02	14		
10	IR	0.2000E-02	INT		NREF	S100	0.0	0.0	0.0			
				2	-IRD	S101	0.1000E 01	0.3125E 01	0.3125E 01	15		
11	TIME	0.1000E 02	INT		NREF	S100	0.0	0.0	0.0		2	-IRD
					NREF	S101	0.1000E 01	0.1000E 00	0.1000E 00	16	15	-TIME
12	-S00	0.6250E-03	INV	1	S00	S101	0.1000E 01	0.1000E 01	0.1000E 01		2	-IRD
13	X1	0.2500E-01	INV	6	-X1	S101	0.1000E 01	0.1000E 01	0.1000E 01		1	S00
											7	X1D
											9	-X2D
14	-X2	0.1000E 00	INV	8	X2	S101	0.1000E 01	0.1000E 01	0.1000E 01		7	X1D
											9	-X2D
15	-TIME	0.1000E 02	INV	11	TIME	S101	0.1000E 01	0.1000E 01	0.1000E 01		2	-IRD
											3	L
											5	FA

NO TIME SCALE FACTOR NEEDED

Figure 7.3

Patching Information for Example 1



Pot. Number	Description	Pot. Number	Description
1	T_1/T_m	9	$\dot{X}_{2m}/X_{2m}$
2	$5*T_m/F_{am}$	10	$(\dot{X}_{1m}*D_1)/(X_{2m}*M_2)$
3	$\dot{X}_{1m}/X_{1m}$	11	D_1/M_2
4	$'(\dot{X}_{2m}*D_1)/(\dot{X}_{1m}*M_1)$	12	$(X_{2m}*(K_1+K_2))/(\dot{X}_{2m}*M_2)$
5	D_1/M_1	13	$(X_{1m}*K_1)/(\dot{X}_{2m}*M_2)$
6	$(X_{1m}*K_1)/(\dot{X}_{1m}*M_1)$	14	$(X_{3m}*K_2)/(\dot{X}_{2m}*M_2)$
7	$(X_{2m}*K_1)/(\dot{X}_{1m}*M_1)$	15	$\dot{I}_{rm}/I_{rm}$
8	$F_{am}/(\dot{X}_{1m}*M_1)$	16	$1/T_m$

Figure 7.4

Patching Diagram from CATS for Example 1

7.3 Example 2: Body Water Regulatory System

Figure 7.5 shows a simplified block diagram of the feedback mechanism of the body water regulatory system. The differential equations of the system are formulated by dividing the model into three compartments representing the stomach, the small intestine and the plasma volume. These equations are

$$\dot{S} = D_r - K_1 S$$

$$\dot{G} = K_1 S - K_2 S$$

$$\dot{P} = K_2 G - U$$

$$\dot{A} = F - K_3 A$$

$$A_c = A/P$$

$$K_2 = 3(C_1 O + C_2)P$$

with initial conditions $S(0) = G(0) = 0$, $P(0) = 3$, $A(0) = 10$ where

S = volume of water in stomach (liters)

G = volume of water in intestine (liters)

P = volume of plasma (liters)

A = level of ADH in plasma (milliunits)

A_c = concentration of ADH in plasma (milliunits/liter)

U = urine flow rate (milliliter/minute) defined as follows

$$U = \begin{cases} 20-6A_c & \text{if } A_c \leq 3.17 \\ 1 & \text{otherwise} \end{cases}$$

O = osmolarity of ingested drink = 300 milliosmols

K_1 = stomach loss rate constant = 4 $(\text{hours})^{-1}$

K_2 = intestinal loss rate parameter $(\text{hours})^{-1}$

K_3 = rate of disposition of ADH = 2 $(\text{hours})^{-1}$

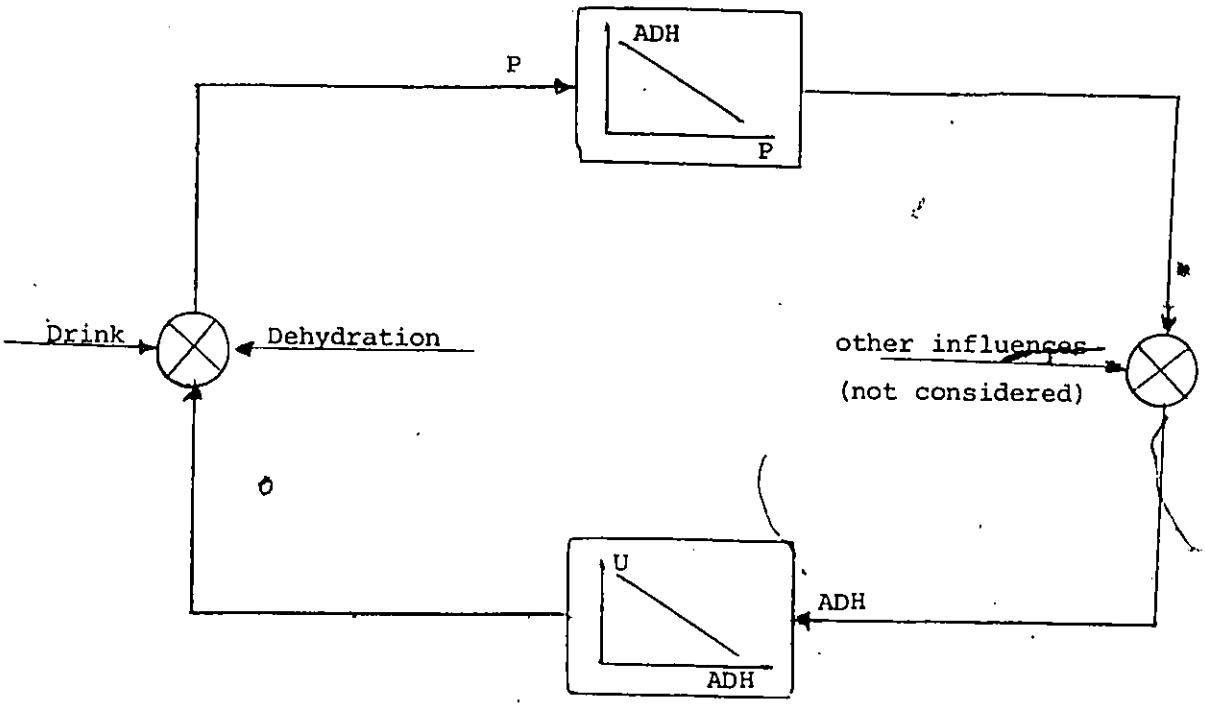


Figure 7.5

Simplified Block Diagram of the

Body Water Regulatory System

D_r = drinking rate (liter/hour) defined as follows:

$$D_r = \begin{cases} 6 & \text{if } t \leq 0.167 \text{ hours} \\ 0 & \text{otherwise} \end{cases}$$

C_1 = intestinal loss rate constant = -0.0097

C_2 = intestinal loss rate constant = 3

F = functional relationship between the rate of ADH release and the plasma volume defined as follows:

$$F = \begin{cases} 420-133P & \text{if } P \leq P_2 = 3 \\ 170-50P & \text{if } P_2 < P \leq P_3 = 3.4 \\ 0 & \text{if } P > P_3 \end{cases}$$

A CSMP program corresponding to this system is given in Figure 7.6. The resulting scaling and patching information generated by CATS is given in Figure 7.7. The a/h computer diagram drawn from the patching data is shown in Figure 7.8.

```
*         EXAMPLE 2 - BODY WATER REGULATORY SYSTEM

MACRO   YY=INSW(L,X1,X2)
        LDUMY=L-0.5
        YY=FCNSW(LDUMY,X1,X1,X2)
ENDMAC

INCON SZ=0,GZ=0,PZ=3,AZ=10
PARAM P2=3,P3=3.4,V=1,TIN=0.167,O=(300,250,200,100,0)
CONST C1=-0.0097,C2=3,K1=4,K3=2

DYNAMIC
Z=K2*G
S=INTGRL(SZ,DR-K1*S)
G=INTGRL(GZ,K1*S-Z)
P=INTGRL(PZ,Z-0.06*U)
A=INTGRL(AZ,F-K3*A)
AC=A/P
K2=3*(C1*O+C2)/P

*         GENERATION OF F(ADH SECRETION RATE)
F1=170-50*P
F2=250-P*83.3
F1S=INSW(L1,0,F1)
F2S=INSW(L2,0,F2)
L1=CCMPAR(P3,P)
L2=CCMPAR(P2,P)
F=F1S+F2S

*         GENERATION OF U(URINE FLOW RATE)
U1=19-6*AC
U1S=INSW(L3,0,U1)
L3=COMPAR(U1,0)
U=U1S+1

*         GENERATION OF DR(DRINKING RATE)
L=COMPAR(TIME,TIN)
DR=INSW(L,6*V,04)

TIMER FINTIM=10,DELT=0.01,OUTDEL=0.1
PRTPLT   AC,A,P,F,U
METHOD  RKS
END
```

Figure 7.6

CSMP program for Example 2

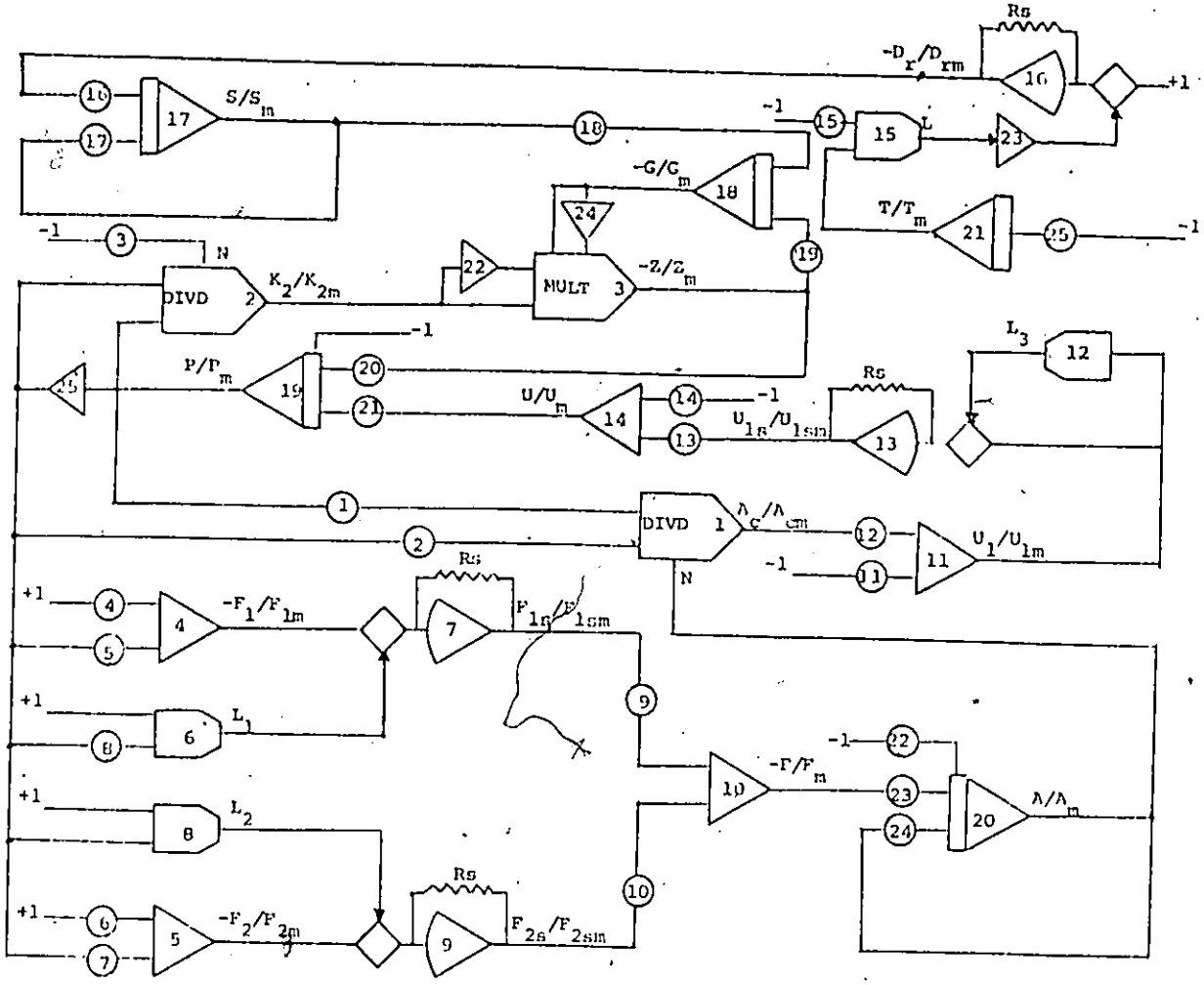
TABLE FOR ANALOG CIRCUIT

OUTPUT COMP #	OUTPUT LABEL	MAXIMUM VALUE	DEVICE TYPE	INPUT(S) COMP #	INPUT(S) LABEL	PROBLEM PARM	GAIN VALUE	SCALE FACTOR	NET INPUT GAIN	POT #	TARGET(S) COMP #	TARGET(S) LABEL
1	AC	0.4500E 01	DIVD	20	A	S101	0.1000E 01	0.1000E 01	0.1000E 01		11	U1
				25	-P	S101	0.1000E 01	0.9000E 00	0.9000E 00	1		
				19	P	S101	0.1000E 01	0.9000E 00	0.9000E 00	2		
2	K2	0.1000E 00	DIVD		NREF	S113	0.2700E 00	0.3333E 01	0.9000E 00	3	3	-Z
				25	-P	S101	0.1000E 01	0.1000E 01	0.1000E 01		22	-K2
				19	P	S101	0.1000E 01	0.1000E 01	0.1000E 01			
3	-Z	0.1000E 00	MULT	2	K2	S101	0.1000E 01	0.1000E 01	0.1000E 01		18	-G
				24	G	S101	0.1000E 01	0.1000E 01	0.1000E 01		19	P
				22	-K2	S101	0.1000E 01	0.1000E 01	0.1000E 01			
				18	-G	S101	0.1000E 01	0.1000E 01	0.1000E 01			
4	-F1	0.2500E 02	SUMMER		PREF	S105	0.1700E 03	0.4000E-01	0.6800E 01	4	7	F1S
				25	-P	S106	0.5000E 02	0.1200E 00	0.6000E 01	5		
5	-F2	0.4500E 01	SUMMER		PREF	S107	0.2500E 03	0.2222E 00	0.5556E 02	6	9	F2S
				25	-P	S108	0.8330E 02	0.6667E 00	0.5553E 02	7		
6	L1	0.1000E 01	COMP		PREF	P3	0.3400E 01	0.2941E 00	0.1000E 01		7	F1S
				25	-P	S101	0.1000E 01	0.8824E 00	0.8824E 00	8		
7	F1S	0.2500E 02	SW	6	L1	S101	0.1000E 01	0.1000E 01	0.1000E 01		10	-F
				4	-F1	S101	0.1000E 01	0.1000E 01	0.1000E 01			
8	L2	0.1000E 01	COMP		PREF	P2	0.3000E 01	0.3333E 00	0.1000E 01		9	F2S
				25	-P	S101	0.1000E 01	0.1000E 01	0.1000E 01			
9	F2S	0.4500E 01	SW	8	L2	S101	0.1000E 01	0.1000E 01	0.1000E 01		10	-F
				5	-F2	S101	0.1000E 01	0.1000E 01	0.1000E-01			
10	-F	0.3000E 02	SUMMER	7	F1S	S101	0.1000E 01	0.8333E 00	0.8333E 00	9	20	A
				9	F2S	S101	0.1000E 01	0.1500E 00	0.1500E 00	10		
11	U1	0.7000E 01	SUMMER		NREF	S109	0.1900E 02	0.1429E 00	0.2714E 01	11	12	L3
				1	AC	S110	0.6000E 01	0.6429E 00	0.3857E 01	12	13	-U1S
12	L3	0.1000E 01	COMP	11	U1	S101	0.1000E 01	0.1000E 01	0.1000E 01		13	U1S
13	-U1S	0.7000E 01	SW	12	L3	S101	0.1000E 01	0.1000E 01	0.1000E 01		14	U
				11	U1	S101	0.1000E 01	0.1000E 01	0.1000E 01			
14	U	0.1500E 01	SUMMER	13	-U1S	S101	0.1000E 01	0.4667E 01	0.4667E 01	13	19	P
					NREF	S101	0.1000E 01	0.6667E 00	0.6667E 00	14		
15	L	0.1000E 01	COMP	21	TIME	S101	0.1000E 01	0.1000E 01	0.1000E 01		23	-L
					NREF	TIN	0.1670E 00	0.1000E 00	0.1670E-01	15		
16	-DR	0.6000E 01	SW	23	-L	S101	0.1000E 01	0.1000E 01	0.1000E 01		17	S
					PREF	S114	0.6000E 01	0.1667E 00	0.1000E 01			
17	S	0.7500E 00	INT		NREF	SZ	0.0	0.0	0.0		17	S
				16	-DR	S101	0.1000E 01	0.8000E 01	0.8000E 01	16	18	-G
				17	S	K1	0.4000E 01	0.1000E 01	0.4000E 01	17		
18	-G	0.1000E 01	INT		PREF	GZ	0.0	0.0	0.0		3	-Z
				17	S	K1	0.4000E 01	0.7500E 00	0.3000E 01	18	24	G
				3	-Z	S101	0.1000E 01	0.1000E 01	0.1000E 01	19		
19	P	0.3000E 01	INT		NREF	PZ	0.3000E 01	0.3333E 00	0.1000E 01		1	AC
				3	-Z	S101	0.1000E 01	0.3333E-01	0.3333E-01	20	2	K2
				14	U	S103	0.6000E-01	0.5000E 00	0.3000E-01	21	25	-P
20	A	0.1500E 02	INT		NREF	AZ	0.1000E 02	0.6667E-01	0.6667E 00	22	1	AC
				10	-F	S101	0.1000E 01	0.2000E 01	0.2000E 01	23	20	A
				20	A	K3	0.2000E 01	0.1000E 01	0.2000E 01	24		
21	TIME	0.1000E 02	INT		NREF	S100	0.0	0.0	0.0		15	L
					NREF	S101	0.1000E 01	0.1000E 00	0.1000E 00	25		
22	-K2	0.1000E 00	INV	2	K2	S101	0.1000E 01	0.1000E 01	0.1000E 01		3	-Z
23	-L	0.1000E 01	INV	15	L	S101	0.1000E 01	0.1000E 01	0.1000E 01		16	-DR
24	G	0.1000E 01	INV	18	-G	S101	0.1000E 01	0.1000E 01	0.1000E 01		3	-Z
25	-P	0.3000E 01	INV	19	P	S101	0.1000E 01	0.1000E 01	0.1000E 01		1	AC
											2	K2
											4	-F1
											5	-F2
											6	L1
											8	L2

NO TIME SCALE FACTOR NEEDED

Figure 7.7

Patching Information for Example 2



Pot. Number	Description	Pot. Number	Description
1	$P_m/(\Lambda_m/\Lambda_{cm})$	14	$1/U_m$
2	$P_m/(\Lambda_m/\Lambda_{cm})$	15	T_{1n}/T_m
3	$(3*(C_1*0+C_2)/K_{2m})/P_m$	16	D_{rm}/S_m
4	$170/F_{1m}$	17	K_1
5	$50*P_m/F_{1m}$	18	K_1*S_m/G_m
6	$250/F_{2m}$	19	Z_m/G_m
7	$83.3*P_m/F_{2m}$	20	Z_m/P_m
8	P_m/P_3	21	$0.06*U_m/P_m$
9	F_{1sm}/F_m	22	$10/\Lambda_m$
10	F_{2sm}/F_m	23	F_m/Λ_m
11	$19/U_{1m}$	24	K_3
12	$6*\Lambda_{cm}/U_{1m}$	25	$1/T_m$
13	U_{1sm}/U_m		

Figure 7.8

Patching Diagram from CATS for Example 2

7.4 Example 3: Chased Target Problem

Consider the problem shown in Figure 7.9, where a target vehicle is moving with a fixed horizontal velocity and falling under gravitational force. The chasing vehicle has a thrust, T, directed along the line joining the chasing and target vehicles. This line makes an angle of θ with the horizontal. The chasing vehicle is guided to point at the target vehicle at all times. The differential equations describing the dynamics of the vehicles are as follows:

$$\ddot{Y}_t = -G$$

$$\dot{X}_t = V_{t0}$$

$$\ddot{X}_c = T\cos\theta$$

$$\ddot{Y}_c = T\sin\theta$$

$$D_x = X_t - X_c$$

$$D_y = Y_t - Y_c$$

$$H = \sqrt{D_x^2 + D_y^2}$$

$$\cos\theta = D_x/H$$

$$\sin\theta = D_y/H$$

with initial conditions

$$X_c(0) = 0$$

$$Y_c(0) = 2200$$

$$X_t(0) = 3000$$

$$Y_t(0) = 2000$$

$$\dot{X}_c(0) = V_{c0}\cos\theta(0)$$

$$\dot{Y}_c(0) = V_{c0}\sin\theta(0)$$

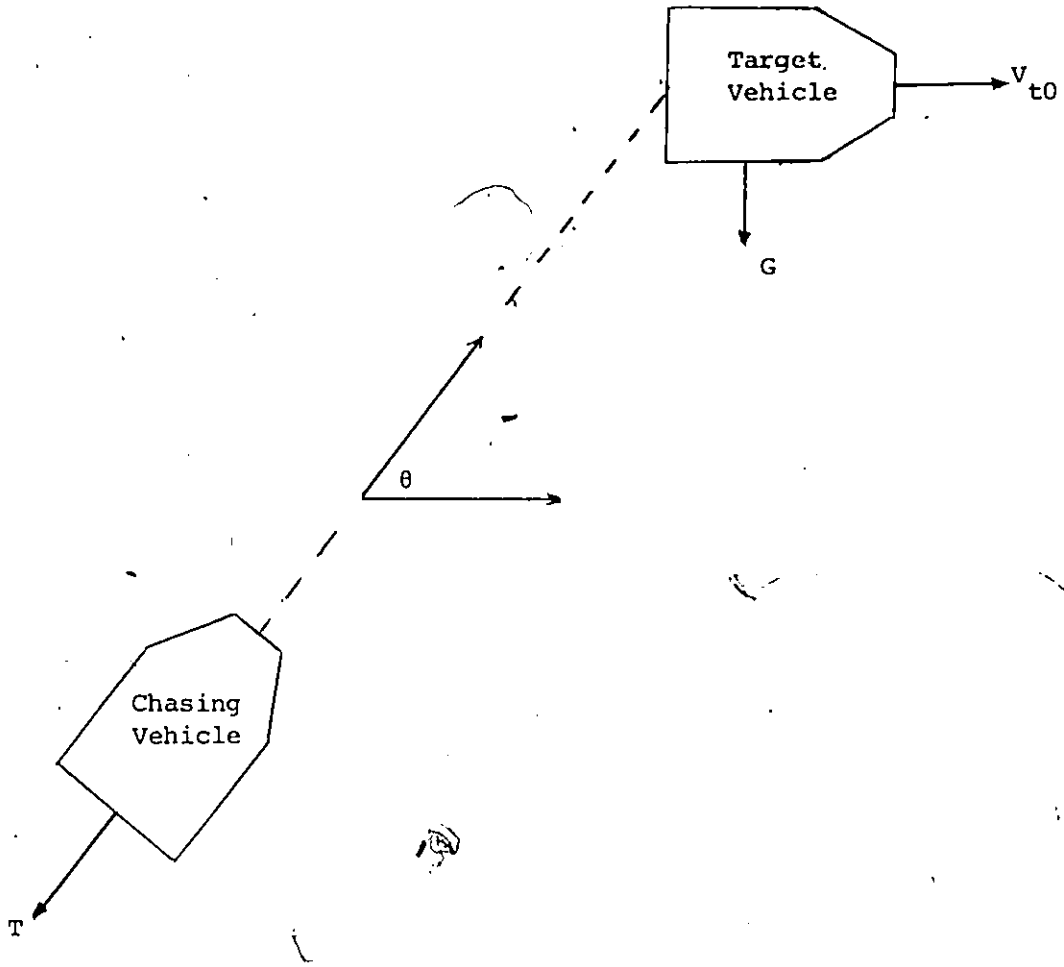


Figure 7.9

Vehicle Configuration for the

Chased Target Problem

where

X_c = horizontal position of chasing vehicle (feet)

Y_c = vertical position of chasing vehicle (feet)

X_t = horizontal position of target vehicle (feet)

Y_t = vertical position of target vehicle (feet)

V_{c0} = initial velocity of chasing vehicle = 100 ft/sec

V_{t0} = horizontal velocity of target vehicle = 400 ft/sec

T = thrust of chasing vehicle = 500 ft·lb/sec

G = gravitational force = 32 lb·ft/sec^2

A CSMP program corresponding to this system is given in Figure 7.10. The resulting scaling and patching information is given in Figure 7.11. The a/h computer diagram drawn from the patching data is shown in Figure 7.12.

```
*          EXAMPLE 3 - CHASED TARGET PROBLEM

PARAM T=500.,G=-32.,VCO=100.,VTO=400.
DX=XT-XC
DY=YT-YC
VXO=VCO*COSTHT
VYO=VCO*SINTHT
COSTHT=DX/H
SINTHT=DY/H
H=(DX**2+DY**2)**0.5
XCD=INTGRL(VXO,T*COSTHT)
YCD=INTGRL(VYO,T*SINTHT)
XC=INTGRL(0.,XCD)
YC=INTGRL(2200.,YCD)
YTD=INTGRL(0.,G)
XT=INTGRL(3000.,VTO)
YT=INTGRL(2000.,YTD)
SIGNL=CMPAR(H,100.)
TIMER  DELT=.01,FINTIM=5.
END
```

Figure 7.10

CSMP program for Example 3

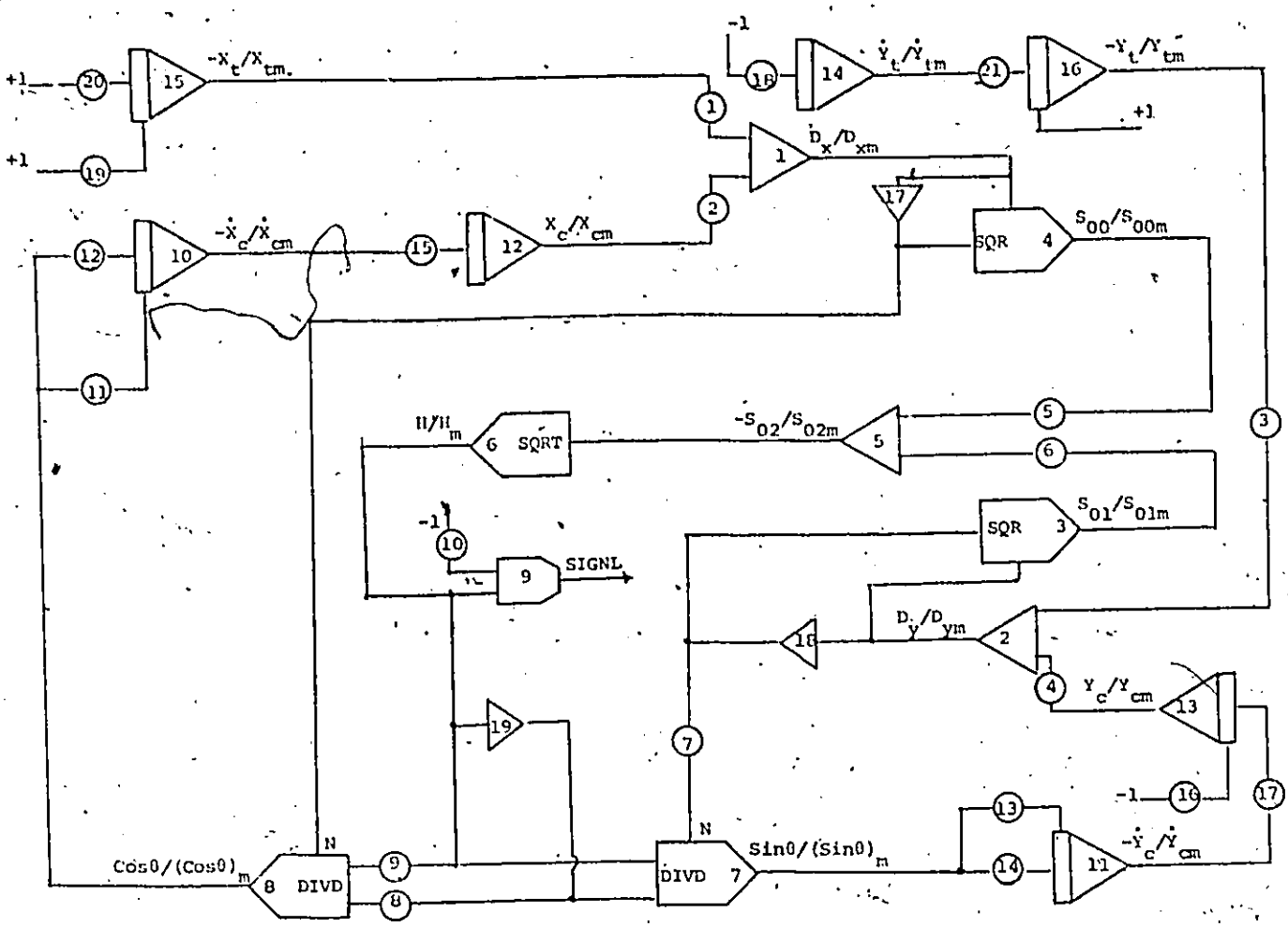
TABLE FOR ANALOG CIRCUIT

OUTPUT COMP #	OUTPUT LABEL	MAXIMUM VALUE	DEVICE TYPE	INPUT(S) COMP #	INPUT(S) LABEL	PROBLEM PARM	GAIN VALUE	SCALE FACTOR	NET INPUT GAIN	POT #	TARGET(S) COMP #	TARGET(S) LABEL
1	DX	0.3500E 04	SUMMER	15	-XT	S101	0.1000E 01	0.1429E 01	0.1429E 01	1	4	S00
				12	XC	S101	0.1000E 01	0.1714E 01	0.1714E 01	2	17	-DX
2	DY	0.2000E 03	SUMMER	16	-YT	S101	0.1000E 01	0.1000E 02	0.1000E 02	3	3	SC1
				13	YC	S101	0.1000E 01	0.1250E 02	0.1250E 02	4	18	-DY
3	S01	0.4000E 05	SQR	18	-DY	S101	0.1000E 01	0.1000E 01	0.1000E 01		5	-SQ2
				2	DY	S101	0.1000E 01	0.1000E 01	0.1000E 01			
4	S00	0.1225E 08	SQR	17	-DX	S101	0.1000E 01	0.1000E 01	0.1000E 01		5	-SQ2
				1	DX	S101	0.1000E 01	0.1000E 01	0.1000E 01			
5	-S02	0.1000E 08	SUMMER	4	S00	S101	0.1000E 01	0.1225E 01	0.1225E 01	5	6	H
				3	S01	S101	0.1000E 01	0.4000E-02	0.4000E-02	6		
6	H	0.3162E 04	SQRT	5	-S02	S101	0.1000E 01	0.1000E 01	0.1000E 01		7	SINTHT
											8	COSTHT
											9	SIGNL
											15	-H
7	SINTHT	0.1000E 01	DIVD	18	-DY	S101	0.1000E 01	0.6325E-01	0.6325E-01	7	11	-YCD
				19	-H	S101	0.1000E 01	0.1000E 01	0.1000E 01		11	-YCD
				6	H	S101	0.1000E 01	0.1000E 01	0.1000E 01			
8	COSTHT	0.1000E 01	DIVD	17	-DX	S101	0.1000E 01	0.1000E 01	0.1000E 01		10	-XCD
				19	-H	S101	0.1000E 01	0.9035E 00	0.9035E 00	8	10	-XCD
				6	H	S101	0.1000E 01	0.9035E 00	0.9035E 00	9		
9	SIGNL	0.1000E 01	COMP	6	H	S101	0.1000E 01	0.1000E 01	0.1000E 01			
					NREF	S107	0.1000E 03	0.3162E-03	0.3162E-01	10		
10	-XCD	0.2500E 04	INT	8	COSTHT	VCO	0.1000E 03	0.4000E-03	0.4000E-01	11	12	XC
				8	COSTHT	T	0.5000E 03	0.4000E-03	0.2000E 00	12		
11	-YCD	0.4000E 03	INT	7	SINTHT	VCO	0.1000E 03	0.2500E-02	0.2500E 00	13	13	YC
				7	SINTHT	T	0.5000E 03	0.2500E-02	0.1250E 01	14		
12	XC	0.6000E 04	INT		NREF	S100	0.0	0.0	0.0		1	DX
				10	-XCD	S101	0.1000E 01	0.4167E 00	0.4167E 00	15		
13	YC	0.2500E 04	INT		NREF	S104	0.2200E 04	0.4000E-03	0.8800E 00	16	2	DY
				11	-YCD	S101	0.1000E 01	0.1600E 00	0.1600E 00	17		
14	YTD	0.2000E 03	INT		NREF	S100	0.0	0.0	0.0		16	-YT
					NREF	-G	0.3200E 02	0.5000E-02	0.1600E 00	18		
15	-XT	0.5000E 04	INT		PREF	S105	0.3000E 04	0.2000E-03	0.6000E 00	19	1	DX
					PREF	VTO	0.4000E 03	0.2000E-03	0.8000E-01	20		
16	-YT	0.2000E 04	INT		PREF	S106	0.2000E 04	0.5000E-03	0.1000E 01		2	DY
				14	YTD	S101	0.1000E 01	0.1000E 00	0.1000E 00	21		
17	-DX	0.3500E 04	INV	1	DX	S101	0.1000E 01	0.1000E 01	0.1000E 01		4	S00
											8	COSTHT
18	-DY	0.2000E 03	INV	2	DY	S101	0.1000E 01	0.1000E 01	0.1000E 01		3	SC1
											7	SINTHT
19	-H	0.3162E 04	INV	6	H	S101	0.1000E 01	0.1000E 01	0.1000E 01		7	SINTHT
											8	COSTHT

NO TIME SCALE FACTOR NEEDED

Figure 7.11

Patching Information for Example 3



Pot. Number	Description	Pot. Number	Description
1	X_{tm}/D_{xm}	12	$T/\dot{X}_{cm}$
2	X_{cm}/D_{xm}	13	$V_{co}/\dot{Y}_{cm}$
3	Y_{tm}/D_{ym}	14	$T/\dot{Y}_{cm}$
4	Y_{cm}/D_{ym}	15	$\dot{X}_{cm}/X_{cm}$
5	S_{00m}/S_{02m}	16	$2200/Y_{cm}$
6	S_{01m}/S_{02m}	17	$\dot{Y}_{cm}/Y_{cm}$
7	D_{ym}/H_m	18	$G/\dot{Y}_{tm}$
8	H_m/D_{xm}	19	$3000/X_{tm}$
9	H_m/D_{xm}	20	V_{to}/X_{tm}
10	$100/H_m$	21	$\dot{Y}_{tm}/Y_{tm}$
11	$V_{co}/\dot{X}_{cm}$		

Figure 7.12

Patching Diagram from CATS for Example 3

CHAPTER 8

Conclusions and Further Work

There is, without question, sufficient need for such software packages as CATS to justify their development. CATS enables the user to specify, in a high level language (namely CSMP) the simulation problem being programmed for an a/h computer. Translation of this source language definition into an a/h computer program specification is entirely automatic. Thus the a/h user is relieved, to a large extent from the often tedious task of program preparation.

The CATS program frees the user from most of the restrictions imposed by the earlier PATCH software. In particular, CATS can process model structure statements of arbitrary complexity. However some of the features available in CSMP (e.g. MACROS, PROCEDURES, NOSORT, etc.) have not been considered at this stage, and future work should be directed towards appropriately accommodating these features.

A novel feature in CATS is concerned with determining the minimum allocation of inverters in the a/h computer program. This is achieved by judicious selection of the polarities with which the problem variables are generated. This feature is particularly important in large scale simulation problem.

To handle the inverter minimization problem, CATS program uses the procedure described in [7]. Computation experience has however suggested that the procedure is computationally time consuming. Hence this work should be investigated further to deter-

mine if efficiencies can be achieved.

CATS requires a large digital computer for execution. It is written in FORTRAN for the IBM 360/65, although it is possible to transfer the program to any similar machine.

At this preliminary stage of development it should be noted that CATS cannot be assured of being totally 'bug-free'. The results obtained for the examples considered are, of course, correct but further examples need to be tried and the results evaluated.

A sufficient foundation has been established in this thesis to further develop CATS as a powerful and comprehensive tool to aid a/h computer programmers.

APPENDIX 1

Structure and Organization of CATS

A1.1 Introduction

The program is structured to accommodate a simulation problem of arbitrary size. This is achieved by dynamically allocating the memory size of the machine. The memory requirement for the job is based on a parameter list inputted by the user. The entries in this list are determined from the size of the CSMP program.

The global flowchart of the CATS program is given in Figure A1.1.

A1.2 Dimensioning the Arrays

Except for the main program, CONTRL, all the FORTRAN subroutines of CATS are stored as an object module in the system. The sizes of all the arrays contained in these subroutines are allocated at run time. The proper sizes of these arrays are established in the main program CONTRL which is created during the execution of CATS.

Within the system is a skeleton program for CONTRL, as shown in Figure A1.2. All the $ symbols appearing within the array declarations are replaced subsequently by numeric constants which define the sizes of the arrays for the particular job. This operation is performed by the program DIMEN whose structure is shown in Figure A1.3.

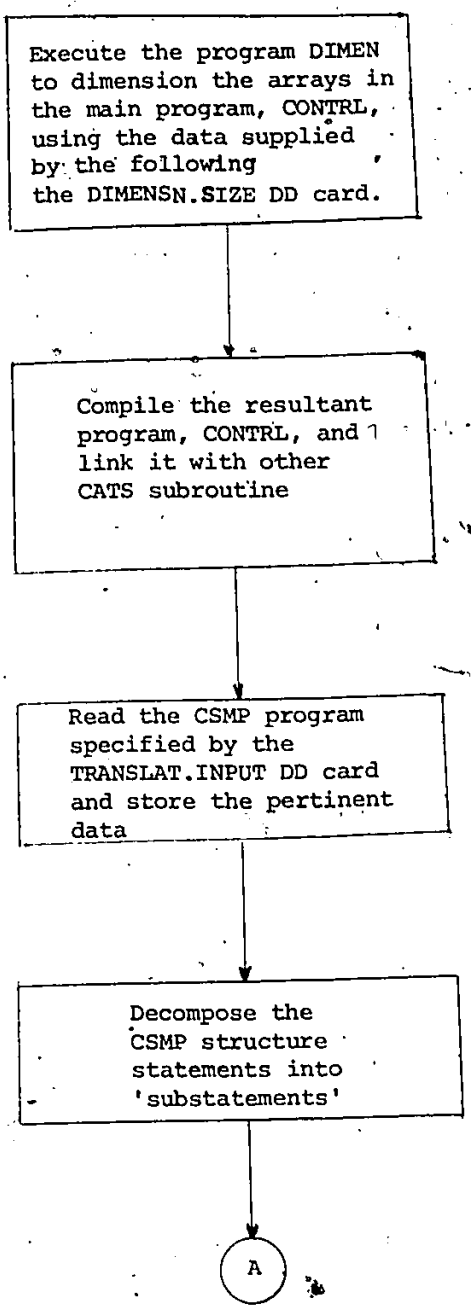


Figure A1.1

Flow Diagram of CATS Execution Model

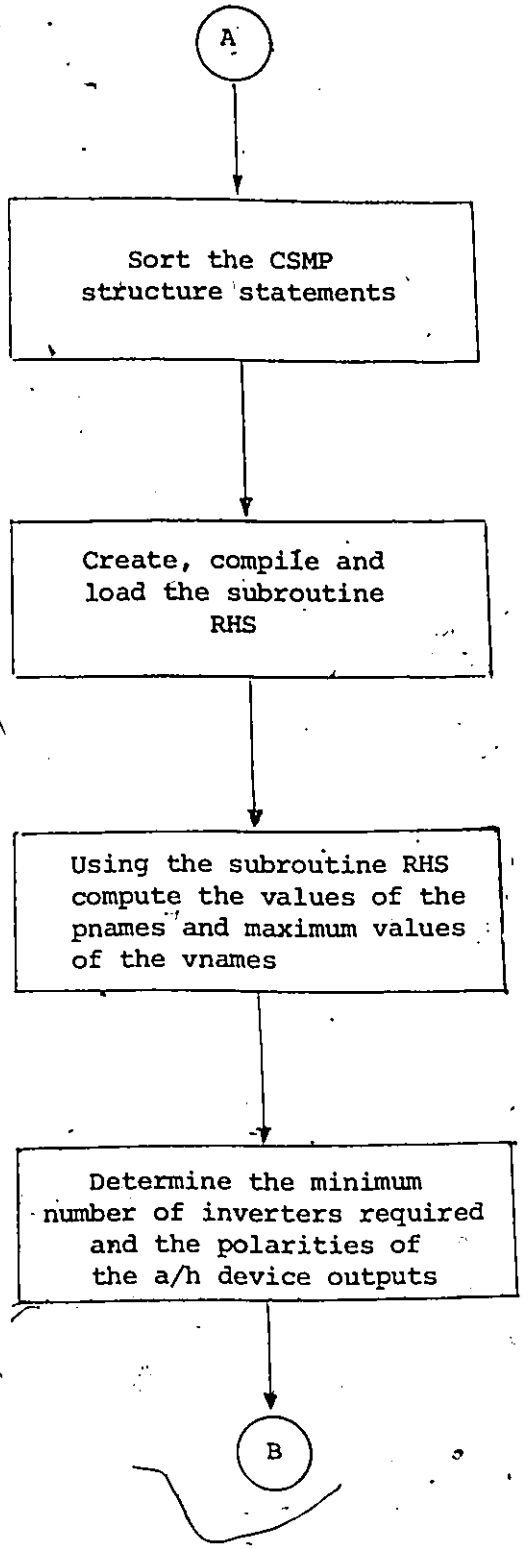


Figure A1.1 (cont'd)

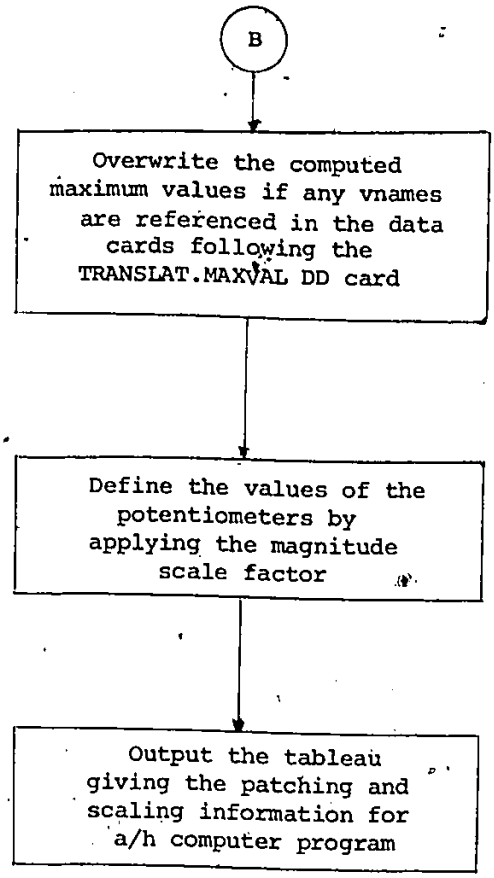


Figure A1.1 (cont'd)

```
      IMPLICIT INTEGER*2 (A-Z)
      REAL*8 PNAME($$$$)/'S100','S101','S102',$$$$*'        '/,
     1VNAME($$$$)/'TIME','FINTIM',$$$$*'        '/,MATRIX($$$$,2)
      REAL*4 PVAL($$$$)/0.0,1.0,2.0,$$$$*-0.539760E+70/,
     1VMAX($$$$)/$$$$*-0.539760E+70/,GAIN($$$$)/$$$$*1.0/,Z($$$$),
     1F($$$$),S1000X($$$$),S1000F($$$$),S1000V($$$$),DXZ($$$$),
     1TESTX($$$$),TESTV($$$$),XNM3($$$$),XNM2($$$$),XNM1($$$$),
     1XN($$$$),XNP1($$$$),CN($$$$),CNP1($$$$),PN($$$$),PNP1($$$$),
     1MOD($$$$),FMOD($$$$),FNM2($$$$),FNM1($$$$),FN($$$$),FNP1($$$$),
     1Q1($$$$),F1($$$$)
      INTEGER SZNAME,SZSC,SZTEMP,SZPSTR,SZDPDR,ADR,SZMATX
      INTEGER*2 SC($$$$)/$$$$*' '/,TEMP(320),CODE(50),POST(50,4),
     1NODTYP($$$$)/$$$$*0/,NODIN($$$$)/$$$$*0/,NODPTR($$$$)/$$$$*0/,
     1NODSRC($$$$)/$$$$*0/,BRGAIN($$$$)/$$$$*2/,LABEL($$$$)/$$$$*0/,
     1NODOUT($$$$)/$$$$*0/,PTRGO($$$$)/$$$$*0/,NODGO($$$$)/$$$$*0/,
     1OPR1(50),OPR2(50),OPR(50)
      LOGICAL*1 A($$$$,$$$$),B($$$$,$$$$),MROW($$$$)/$$$$*.FALSE./,
     1PSTAR($$$$)/$$$$*.FALSE./,MDFLAG($$$$)/$$$$*.TRUE./,ROW($$$$),
     1COL($$$$),D($$$$,$$$$)/$$$$$$*.FALSE./,CHKEMP(3,$$$$),Q($$$$),
     1SZK($$$$),R($$$$),RO($$$$),BIT($$$$),RJ($$$$),SP($$$$,$$$$),
     1SM($$$$,$$$$),SZ($$$$,$$$$),SL($$$$),R1($$$$)
      EQUIVALENCE (MATRIX(1),PNAME(1)),(MATRIX($$$$),VNAME(1)),
     1(SC($$$$),TEMP(1)),(POST(1),CODE(1)),(POST(51),OPR(1)),
     1(POST(101),OPR1(1)),(POST(151),OPR2(1)),(SC(1),S1000X(1)),
     1(SC($$$$),S1000F(1)),(SC($$$$),S1000V(1)),(SC($$$$),DXZ(1)),
     1(SC($$$$),TESTX(1)),(SC($$$$),TESTV(1)),(SC($$$$),XNM3(1)),
     1(SC($$$$),XNM2(1)),(SC($$$$),XNM1(1)),(SC($$$$),XN(1)),
     1(SC($$$$),XNP1(1)),(SC($$$$),CN(1)),(SC($$$$),CNP1(1)),
     1(SC($$$$),PN(1)),(SC($$$$),PNP1(1)),(SC($$$$),MOD(1)),
     1(SC($$$$),FMOD(1)),(SC($$$$),FNM2(1)),(SC($$$$),FNM1(1)),
     1(SC($$$$),FN(1)),(SC($$$$),FNP1(1)),(SC($$$$),Q1(1)),
     1(SC($$$$),F1(1)),(D(1),A(1)),(D($$$$),B(1))
      COMMON/SIZES/SZNAME,SZSC,SZTEMP,SZPSTR,SZDPDR,SZMATX,ADR
      SZNAME=$$$$
      SZSC=$$$$
      SZTEMP=320
      SZPSTR=50
      SZDPDR=$$$$
      SZMATX=$$$$
      ADR=$$$$
      FRELTH=$$$$
      ADR1=0
      PTRFST=1
      PTRD=1
      LAST=0
      CALL BEGIN (SC,TEMP,MATRIX,Z,F,PTRFST,PTRD,FRELTH,ADR1,
     1PNAME,VNAME,PVAL,VMAX, GAIN,NODTYP,NODIN,NODPTR,NODSRC,BRGAIN,
     1MDFLAG,LABEL,NODOUT,PTRGO,NODGO,LAST,CODE,LC,POST,OPR1,OPR2,OPR,
     1A,B,PSTAR,ROW,COL,CHKEMP,Q,SZK,R,RO,BIT,RJ,R1,SP,SM,SZ,SL,D,
     1S1000X,S1000F,S1000V,DXZ,TESTX,TESTV,XNM3,XNM2,XNM1,XN,XNP1,
     1CN,CNP1,PN,PNP1,MOD,FMOD,FNM2,FNM1,FN,FNP1,Q1,F1,MROW)
      STOP
      END
```

Figure A1.2

Skeleton program for CONTRI

```
      INTEGER DUMMY(118)
      INTEGER*2 ITEST/' '/,DOLLAR/'$'/
      LOGICAL*1 VECTOR(4240),SIZE(472),LTEST(2)
      EQUIVALENCE (ITEST,LTEST(1))
      CALL SET899(DUMMY,472)
      READ(5,1)N1,N2,N3,N4,N5
1     FORMAT(5I2)
      J1=N1+3*N2+4*N3+N5
      J2=J1-3
      J3=J1-2
      J4=J1*2
      J5=(4*N3+N4)*5
      J6=4*N3
      J7=(N2+N3*2)*80+320
      J8=J5*2
      J9=J8*J8
      J10=J7-320
      J11=J10+1
      K1=J1+1
      K2=J7-319
      I=8*N3
      K3=1+I
      K4=K3+I
      K5=K4+I
      K6=K5+I
      K7=K6+I
      K8=K7+I
      K9=K8+I
      K10=K9+I
      K11=K10+I
      K12=K11+I
      K13=K12+I
      K14=K13+I
      K15=K14+I
      K16=K15+I
      K17=K16+I
      K18=K17+I
      K19=K18+I
      K20=K19+I
      K21=K20+I
      K22=K21+I
      K23=K22+I
      K24=K23+I
      K25=J6*J6+1
      WRITE(99,2)J1,J2,J1,J3,J4,J1,J2,J1,J1,J4,J4,J5,J5,
     1J6,J5,J5,J6,J6,J5,J5,J6,J6,J5,J6,J5,J5,J6,J6,J5,J5,J6,J6,J6,J6,J6,
     1J5,J7,J7,J1,J1,J1,J1,J1,J1),J1,J5,J5,J5,J5,J1,J1,J1,J1,J1,J1,J5,J5,
     1J6,J6,J5,J6,J6,J5,J5,J5,J5,J5,J5,J5,J5,J8,J8,J9,J5,J5,J6,J6,J6,J8,J6,
     1J6,J6,J6,J6,J6,J5,J6,J6,      K1,K2,K3,K4,K5,K6,K7,K8,K9,K10,K11,
     1K12,K13,K14,K15,K16,K17,K18,K19,K20,K21,K22,K23,K24,K25,
     1J1,J7,J9,J6,J11,J10
2     FORMAT(70I4,I6,46I4)
      READ(99,3)(SIZE(K),K=1,472)
3     FORMAT(200A1,200A1,72A1)
      READ(4,6)(VECTOR(N),N=1,4240)
5     FORMAT(80A1)
      N=0
      DO 5 K=1,4240
      LTEST(1)=VECTOR(K)
      IF (ITEST.NE.DOLLAR) GO TO 5
      N=N+1
      VECTOR(K)=SIZE(N)
5     CONTINUE
      WRITE(6,7)(VECTOR(K),K=1,4240)
7     FORMAT(80A1)
      STOP
      END
```

Figure A1.3

To initiate the dimensioning operation the user inputs five parameters via the DIMENSN.SIZE DD * card. These parameters, whose input format is '5I2', have the following significance:

N1 - approximate number of pnames defined in the PARM, INCON and CONST statements of the CSMP program.

N2 - approximate number of structure statements within the Initial segment of the CSMP program.

N3 - approximate number of structure statements within the Dynamic segment of the CSMP program.

N4 - approximate number of FUNCTION statements in the CSMP program.

N5 - an additive factor for increasing the sizes of the arrays.

As an illustration, for the CSMP program of Example 3 - 'Chased Target Problem', the following input parameter list was used:

N1 = 10, N2 = 0, N3 = 12, N4 = 0, N5 = 0.

The resulting CONTRL program is shown in Figure A1.4.

A1.3 Execution of CATS

The resulting program, CONTRL, is then compiled and linked with the remaining CATS subroutines to form the module which performs the a/h translation operation. The processing operation is initiated by a call to the subroutine BEGIN in the CONTRL program. Subroutine BEGIN, in turn, calls all the necessary subroutines to process the CSMP program.

The CSMP program is read in as data which is specified by the TRANSLAT.SIZE DD * card in the procedure.

There may be circumstances where the user does not wish to

```
FORTRAN IV G LEVEL  21                CONTRL          DATE = 76219        21/38/11          PAGE 0001

0001                  IMPLICIT INTEGER*2 (A-Z)
0002                  REAL*8 PNAME(  58)/'S100','S101','S102',  55*'        '/,
                     1VNAME(  58)/'TIME','FINTIM',  56*'        '/,MATRIX( 116,2)
0003                  REAL*4 PVAL(  58)/0.0,1.0,2.0,  55*-0.539760E+70/,
                     1VMAX(  58)/  58*-0.539760E+70/,GAIN( 116)/ 116*1.0/,Z( 240),
                     1F( 240),S1000X(  48),S1000F(  48),S1000V(  48),DXZ(  48),
                     1TESTX(  48),TESTV(  48),XNM3(  48),XNM2(  48),XNM1(  48),
                     1XN(  48),XNP1(  48),CN(  48),CNP1(  48),PN(  48),PNP1(  48),
                     1MOD(  48),FMOD(  48),FNM2(  48),FNM1(  48),FN(  48),FNP1(  48),
                     101(  48),F1(  48)
0004                  INTEGER SZNAME,SZSC,SZTEMP,SZPSTR,SZDPDR,ADR,SZMATX
0005                  INTEGER*2 SC(2240)/2240*' '/,TEMP(320),CODE(50),POST(50,4),
                     1NODTYP(  58)/  58*0/,NODIN(  58)/  58*0/,NODPTR(  58)/  58*0/,
                     1NODSRC( 240)/ 240*0/,BRGAIN( 240)/ 240*2/,LABEL(  58)/  58*0/,
                     1NODOUT(  58)/  58*0/,PTRGO(  58)/  58*0/,NODGO( 240)/ 240*0/,
                     1OPR1(50),OPR2(50),OPR(50)
0006                  LOGICAL*1 A(  48,  48),B(  48,  48),MROW(  48)/  48*.FALSE./,
                     1PSTAR(  48)/  48*.FALSE./,MDFLAG( 240)/ 240*.TRUE./,ROW(  48),
                     1COL(  48),D(  96,  96)/  9216*.FALSE./,CHKEMP(3,  48),Q(  48),
                     1SZK(  48),R(  48),RO(  48),BIT(  96),RJ(  48),SP(  48,  48),
                     1SM(  48,  48),SZ(  48,  48),SL(  48),R1(  48)
0007                  EQUIVALENCE (MATRIX(1),PNAME(1)),(MATRIX(  59),VNAME(1)),
                     1(SC(1921),TEMP(1)),(POST(1),CODE(1)),(POST(51),OPR(1)),
                     1(POST(101),OPR1(1)),(POST(151),OPR2(1)),(SC(1),S1000X(1)),
                     1(SC(  97),S1000F(1)),(SC( 193),S1000V(1)),(SC( 289),DXZ(1)),
                     1(SC( 385),TESTX(1)),(SC( 481),TESTV(1)),(SC( 577),XNM3(1)),
                     1(SC( 673),XNM2(1)),(SC( 769),XNM1(1)),(SC( 865),XN(1)),
                     1(SC( 961),XNP1(1)),(SC(1057),CN(1)),(SC(1153),CNP1(1)),
                     1(SC(1249),PN(1)),(SC(1345),PNP1(1)),(SC(1441),MOD(1)),
                     1(SC(1537),FMOD(1)),(SC(1633),FNM2(1)),(SC(1729),FNM1(1)),
                     1(SC(1825),FN(1)),(SC(1921),FNP1(1)),(SC(2017),Q1(1)),
                     1(SC(2113),F1(1)),(D(1),A(1)),(D(2305),B(1))
0008                  COMMON/SIZES/SZNAME,SZSC,SZTEMP,SZPSTR,SZDPDR,SZMATX,ADR
0009                  SZNAME=  58
0010                  SZSC=2240
0011                  SZTEMP=320
0012                  SZPSTR=50
0013                  SZDPDR=  96
0014                  SZMATX=  48
0015                  ADR=1921
0016                  FRELTH=1920
0017                  ADR1=0
0018                  PTRFST=1
0019                  PTRD=1
0020                  LAST=0
0021                  CALL BEGIN (SC,TEMP,MATRIX,Z,F,PTRFST,PTRD,FRELTH,ADR1,
                     1PNAME,VNAME,PVAL,VMAX, GAIN,NODTYP,NODIN,NODPTR,NODSRC,BRGAIN,
                     1MDFLAG,LABEL,NODOUT,PTRGO,NODGO,LAST,CODE,LC,POST,OPR1,OPR2,OPR,
                     1A,B,PSTAR,ROW,COL,CHKEMP,Q,SZK,R,RO,BIT,RJ,R1,SP,SM,SZ,SL,D,
                     1S1000X,S1000F,S1000V,DXZ,TESTX,TESTV,XNM3,XNM2,XNM1,XN,XNP1,
                     1CN,CNP1,PN,PNP1,MOD,FMOD,FNM2,FNM1,FN,FNP1,Q1,F1,MROW)
0022                  STOP
0023                  END
```

Figure A1.4

Main program CONTRL for Example 3

use the maximum values computed by CATS for the problem vnames. An option to permit this is provided. The format for overwriting the maximum values depends on the knowledge of the structure of the a/h circuit generated by CATS and hence it is not feasible to use this option on the first run. As an illustration we consider the Target Chase Problem of Example 3.

Suppose the user wishes to redefine the maximum values of DX, DY, YCD, and XC on a subsequent run of the problem. The following four cards, one for each vname, are inputted after the TRANSLAT.MAVLAL DD * card.

```
 1      .5000E04
 2      .5000E04
11      .5000E03
12      .1000E05
```

On each card the format is 'I2,3X,E11.4'. The ordering of these cards is irrelevant since the component number (the first integer) defines the vname.

A1.4 JCL for Executing CATS Job

The CATS job is divided into the following four distinct steps:

1) DIMENSN - creation of CONTRL program
2) FORT - compilation of CONTRL program
3) LKED - linkage of CONTRL program
4) TRANSLAT- execution of CONTRL program

The flowchart for the catalogued procedure is given in Figure A1.5. The JCL required for the execution of CATS job is shown below.

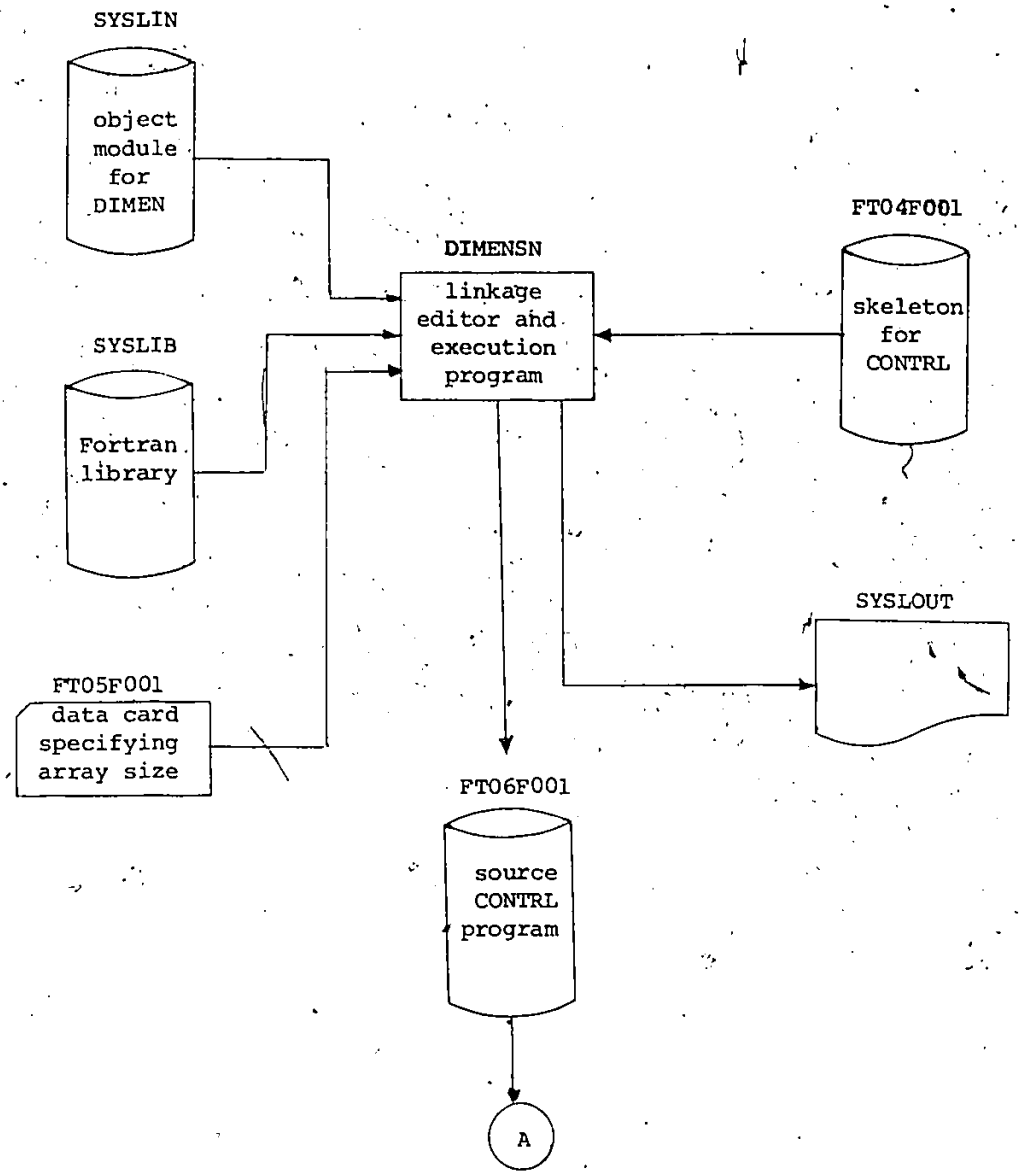


Figure A1.5

Functional Structure of CATS

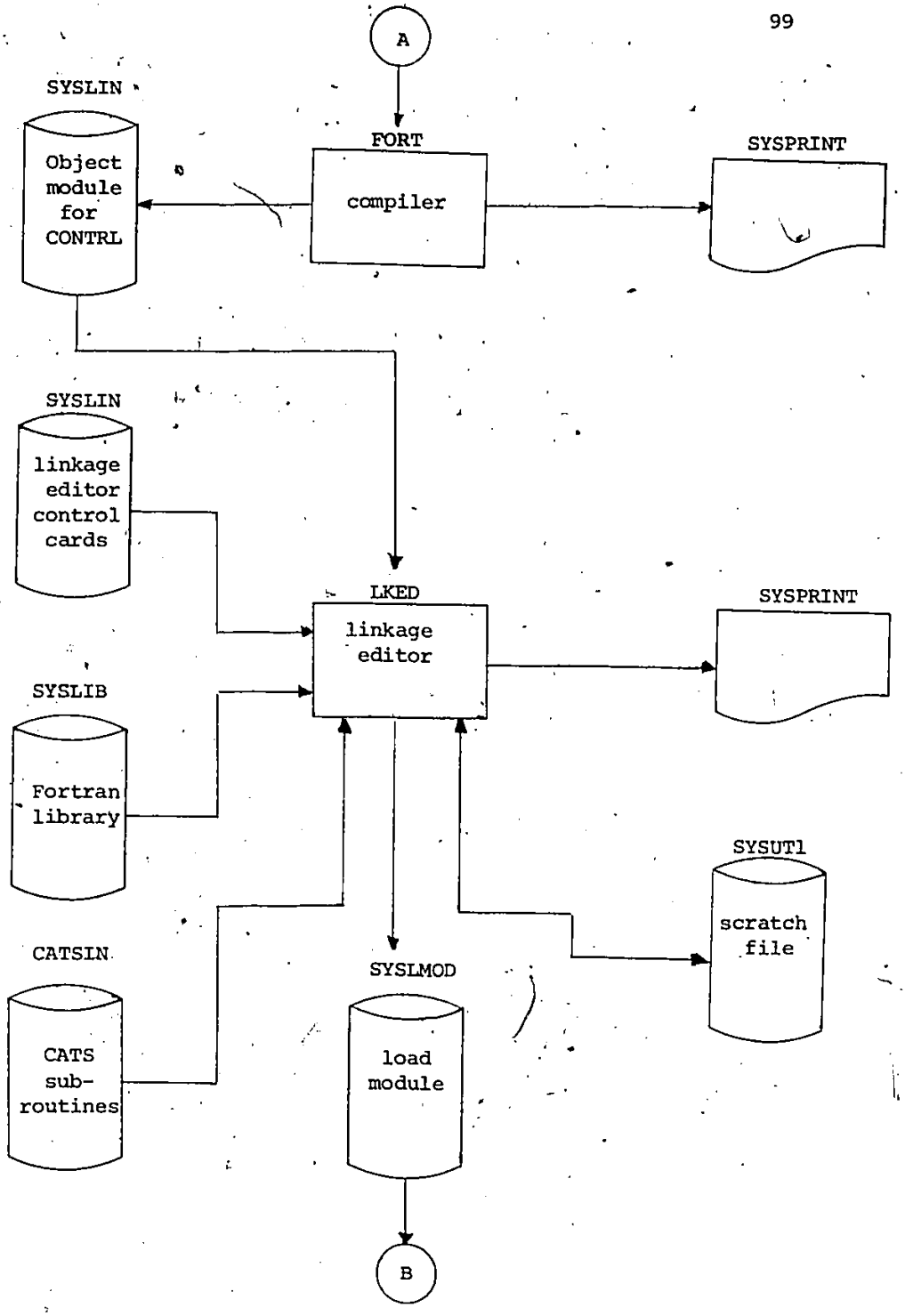


Figure A1.5 (cont'd)

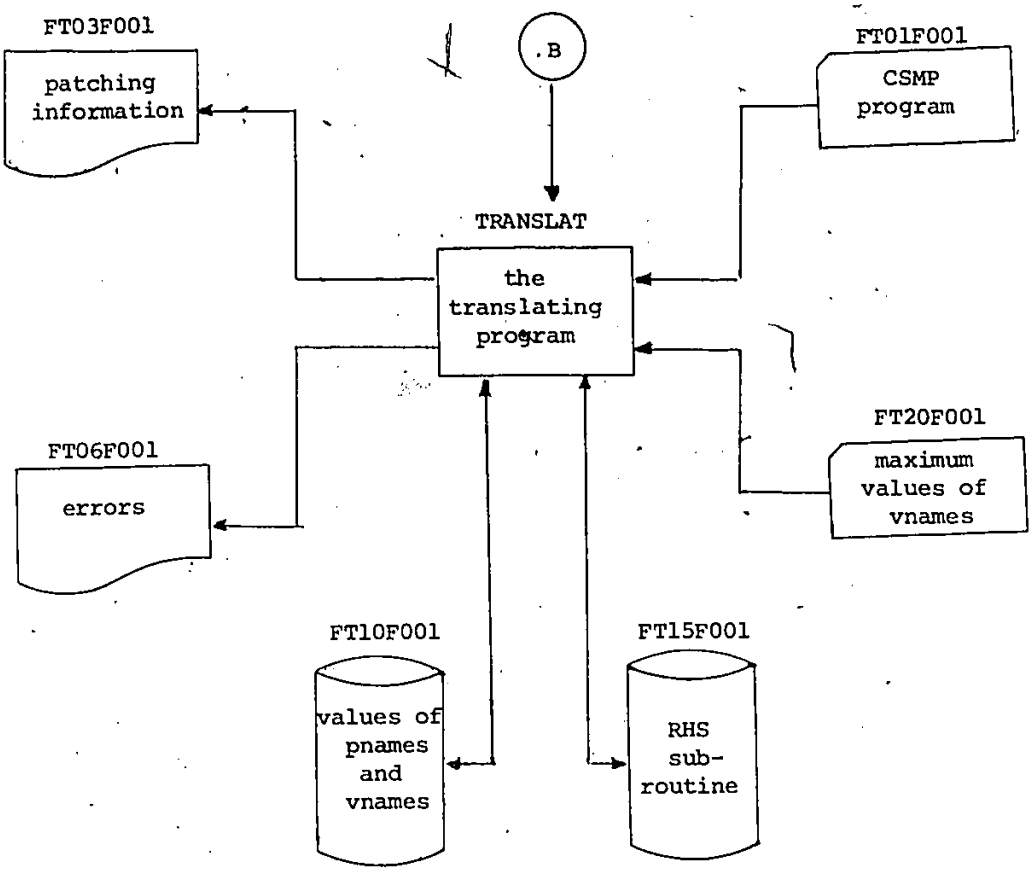


Figure A1.5 (cont'd)

```
// EXEC CATS,CATS.REG=XXX,CATSTIM=XXX
//DIMENSN.SIZE DD *
{a parameter-list card defining an
approximate size of CSMP program
to dimension the arrays}
//TRANSLAT.INPUT DD *
{source CSMP program defining
the simulation problem}
//TRANSLAT.MAXVAL DD *
{optional cards to alter the
maximum values of vnames
during the run}
/*
```

Note that the default values for the CATSREG and CATSTIM parameters are 200K and 2 minutes respectively. To overwrite the CATSREG parameter the value should be greater than 200K. This is due to the need for executing the FORTRAN compiler called during the TRANSLAT step.

APPENDIX 2

Subroutines associated with CATS

This appendix gives a brief description of the subroutines in CATS (Table A2.1) and summarizes the interdependencies of these subroutines (Tables A2.2 and A2.3).

Subroutine	Description
ACCESS	determines the maximum values of vnames
ADJUST	deletes a statement from SC
AMATX	creates dependency matrix for the minimization operation
APPLY	applies a policy to the modified dependency matrix during the inverter minimization
BEGIN	contains calls to subroutines to translate the CSMP statements
BMATX	determines the rows and columns of the dependency matrix which are to be deleted to produce modified dependency matrix
BUILD	build a character string statement from the numeric codes
CMPR	compares the label of a CSMP statement to examine the label of a CSMP statement to determine its validity
CMPXPL	decomposes the CSMP macro CMPXPL into equivalent INTGRL form
CNTCRD	determines if a CSMP statement is continued on the following card
COMBTN	selects the combination of subsets to determine an applicable policy for the inverter minimization
COMPT	selects the subsets that will subsequently be used to determine a policy
CONTRL	main program created, within CATS execution, for initializing the dimension of the arrays.
CRTEQV	creates FORTRAN EQUIVALENCE statements for the RHS subroutine
CRTIOS	creates FORTRAN READ/WRITE statements for the RHS subroutine
CRTRHS	creates the subroutine RHS
CRTSET	creates the sets to be used for the inverter minimization operator

Table A2.1

Names and Description of CATS subroutines

Subroutine	Description
CRTSTM	creates the FORTRAN assignment statements for the RHS subroutine
DECOMP	decomposes the CSMP macros CMPXPL, LEDLAG and REALPL
DIMEN	creates the main program CONTRL
DYNA	decomposes the source statements into substatements
ERRM	prints out any error message generated by CATS
EXPON	processes an expression having negative exponent
FILLSC	stores the CSMP structure statements in SC
FORASM	compiles and loads the subroutine RHS (assembler routine)
GENP	generates system pnames
GENV	generates system vnames
GETNB	extracts numeric constants from the source statements
GETINV	processes any source statement that corresponds to an inverter
HASH	locates a natural position for an identifier in PNAME/VNAME using a hashing technique
INPUT	processes a statement to determine inputs for a/h devices
INPUT1	processes a statement corresponding to an a/h device requiring only one input
INPUT2	processes a statement corresponding to an a/h device requiring only two inputs
INPUT3	processes a statement corresponding to an a/h device requiring more than two inputs
INSERT	inserts a statement into SC
INVERT	generates inverters for a/h computer, as determined via the minimization operation
LABEL	generates system names for substatements

Table A2.1 (cont'd)

Subroutine	Description
LEDLAG	decomposes the CSMP macro LEDLAG into an equivalent INTGRL form
LEXY	generates numeric codes for the source statements
LOCATE	locates positions of identifiers in the PNAME/VNAME vectors
LOGIC	determines the cost index of a policy applied to the modified dependency matrix
MAGSCL	computes the magnitude scale factor for pot settings
MAXVAL	computes the values of pnames and maximum values of vnames
MINI	minimizes the number of inverters for a/h computer program
MOVE	moves any misplaced Initial statement in the Dynamic segment into the Initial segment
NAMID	extracts the identifiers in the source CSMP program
NEW$A	determines the polarities of the outputs of the a/h devices
NEWSTM	generates new substatements from the source statements and stores them in SC
NONLIN	updates the inputs to non-linear devices
OUTPUT	prints out the CATS output
OUTPT1	prints out the result of the minimization procedure
PATCH	updates the analog tableau after inserting inverters
PCM	predictor-corrector method to solve differential equations
PROCES	processes the data, control and translation statements of the CSMP program
READIN	reads and processes the input data which is the CSMP program

Table A2.1 (cont'd)

Subroutine	Description
REALPL	decomposes the CSMP macro REALPL into an equivalent INTGRL form
REDABS	decomposes the 'ABS' function into equivalent sub-statements, implementable on a/h computer
REDSTM	decomposes the source expression into subexpressions
RHS	executes the assignment statements of the CSMP program
SET$S	extracts a particular set during the inverter minimization operation of the modified dependency matrix
SORT	initializes the various arrays for sorting operation
SORT1	sorts the source statements into a natural order
SORT2	selects and repositions each source statement into a natural order
STARTR	a Runge-Kutta technique used to generate starting values for the PCM routine
STMTYP	determines and stores the source structure statements in SC
STRCUR	builds up the tableau for the a/h program from the source statements
SUBSET	determines if a set S_1 is a subset of S_2
SWITCH	repositions a source statement after a specified statement
TABLE	determine if a device output needs to be included in the magnitude scaling operation
TRANS	translates the storage format of an identifier
TYPE	determines the type of a/h device corresponding to the source statement

Table A2.1 (cont'd)

Subroutine	Calls to
ACCESS	-
ADJUST	-
AMATX	-
APPLY	- CRTSET
BEGIN	- AMATX, BMATX, CRTRHS, DECOMP, DYNA, FORASM, GETINV, INCOND, INVERT, MAGSCL, MAXVAL, MINI, MOVE, OUTPUT, PATCH, READIN, SORT, STMTYP, STRCUR
BUILD	- TRANS
BMATX	-
CMPR	- NAMID
CMPXPL	- ADJUST, GENV, INSERT
CNTCRD	-
COMBTN	-
COMPT	- SET$S, SUBSET
CONTRL	- BEGIN
CRTEQV	- TRANS
CRTIOS	- TRANS
CRTRHS	- CRTEQV, CRTIOS, CRTSTM
CRTSET	-
CRTSTM	- BUILD
DECOMP	- CMPXPL, LEDLAG, REALPL
DIMEN	-
DYNA	- ADJUST, EXPON, LABEL, NEWSTM, REDSTM
ERRM	-
EXPON	-

Table A2.2

Subroutine Calls to

Subroutine		Calls to
FILLSC	-	CNTCRD, INSERT
FORASM	-	IEYFORT, IEWL (FORTRAN compiler and loader)
GENP	-	ERRM, LOCATE, TRANS
GENV	-	ERRM, LOCATE, TRANS
GETINV	-	GENP
GETNB	-	SETB99
HASH	-	
INCOND	-	
INPUT	-	GENP, INSERT, SWITCH
INPUT1	-	INPUT
INPUT2	-	INPUT
INPUT3	-	INPUT
INSERT	-	ERRM
INVERT	-	ERRM, NONLIN
LABEL	-	GENP, GENV
LEDLAG	-	ADJUST, GENV, INSERT
LEXY	-	ERRM, GENP, GETNB, LOCATE, NAMID
LOCATE	-	ERRM, HASH
LOGIC	-	APPLY, SET$S, SUBSET
MAGSCL	-	TABLE
MAXVAL	-	ACCESS, PCM, RHS, SETB99
MINI	-	APPLY, COMBTN, COMPT, LOGIC, OUTPT1, SET$S, SUBSET
MOVE	-	LOCATE
NAMID	-	ERRM, TRANS
NEW$A	-	

Table A2.2 (cont'd)

Subroutine	Calls to
NEWSTM	INSERT, REDABS, TYPE
NONLIN	-
OUTPUT	-
OUTPT1	APPLY
PATCH	-
PCM	ACCESS, RHS, STARTR
PROCES	CNTCRD, GETNR, LOCATE, NAMID
READIN	CMPR, CNTCRD, FILLSC, LOCATE, NAMID, PROCES
REALPL	ADJUST, INSERT
REDABS	GENV, INSERT
REDSTM	-
RHS	CSMP functions
SET$S	-
SORT	ERRM, SORT1, SORT2
SORT1	-
SORT2	SWITCH
STARTR	RHS
STMTYP	INSERT, LEXY
STRCUR	GENP, INPUT1, INPUT2, INPUT3, INSERT, SWITCH
SUBSET	-
SWITCH	-
TABLE	-
TRANS	-
TYPE	-

Table A2.2 (cont'd)

Subroutine		Called from
ACCESS	-	MAXVAL, PCM
ADJUST	-	CMPXPL, DYNA, LEDLAG, REALPL
AMATX	-	BEGIN
APPLY	-	LOGIC, MINI, OUTPT1
BEGIN	-	CONTROL
BMATX	-	BEGIN
BUILD	-	CRTSTM
CMPR	-	READIN
CMPXPL	-	DECOMP
CNTCRD	-	FILLSC, PROCES, READIN
COMBTN	-	MINI
COMPT	-	MINI
CONTRL	-	
CRTEQV	-	CRTRHS
CRTEOS	-	CRTRHS
CRTRHS	-	BEGIN
CRTSET	-	APPLY
CRTSTM	-	CRTRHS
DECOMP	-	BEGIN
DIMEN	-	
DYNA	-	BEGIN
ERRM	-	GENP, GENV, INSERT, INVERT, LEXY, LOCATE NAMID, SORT
EXPON	-	DYNA
FILLSC	-	READIN

Table A2.3

Subroutine Calls from

Subroutine	Called from
FORASM	- BEGIN
GENP	- GETINV, INPUT, LABEL, LEXY, STRCUR
GENV	- CMPXPL, LABEL, LEDLAG, REDABS
GETINV	- BEGIN
GETNB	- LEXY, PROCES
HASH	- LOCATE
INPUT	- INPUT1, INPUT2, INPUT3
INPUT1	- STRCUR
INPUT2	- STRCUR
INPUT3	- STRCUR
INSERT	- CMPXPL, FILLSC, INPUT, LEDLAG, NEWSTM, REALPL, REDABS, STMTYP, STRCUR
INVERT	- BEGIN
LABEL	- DYNA
LEDLAG	- DECOMP
LEXY	- STMTYP
LOCATE	- GENP, GENV, LEXY, MOVE, PROCES, READIN
LOGIC	- MINI
MAGSCL	- BEGIN
MAXUAL	- BEGIN
MINI	- BEGIN
MOVE	- BEGIN
NAMID	- CMPR, LEXY, PROCES, READIN
NEW$A	- BEGIN
NEWSTM	- DYNA
NONLIN	- INVERT

Table A2.3 (cont'd)

Subroutine		Called from
OUTPUT	-	BEGIN
OUTPT1	-	MINI
PATCH	-	BEGIN
PCM	-	MAXVAL
PROCES	-	READIN
READIN	-	BEGIN
REALPL	-	DECOMP
REDABS	-	NEWSTM
REDSTM	-	DYNA
RHS	-	MAXVAL, PCM, STARTR
SET$S	-	COMPT, LOGIC, MINI
SORT	-	BEGIN
SORT1	-	SORT
SORT2	-	SORT
STARTR	-	PCM
STMTYP	-	BEGIN
STRCUR	-	BEGIN
SUBSET	-	COMPT, LOGIC, MINI
SWITCH	-	INPUT, SORT2, STRCUR
TABLE	-	MAGSCL
TRANS	-	BUILD, CRTEQV, CRTIOS, GENP, GENV, NAMID
TYPE	-	NEWSTM

Table A2.3 (cont'd)

APPENDIX 3

Restrictions on the use of CATS

The users are restricted in the following wasy in the structure of the CSMP program for the problem specification.

1. The following CSMP functions have not been implemented in CATS and therefore should not be used:
   DERIV, DELAY, ZHOLD, IMPL, FCNSW, OUTSW, RST, QNTZR, DEADSP, HSTRSS, STEP, RAMP, IMPULS, PULSE, SINE, GAUSS, RNDGEN, AMAX0, AMAX1, MAX0, MAX1, AMIN0, AMIN1, MIN0, MIN1.
2. The structure statements within the MACRO/ENDMAC, PROCEDURE/ENDPRO and NOSORT blocks are discarded by CATS.
3. A statement may only be continued on as many as 3 cards for a total of 4 cards.
4. Due to the hardware configuration of the a/h machine, the magnitude of the exponent values for an exponential operator is restricted to 0.5 and 2.
5. Certain variable names are reserved for the use of the system and must not appear in a CSMP program. These names are S1000, S10001, S10002, S10003, S10004, S1000F, S1000X, S1000V, S1dd and Sdd, where d is a decimal digit.

Moreover the CSMP program for CATS should be a valid program, observing all the CSMP syntax rules.

REFERENCES

[1] C. Green, H.D. Hoop and A. Debroux, "APACHE - A breakthrough in analog computing", IRE Trans. Electron. Comput., Vol. EC-11, Oct. 1962, pp. 699-706.

[2] H.B. Rigas, "Compilers for analog and hybrid computation", Proc. SCSC, San Diego, California, June 14-June 16, 1972, pp. 310-318.

[3] P.E. Rook, "Hybrid compilation using ACTRAN", Proc. SCSC, San Diego, California, June 14-June 16, 1972, pp. 298-309.

[4] H.H.W. Pitcher, "What can Automatic Programming and Scaling of Equations (APSE) do for the user?", Proc. SCSC, San Diego, California, June 14-June 16, 1972, pp. 1-7.

[5] "System/360 Continuous System Modelling Program User's Manual", IBM Corp. Tech. Publication Dept., Form GH20-0367-3, Jan. 1972.

[6] D.E. Knuth, "A history of writing compilers", in Compiler Techniques (B.W. Pollack, editor), Auberbach 1972, pp. 43-48.

[7] L.G. Birta and R.R. Shah, "The minimization of inverters in analog computer programming", to appear in IEEE Trans. on Computers, (Tech. Rept. TR75-09, Dept. of Comp. Sc., Univ. of Ottawa).

[8] "Hoi Reference Handbook", Electronic Assoc. Inc. publication 00827.0021-1, Sept. 1970.

[9] G.R. Harris, "A model of the body water regulatory system", CoED Trans., Vol. V, No. 2, Feb. 1973.

[10] H.B. Rigas and D.J. Coombs, "PATCH: Analog computer patching from a digital simulation language", IEEE Trans. on Computers, Vol. C-20, No. 10, Oct. 1971, pp. 1140-1146.

[11] H.B. Rigas and D.J. Coombs, "Modifications to a digital simulation program to facilitate automatic patching", Simulation, Oct. 1972, pp. 133-138.

[12] P.E. Rook, "ACTRAN: An analog/hybrid compiler", Proc. AICA/IFIP Conf. on Hybrid Computation, Munich Germany, Aug. 31-Sept. 4, 1970, pp. 758-777.

[13] M.A. Franklin and J.C. Strauss, "Automated programming of analog hybrid computers - a review", Simulation, Jan. 1972, pp. 11-19.

VITA

Name: Ratilal Raichand Shah

Born: Nairobi, Kenya
July 21, 1950

Education: Bachelor of Science (Honours) 1974
Department of Computer Science
University of Ottawa
Ottawa, Ontario
Canada

Enrolled in M.A.Sc. program in Electrical Engineering at University of Ottawa from Sept. 1974 to Sept. 1976.