INFORMATION TO USERS

THIS DISSERTATION HAS BEEN MICROFILMED EXACTLY AS RECEIVED

This copy was produced from a microfiche copy of the original document. The quality of the copy is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

PLEASE NOTE: Some pages may have indistinct print. Filmed as received:

Canadian Theses Division
Cataloguing Branch
National Library of Canada
Ottawa, Canada K1A 0N4

AVIS AUX USAGERS

LA THESE A ETÉ MICROFILMÉE TELLE QUE NOUS L'AVONS RECUE

Cette copie a été faite à partir d'une microfiche du document original. La qualité de la copie dépend grandement de la qualité de la thèse soumise pour le microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

NOTA BENE: La qualité d'impression de certaines pages peut laisser à désirer. Microfilmée telle que nous l'avons reçue.

Division des thèses canadiennes
Direction du catalogage
Bibliothèque nationale du Canada
Ottawa, Canada K1A 0N4
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

© R.R. Shah, Ottawa, Canada, 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

CHAPTER 1 Introduction

1.1 Preliminary Remarks

1.2 Relationship with PATCH

1.3 Summary of the Contents of the Thesis

CHAPTER 2 Data Structures and the Organization of CATS

2.1 Introduction

2.2 Data Structures for CSMP Structure Statements

2.3 Data Structures for Identifiers

2.4 Data Structures for Patching

CHAPTER 3 Initial Processing of the Source Statements

3.1 Introduction

3.2 Processing of the CSMP Program Data

3.3 Coding the Structure Statements

CHAPTER 4 Decomposition of the Source Statements

4.1 Introduction

4.2 Definitions

4.3 Transformation of the CSMP Macros

4.4 The Decomposition Procedure

4.5 Creation of POST Matrix

4.6 Processing an Expression with an Exponential Operator having Negative Exponent
4.7 Substatement Creation from POST Matrix

CHAPTER 5 Sorting

5.1 Introduction

5.2 Natural Order

5.3 The Dependency Matrix

5.4 Associated Parallel Vectors

5.5 The Sorting Operation

5.6 Reordering of the Statements in the SC Vector

CHAPTER 6 Magnitude Scaling and a/h Program Preparation

6.1 Introduction

6.2 Generating Maximum Value Estimates

6.3 Magnitude Scaling Procedure

6.4 Determining the Polarities of the Output Variables

6.5 The a/h Program Output

CHAPTER 7 Examples

7.1 Introduction

7.2 Example 1: Automobile Suspension System

7.3 Example 2: Body Water Regulatory System

7.4 Example 3: Chased Target Problem

CHAPTER 8 Conclusions and Further Work

APPENDIX 1 Structure and Organizing of CATS

A1.1 Introduction

A1.2 Dimensioning the Arrays

A1.3 Execution of CATS

A1.4 JCL for Executing CATS Job
APPENDIX 2. Subroutines associated with CATS
APPENDIX 3. Restrictions on the use of CATS
REFERENCES
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Interactions Amongst the Data Structures</td>
<td>17</td>
</tr>
<tr>
<td>4.1</td>
<td>Flowchart of the Decomposition Procedure</td>
<td>32</td>
</tr>
<tr>
<td>4.2</td>
<td>Snapshots of the Decomposition Algorithm</td>
<td>35</td>
</tr>
<tr>
<td>4.3</td>
<td>The Post Matrix Resulting from the Source Statement</td>
<td>37</td>
</tr>
<tr>
<td>5.1</td>
<td>Flowchart of the Sorting Procedure</td>
<td>50</td>
</tr>
<tr>
<td>5.2</td>
<td>Snapshots of the Sorting Procedure</td>
<td>52</td>
</tr>
<tr>
<td>5.3a</td>
<td>Configuration before Repositioning $X_2$ to Logically follow $X_1$</td>
<td>54</td>
</tr>
<tr>
<td>5.3b</td>
<td>Configuration after Repositioning $X_2$ to Logically follow $X_1$</td>
<td>55</td>
</tr>
<tr>
<td>7.1</td>
<td>Simplified Model of Automobile Suspension System</td>
<td>70</td>
</tr>
<tr>
<td>7.2</td>
<td>CSMP Program for Example 1</td>
<td>72</td>
</tr>
<tr>
<td>7.3</td>
<td>Patching Information for Example 1</td>
<td>73</td>
</tr>
<tr>
<td>7.4</td>
<td>Patching Diagram from CATS for Example 1</td>
<td>74</td>
</tr>
<tr>
<td>7.5</td>
<td>Simplified Model of the Body Water Regulatory System</td>
<td>76</td>
</tr>
<tr>
<td>7.6</td>
<td>CSMP Program for Example 2</td>
<td>78</td>
</tr>
<tr>
<td>7.7</td>
<td>Patching Information for Example 2</td>
<td>79</td>
</tr>
<tr>
<td>7.8</td>
<td>Patching Diagram from CATS for Example 2</td>
<td>80</td>
</tr>
<tr>
<td>7.9</td>
<td>Vehicle Configuration for the Chased Target Problem</td>
<td>82</td>
</tr>
<tr>
<td>7.10</td>
<td>CSMP Program for Example 3</td>
<td>84</td>
</tr>
</tbody>
</table>
7.11 Patching Information for Example 3
7.12 Patching Diagram from CATS for Example 3
Al.1 Flow Diagram of CATS Execution Model
Al.2 Skeleton program for CONTRL
Al.3 Subroutine. DIMEN
Al.4 Main program CONTRL for Example 3
Al.5 Functional Structure of CATS
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1a</td>
<td>Coding Convention for Extended Operators^3</td>
<td>22</td>
</tr>
<tr>
<td>3.1b</td>
<td>Coding Convention for Function Names</td>
<td>23</td>
</tr>
<tr>
<td>4.1</td>
<td>Transformations of the CSMP Macros</td>
<td>29</td>
</tr>
<tr>
<td>4.2</td>
<td>Categories of Operators</td>
<td>31</td>
</tr>
<tr>
<td>4.3</td>
<td>Priorities of Operators</td>
<td>31</td>
</tr>
<tr>
<td>4.4</td>
<td>Transformations Used in Handling V^†(-P)</td>
<td>39</td>
</tr>
<tr>
<td>4.5</td>
<td>Coding Convention for a/h Devices</td>
<td>44</td>
</tr>
<tr>
<td>6.1</td>
<td>Determination of Maximum Values</td>
<td>59</td>
</tr>
<tr>
<td>6.2</td>
<td>Auxiliary Gain, G, for Magnitude Scaling</td>
<td>61</td>
</tr>
<tr>
<td>6.3</td>
<td>'Housekeeping' Operation for Magnitude Scaling</td>
<td>62</td>
</tr>
<tr>
<td>A2.1</td>
<td>Names and Description of CATS Subroutines</td>
<td>103</td>
</tr>
<tr>
<td>A2.2</td>
<td>Subroutine Calls to</td>
<td>107</td>
</tr>
<tr>
<td>A2.3</td>
<td>Subroutine Calls from</td>
<td>110</td>
</tr>
</tbody>
</table>
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 $/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

\[
\text{CONST} \quad \text{THREE}=3, \text{FOUR}=4
\]

\[ X = \text{THREE}*Y+\text{FOUR}*Z \]
3) An arithmetic expression is not allowed in the argument of the INTGRL statement. Thus the statement

\[ X = \text{INTGRL(TWO,} X+Y) \]

must be written as:

\[ Z = X+Y \]

\[ X = \text{INTGRL(TWO,} Z) \]

The CATS program is free of all the above restrictions.

The handling of complex structure statements; e.g.

\[ X = \text{Y+LIMIT(A,B+C,M+N)}/Z \]

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 Jth statement in the sequence. The vname appearing on the left-hand side of the Jth 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 Jth structure statement; i.e. the vname which represents the 'output' of the Jth structure statement (or the Jth '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 Jth structure statement is V, whose location in VNAME is KV, then

\[ \text{LABEL(J)} = K_V \]

\[ \text{NODTYP(K_V)} = T \]

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 Jth 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

\[ \text{NODIN}(K_v) = N \]

then the device generating the vname stored in \( \text{VNAME}(K_v) \) 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 $\text{LABEL}(R_1),$ $\text{LABEL}(R_2)$ ... $\text{LABEL}(R_N)$. It may however occur that one of the $R_k$ is 0 or -1. This circumstance implies an input from PRED or NRED 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

$$\text{BRGAIN}(M) = S_1$$
$$\text{BRGAIN}(M+1) = S_2$$
$$\vdots$$
$$\text{BRGAIN}(M+N-1) = S_N$$
This would imply that the gains associated with the inputs are in
locations $S_1, S_2, \ldots, S_N$ 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 out-
puts 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:

\[
\begin{align*}
\text{GAIN}(M) &= c_1 \\
\text{GAIN}(M+1) &= -c_2 \\
\text{GAIN}(M+2) &= c_3 \quad (c_1 > 0) \\
& \vdots \\
\text{GAIN}(M+N-1) &= -c_N
\end{align*}
\]
This would imply that the inputs whose names are referenced by $\text{LABEL}(R_1)$, $\text{LABEL}(R_2)$, \ldots $\text{LABEL}(R_N)$, enter the device in question through gains whose magnitudes are $c_1$, $c_2$, \ldots, $c_N$. Furthermore, the example indicates that the vnames referenced by $\text{LABEL}(R_2)$ and $\text{LABEL}(R_N)$ 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

$$\text{NODOUT}(K_V) = L$$

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(KV) = L
PTRGO(KV) = J
NODGO(J) = R1
NODGO(J+1) = R2

\vdots
NODGO(J+L-1) = RL

then the devices which require V as an input are those whose output
labels are referenced by LABEL(R1), LABEL(R2), ..., LABEL(RL).

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 suppressed 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 $\alpha \cdot \beta$. The integral part of this number (namely $\alpha$) is a pointer to the first location in Z associated with the function. The fractional part (namely $\beta$) 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
<table>
<thead>
<tr>
<th>EXTENDED OPERATOR</th>
<th>CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>104</td>
</tr>
<tr>
<td>(</td>
<td>105</td>
</tr>
<tr>
<td>)</td>
<td>106</td>
</tr>
<tr>
<td>,</td>
<td>111</td>
</tr>
<tr>
<td>+</td>
<td>112</td>
</tr>
<tr>
<td>-</td>
<td>113</td>
</tr>
<tr>
<td>*</td>
<td>121</td>
</tr>
<tr>
<td>/</td>
<td>122</td>
</tr>
<tr>
<td><strong>(+)</strong></td>
<td>131*</td>
</tr>
</tbody>
</table>

**Table 3.1(a)**

Coding Convention for Extended Operators

* In the sequel + represents the exponentiation operator.
<table>
<thead>
<tr>
<th>FUNCTION NAME</th>
<th>CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>2</td>
</tr>
<tr>
<td>SQRT</td>
<td>4</td>
</tr>
<tr>
<td>SIN</td>
<td>5</td>
</tr>
<tr>
<td>COS</td>
<td>6</td>
</tr>
<tr>
<td>INLGEN</td>
<td>7</td>
</tr>
<tr>
<td>AFGEN</td>
<td>7</td>
</tr>
<tr>
<td>NOT</td>
<td>8</td>
</tr>
<tr>
<td>TAN</td>
<td>14</td>
</tr>
<tr>
<td>EXP</td>
<td>15</td>
</tr>
<tr>
<td>ALOG</td>
<td>16</td>
</tr>
<tr>
<td>ALOG10</td>
<td>17</td>
</tr>
<tr>
<td>TANH</td>
<td>18</td>
</tr>
<tr>
<td>ATAN</td>
<td>19</td>
</tr>
<tr>
<td>IABS</td>
<td>20</td>
</tr>
<tr>
<td>COMPAR</td>
<td>23</td>
</tr>
<tr>
<td>AND</td>
<td>24</td>
</tr>
<tr>
<td>NAND</td>
<td>25</td>
</tr>
<tr>
<td>IOR</td>
<td>26</td>
</tr>
<tr>
<td>NOR</td>
<td>27</td>
</tr>
<tr>
<td>EOR</td>
<td>28</td>
</tr>
<tr>
<td>INSW</td>
<td>30</td>
</tr>
<tr>
<td>INTGRL</td>
<td>32</td>
</tr>
<tr>
<td>LIMIT</td>
<td>33</td>
</tr>
<tr>
<td>MODINT</td>
<td>34</td>
</tr>
<tr>
<td>REALFL</td>
<td>97</td>
</tr>
<tr>
<td>CMPXPL</td>
<td>98</td>
</tr>
<tr>
<td>LEDLAG</td>
<td>99</td>
</tr>
</tbody>
</table>

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 \( V = \text{rhs} \) where \( \text{rhs} \) 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 CMFXPL 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 (\( \dagger \)) 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

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

where:

$\theta_0$ is either a null operator or a unary + or -

$\theta_i$ 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) \( P/V \)  
(iv) \( V+2 \) or \( V+0.5 \)  
(vi) \( V_1/V_2 \)

Where \( \theta \) is one of the binary operators \( * \) or \( / \).

**Functional expression**

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

(i) \( \text{INTEGRAL}(P, Ce) \)
(ii) \( \text{MODINT}(P, [P, V], [P, V], Ce) \)
(iii) \( \text{LIMIT}(P_1, P_2, Ce) \)
(iv) \( \text{COMPARE}([P, V], [P, V]) \)
(v) \( \text{INSW}([P, V], [P, V], [P, V]) \)
(vi) \( \text{NLFGEN}(\text{name}, V) \)
(vii) \( \text{SQRT}(V) \)
(viii) \( \text{SIN}(V) \)
(ix) \( \text{COS}(V) \)
(x) \( \text{AND}([P, V], [P, V]) \)
(xi) \( \text{NAND}([P, V], [P, V]) \)
(xii) \( \text{IOR}([P, V], [P, V]) \)
(xiii) \( \text{EOR}([P, V], [P, V]) \)
(xiv) \( \text{NOR}([P, V], [P, V]) \)
(xv) \( \text{NOT}(V) \)

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 S\(ldd \) and S\(dd \) 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
<table>
<thead>
<tr>
<th>CSMP Macro-statement</th>
<th>Equivalent 'INTGRL' Statement(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y=REALPL(IC,P,X)</td>
<td>Y=INTGRL(IC,(X-Y)/P)</td>
</tr>
<tr>
<td>Y=LEDLAG(P₁,P₂,X)</td>
<td>Sdd=INTGRL(0,(Sdd-X)/P₂)</td>
</tr>
<tr>
<td></td>
<td>Y=-(P₁*X+(P₁-P₂)*Sdd)/P₂</td>
</tr>
<tr>
<td>Y=CMPXPL(IC₁,IC₂,P₁,P₂,X)</td>
<td>Sdd=INTGRL(IC₂,X-2<em>P₁</em>P₂<em>Sdd-P₂**2</em>Y)</td>
</tr>
<tr>
<td></td>
<td>Y=INTGRL(IC₁,Sdd)</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>STACK</th>
<th>S</th>
<th>INPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>@x=y+LIMIT(A,B</td>
<td>*</td>
<td>C,M+N)/Z;</td>
</tr>
</tbody>
</table>

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
<table>
<thead>
<tr>
<th>Operator Category</th>
<th>Operator Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary</td>
<td>+ - * / †</td>
</tr>
<tr>
<td>unary</td>
<td>unary - and function name e.g. INTGRL, COS ... etc.</td>
</tr>
<tr>
<td>special</td>
<td>( ) = @ ;</td>
</tr>
</tbody>
</table>

Table 4.2

Categories of Operators

<table>
<thead>
<tr>
<th>Priority</th>
<th>Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>@ = ( ) ;</td>
</tr>
<tr>
<td>1</td>
<td>+ - ,</td>
</tr>
<tr>
<td>2</td>
<td>* /</td>
</tr>
<tr>
<td>3</td>
<td>†</td>
</tr>
<tr>
<td>4</td>
<td>unary</td>
</tr>
</tbody>
</table>

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:

$$T_i = X \theta Y$$

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

\[ T_i = 0 \times \]

is generated. The top two items of the stack are deleted (Box 11). The temporary label \( T_i \) 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 \( X = Y + \text{LIMIT}(A, B + C, M + N)/2 \) 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 \( T_i \), 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 \( T_i \) is replaced by the vname that existed on the left-
Figure 4.2

Snapshots of the Decomposition Algorithm Applied to $X=Y+\text{LIMIT}(A,B\times C,M+N)/Z$
hand side of the original statement.

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

\[ X = Y + \text{LIMIT}(A, B \times C, M+N)/Z \]

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 precede and follow the structure \( V^P \) in the source expression. For example, if the structure occurs as \( V^P \times VL \), 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 \( \ldots \cdot V^P \cdot \ldots \) will give rise to the following entries in the POST matrix:
Figure 4.3

The POST Matrix Resulting from

the Source Statement

\[ x = y + \text{LIMIT}(a, b \cdot c, m + n)/2 \]
\[
\begin{bmatrix}
\vdots \\
T_i & - & P & 0 \\
T_j & + & V & T_i \\
\vdots 
\end{bmatrix}
\]

where \(T_i\) and \(T_j\) 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, \(\theta\) represents a null operator, \(\theta_1\) represents one of the binary operators \(+\) or \(-\) and \(\theta_2\) represents one of the binary operators \(*\), \(+\) or \(-\). \(T_i, T_j\) and \(T_k\) 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. \(T_i\) (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
<table>
<thead>
<tr>
<th>Portion of the source expression</th>
<th>Corresponding Generated subexpression</th>
<th>Equivalent form of the expression</th>
<th>Corresponding Generated subexpression</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i) ( 0 \cdot \text{V}^+(-\text{P}) \cdot 0 )</td>
<td>(-P) (\text{V}^+\text{T}1)</td>
<td>(\frac{1}{\text{V}^+\text{P}})</td>
<td>(\text{V}^+\text{P})</td>
</tr>
<tr>
<td></td>
<td>(-P) (\text{V}^+\text{T}1) (\text{V}1)</td>
<td>(\text{V}1/\text{V}^+\text{P})</td>
<td>(\text{V}^+\text{P})</td>
</tr>
<tr>
<td></td>
<td>(-P) (\text{V}^+\text{T}1) (\text{T}1) (\text{V}1)</td>
<td>(\text{V}1/\text{V}^+\text{P})</td>
<td>(\text{V}^+\text{P})</td>
</tr>
<tr>
<td></td>
<td>(-P) (\text{V}^+\text{T}1) (\text{T}1)</td>
<td>(\text{V}1/\text{V}^+\text{P})</td>
<td>(\text{V}^+\text{P})</td>
</tr>
<tr>
<td></td>
<td>(-P) (\text{V}^+\text{T}1) (\text{T}1)</td>
<td>(\text{V}1/\text{V}^+\text{P})</td>
<td>(\text{V}^+\text{P})</td>
</tr>
<tr>
<td></td>
<td>(-P) (\text{V}^+\text{T}1) (\text{T}1)</td>
<td>(\text{V}1/\text{V}^+\text{P})</td>
<td>(\text{V}^+\text{P})</td>
</tr>
<tr>
<td></td>
<td>(-P) (\text{V}^+\text{T}1) (\text{T}1)</td>
<td>(\text{V}1/\text{V}^+\text{P})</td>
<td>(\text{V}^+\text{P})</td>
</tr>
<tr>
<td>(iv) ( \text{V}1^+\text{V}^+(-\text{P}) \cdot \text{V}1 )</td>
<td>(\text{V}1)</td>
<td>(\frac{1}{\text{V}^+\text{T}1\text{V}1})</td>
<td>(\frac{1}{\text{V}^+\text{T}1})</td>
</tr>
<tr>
<td></td>
<td>(\text{V}1)</td>
<td>(\frac{1}{\text{V}^+\text{T}1\text{V}1})</td>
<td>(\frac{1}{\text{V}^+\text{T}1})</td>
</tr>
<tr>
<td>(v) ( \text{V}1\text{V}1^+\text{V}^+(-\text{P}) \cdot \text{V}2 )</td>
<td>(\text{V}1)</td>
<td>(\frac{1}{\text{V}^+\text{T}1\text{V}1\text{V}2})</td>
<td>(\frac{1}{\text{V}^+\text{T}1\text{V}1\text{V}2})</td>
</tr>
<tr>
<td></td>
<td>(\text{V}1)</td>
<td>(\frac{1}{\text{V}^+\text{T}1\text{V}1\text{V}2})</td>
<td>(\frac{1}{\text{V}^+\text{T}1\text{V}1\text{V}2})</td>
</tr>
<tr>
<td>(vi) ( \text{V}1\text{V}1^+\text{V}^+(-\text{P}) \cdot \text{V}2 )</td>
<td>(\text{V}1)</td>
<td>(\frac{1}{\text{V}^+\text{T}1\text{V}1\text{V}2})</td>
<td>(\frac{1}{\text{V}^+\text{T}1\text{V}1\text{V}2})</td>
</tr>
<tr>
<td></td>
<td>(\text{V}1)</td>
<td>(\frac{1}{\text{V}^+\text{T}1\text{V}1\text{V}2})</td>
<td>(\frac{1}{\text{V}^+\text{T}1\text{V}1\text{V}2})</td>
</tr>
<tr>
<td>(vii) ( \text{V}1^+\text{V}^+(-\text{P}) )</td>
<td>(\text{V}1)</td>
<td>(\frac{1}{\text{V}^+\text{T}1\text{V}1})</td>
<td>(\frac{1}{\text{V}^+\text{T}1})</td>
</tr>
<tr>
<td></td>
<td>(\text{V}1)</td>
<td>(\frac{1}{\text{V}^+\text{T}1\text{V}1})</td>
<td>(\frac{1}{\text{V}^+\text{T}1})</td>
</tr>
<tr>
<td>(viii) ( \text{V}1^+\text{V}^+(-\text{P}) )</td>
<td>(\text{V}1)</td>
<td>(\frac{1}{\text{V}^+\text{T}1\text{V}1})</td>
<td>(\frac{1}{\text{V}^+\text{T}1})</td>
</tr>
</tbody>
</table>

Table 4.4

Transformations Used in Handling \( \text{V}^+(-\text{P}) \)
to $T_i$ 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:

$$X = X_1 + X_2 - X_3 + X_4$$

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

- $T_1 = X_1 + X_2$
- $T_2 = T_1 - X_3$
- $Y = T_2 + X_4$

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 \( T_i \) 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:

\[
\begin{array}{|c|c|c|c|}
\hline
\text{S00} & * & B & C \\
\text{T2} & + & M & N \\
\text{T3} & , & \text{S00} & \text{T2} \\
\text{T4} & , & A & \text{T3} \\
\text{S01} & \text{LIMIT} & \text{T4} & 0 \\
\text{S02} & / & \text{S01} & Z \\
\text{X} & + & \text{Y} & \text{S02} \\
\hline
\end{array}
\]

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 \( Y = \text{ABS}(X) \) in a row of POST, gives rise to the following three statements:

\[
\begin{align*}
Sdd &= \text{COMPAR}(X,0) \\
Sdd' &= \text{INSW}(Sdd,0,X) \\
Y &= 2*Sdd' - X
\end{align*}
\]

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

\[
\begin{align*}
X &= Y + S02 \quad (1) \\
S02 &= S01/Z \quad (2) \\
S01 &= \text{LIMIT}(A,T3) \\
&= \text{LIMIT}(A,S00,T2) \quad \text{substitution process} \\
&= \text{LIMIT}(A,S00,M+N) \quad (3) \\
S00 &= B+C \quad (4)
\end{align*}
\]

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

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 those which have the form:

\[ Y = \text{INTGRL}(IC, X) \]

or

\[ Y = \text{MODINT}(IC, X_1, X_2, X) \]

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:

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

In general, however, $\phi_k$ explicitly depends on a subset of the components of $X$ i.e. $\phi_k$ 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

\[ (x^k_{a_1}, x^k_{a_2}, \ldots, x^k_{a_{n_k}}) \]

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

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 $n \times n$ 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 $\theta$.

** It is assumed that the relative order of the components in $X^k$ is the same as in $X$ i.e. $a_i < a_j$ 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 \( k \text{th} \) row of \( D \) corresponds to the \( k \text{th} \) statement of the
sort-set, namely to the statement

\[ X_k = f_k(x^k) \]

The entries in the \( k \text{th} \) row of \( D \) are defined as follows:

\[
D_{kj} = \begin{cases} 
1 & \text{if } X_j \in x^k \\
0 & \text{otherwise}
\end{cases}
\]

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

\[
P2 = P1 \times C \\
Y3 = Y1 + Y2 + Y4 \\
Y4 = Y1 \times Y2 \\
Y1 = \text{INTGRL}(Y1Z, Y1 + Y2 + Y3) \\
Y5 = Y4 + Y2 \\
Y2 = \text{INTGRL}(Y2Z, Y1 + Y2 \times Y4) \\
P1 = A \times B
\]

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 5x5 Dependency matrix then is as follows:

\[
D = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

* 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 kth 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 jth 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 jth 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

START.

Initialize D, ADR, NAMPOS and SEQ vectors

Set: I+1

Set: SEQ(MAX)+K
      NAMPOS(K)+MAX

Set: K+SEQ(I)

Find:
MAX=max{NAMPOS(x_k^a_i)}
1≤i≤n

Set: I+I+1

T

I≤N

F

END

MAX<K

Set: J+I+1

SK+SEQ(J)
SEQ(K-1)+SEQ(J)
NAMPOS(SK)+NAMPOS(SK)-1

T

J≤MAX

F

Set: J=J+1
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
<table>
<thead>
<tr>
<th>Iteration #</th>
<th>Index I</th>
<th>Statement being Processed</th>
<th>Current Contents of SEQ</th>
<th>Current Contents of NAMPOS</th>
<th>Action Performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>$X_1 = \phi_1 (X^1) \equiv \phi_1 (X_5)$</td>
<td>1 2 3 4 5</td>
<td>1 2 3 4 5</td>
<td>reposition $X_1$ in 5th position</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>$X_2 = \phi_2 (X^2) \equiv \phi_2 (X_3)$</td>
<td>2 3 4 5 1</td>
<td>5 1 2 3 4</td>
<td>reposition $X_3$ in 2nd position</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>$X_3 = \phi_3 (X^3) \equiv \phi_3 (0)$</td>
<td>3 2 4 5 1</td>
<td>5 2 1 3 4</td>
<td>none - because of null argument</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>$X_2 = \phi_2 (X^2) \equiv \phi_2 (X_3)$</td>
<td>3 2 4 5 1</td>
<td>5 2 1 3 4</td>
<td>none - because $X_2$ properly positioned</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>$X_4 = \phi_4 (X^4) \equiv \phi_4 (X_3)$</td>
<td>3 2 4 5 1</td>
<td>5 2 1 3 4</td>
<td>none - because $X_4$ properly positioned</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>$X_5 = \phi_5 (X^5) \equiv \phi_5 (0)$</td>
<td>3 2 4 5 1</td>
<td>5 2 1 3 4</td>
<td>none - because of null argument</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>$X_1 = \phi_1 (X^1) \equiv \phi_1 (X_5)$</td>
<td>3 2 4 5 1</td>
<td>5 2 1 3 4</td>
<td>none - because $X_1$ properly positioned</td>
</tr>
</tbody>
</table>

**Figure 5.2**

Snapshots of the Sorting Procedure Applied to

$$X = (P_2, Y_3, Y_4, Y_5, P_1)$$
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.

\[
\begin{align*}
P1 &= A*B \\
P2 &= P1*C \\
Y4 &= Y1*Y2 \\
Y3 &= Y1+Y2+Y4 \\
Y5 &= Y4+Y2 \\
Y1 &= \text{INTGRL}(Y1Z,Y1+Y2+Y3) \\
Y2 &= \text{INTGRL}(Y2Z,Y1+Y2*Y4)
\end{align*}
\]

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 5.3d  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 \( t_1 \)). 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 provides 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
<table>
<thead>
<tr>
<th>Analog Device</th>
<th>Representation</th>
<th>Input-Output Relation</th>
<th>Maximum Value Determination</th>
</tr>
</thead>
<tbody>
<tr>
<td>summer</td>
<td><img src="image" alt="Summer Diagram" /></td>
<td>$Y = (X + Z)$</td>
<td>$Y_m$</td>
</tr>
<tr>
<td>integrator</td>
<td><img src="image" alt="Integrator Diagram" /></td>
<td>$Y = \int \left[ X , dt + Y_0 \right]$</td>
<td>$Y_m$</td>
</tr>
<tr>
<td>mode controlled integration</td>
<td><img src="image" alt="Mode Controlled Integration Diagram" /></td>
<td>$\begin{cases} \int \left[ X , dt + Y_0 \right] &amp; \text{for } X \geq 0, \text{ and } X_2 &gt; 0 \ -Y_0 &amp; \text{for } X \leq 0, \text{ and } X_2 &lt; 0 \end{cases}$</td>
<td>$Y_m$</td>
</tr>
<tr>
<td>multiplier</td>
<td><img src="image" alt="Multiplier Diagram" /></td>
<td>$Y = X \times Z$</td>
<td>$Y_m = X_m \times Z_m$</td>
</tr>
<tr>
<td>squarer</td>
<td><img src="image" alt="Squarer Diagram" /></td>
<td>$Y = X^2$</td>
<td>$Y_m = X_m^2$</td>
</tr>
<tr>
<td>square rooter</td>
<td><img src="image" alt="Square Rooter Diagram" /></td>
<td>$Y = \sqrt{X}$</td>
<td>$Y_m = \sqrt{X_m}$</td>
</tr>
<tr>
<td>divider</td>
<td><img src="image" alt="Divider Diagram" /></td>
<td>$Y = X / Z$</td>
<td>$Y_m = X_m / Z_m$</td>
</tr>
<tr>
<td>function generator</td>
<td><img src="image" alt="Function Generator Diagram" /></td>
<td>$Y = f(X)$</td>
<td>$Y_m = f(X_m)$</td>
</tr>
<tr>
<td>sine</td>
<td><img src="image" alt="Sine Diagram" /></td>
<td>$Y = \sin(X)$</td>
<td>$Y_m = \sin(X_m)$</td>
</tr>
<tr>
<td>cosine</td>
<td><img src="image" alt="Cosine Diagram" /></td>
<td>$Y = \cos(X)$</td>
<td>$Y_m = \cos(X_m)$</td>
</tr>
<tr>
<td>limiter</td>
<td><img src="image" alt="Limiter Diagram" /></td>
<td>$\begin{cases} P_1 &amp; \text{for } X \leq P_1 \ -X &amp; \text{for } -P_1 &lt; X &lt; P_2 \ P_2 &amp; \text{for } X &gt; P_2 \end{cases}$</td>
<td>$Y_m = \max(</td>
</tr>
<tr>
<td>switch</td>
<td><img src="image" alt="Switch Diagram" /></td>
<td>$\begin{cases} -X &amp; \text{for } L = 0 \ -2 &amp; \text{for } L = 1 \end{cases}$</td>
<td>$Y_m = \max(X_m, Y_m)$</td>
</tr>
</tbody>
</table>

Table 6.1

Determination of Maximum Values

$Y_m$ 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.

```
\[ \begin{array}{c}
\text{K} & \text{X/X}_m & \text{G} & \text{J} & \text{Y/Y}_m \\
\end{array} \]
```

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
<table>
<thead>
<tr>
<th>Device J</th>
<th>Linear</th>
<th>Divider (num)</th>
<th>Divider (denom)</th>
<th>Comparator</th>
<th>Switch</th>
<th>Sine/cosine</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear</td>
<td>$X_m/X_m$</td>
<td>$X_m/X_m$</td>
<td>$X_m$</td>
<td>$X_m$</td>
<td>$X_m/Y_m$</td>
<td>$X_m$</td>
</tr>
<tr>
<td>multiplier</td>
<td>$X_m/Y_m$</td>
<td>$X_m/Y_m$</td>
<td>$X_m$</td>
<td>$X_m$</td>
<td>$X_m/Y_m$</td>
<td>$X_m$</td>
</tr>
<tr>
<td>squarer</td>
<td>$X_m/Y_m$</td>
<td>$X_m/Y_m$</td>
<td>$X_m$</td>
<td>$X_m$</td>
<td>$X_m/Y_m$</td>
<td>$X_m$</td>
</tr>
<tr>
<td>square rooter</td>
<td>$X_m/Y_m$</td>
<td>$X_m/Y_m$</td>
<td>$X_m$</td>
<td>$X_m$</td>
<td>$X_m/Y_m$</td>
<td>$X_m$</td>
</tr>
<tr>
<td>divider</td>
<td>$X_m/Y_m$</td>
<td>$X_m/Y_m$</td>
<td>$X_m$</td>
<td>$X_m$</td>
<td>$X_m/Y_m$</td>
<td>$X_m$</td>
</tr>
<tr>
<td>switch</td>
<td>$X_m/Y_m$</td>
<td>$X_m/Y_m$</td>
<td>$X_m$</td>
<td>$X_m$</td>
<td>$X_m/Y_m$</td>
<td>$X_m$</td>
</tr>
<tr>
<td>function generator</td>
<td>$X_m/Y_m$</td>
<td>$X_m/Y_m$</td>
<td>$X_m$</td>
<td>$X_m$</td>
<td>$X_m/Y_m$</td>
<td>$X_m$</td>
</tr>
<tr>
<td>sine/cosine</td>
<td>$1/Y_m$</td>
<td>$X_m/Y_m$</td>
<td>$1$</td>
<td>$1$</td>
<td>$1/Y_m$</td>
<td>$X_m$</td>
</tr>
</tbody>
</table>

Table 6.2

Auxiliary Gain, G, for Magnitude Scaling
<table>
<thead>
<tr>
<th>Device</th>
<th>Configuration after magnitude scaling</th>
<th>Configuration after 'housekeeping'</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparator</td>
<td>![Comparator Diagram]</td>
<td>![Comparator Diagram]</td>
</tr>
<tr>
<td>Divider</td>
<td>![Divider Diagram]</td>
<td>![Divider Diagram]</td>
</tr>
<tr>
<td>Switch</td>
<td>![Switch Diagram]</td>
<td>![Switch Diagram]</td>
</tr>
<tr>
<td>Sin/cosine</td>
<td>![Sin/cosine Diagram]</td>
<td>![Sin/cosine Diagram]</td>
</tr>
</tbody>
</table>

**Table 6.1**

'Mousekeeping' 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 \( \phi_k \)? 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 inverter 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

\[ M_1 x_1 + D_1 (\dot{x}_1 - \dot{x}_2) + K_1 (x_1 - x_2) + F_a = 0 \]
\[ M_2 x_2 + D_1 (\dot{x}_2 - \dot{x}_1) + K_1 (x_2 - x_1) + K_2 (x_2 - x_3) = 0 \]

with initial conditions

\[ x_1(0) = \dot{x}_1(0) = x_2(0) = \dot{x}_2(0) = 0 \]

where

\[ M_1 = \text{one fourth mass of the automobile} = 25 \text{ slugs} \]
\[ M_2 = \text{mass of one wheel} = 2 \text{ slugs} \]
\[ K_1 = \text{linear spring constant of automobile} = 1000 \text{ lb/ft} \]
\[ K_2 = \text{linear spring constant of tire} = 4500 \text{ lb/ft} \]
\[ D_1 = \text{damping constant of shock absorber} = 100 \text{ lb/ft/sec} \]
\[ x_1 = \text{displacement of body of automobile (feet)} \]
\[ x_2 = \text{displacement of wheel (feet)} \]
\[ x_3 = \text{function describing road profile defined as follows:} \]
\[ x_3 = \begin{cases} 
1/12 \text{ ft.} & \text{if } t \leq 0.5 \\
0 & \text{otherwise}
\end{cases} \]

\[ F_a = \text{an externally applied force defined as follows:} \]
\[ F_a = \begin{cases} 
5t & \text{if } t \leq 1 \\
0.2M_1 & \text{if } t > 1
\end{cases} \]

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

\[
\begin{align*}
X1 &= \text{INTGRL}(0., X1D) \\
X1D &= \text{INTGRL}(0., D1 \times X2D / M1 - D1 \times X1 / M1 - K1 \times X1 / M1 + K1 \times X2 / M1 - FA / M1) \\
X2 &= \text{INTGRL}(0., X2D) \\
X2D &= \text{INTGRL}(0., D1 \times X1D / M2 - D1 \times X2D / M2 - (K1 + K2) \times X2 / M2 + K1 \times X1 / M2 + K2 \times X3 / M2) \\
IRD &= \text{TIME} \times X1 \times X2 \\
IR &= \text{INTGRL}(0., IRD) \\
L &= \text{COMPAR}(T1, \text{TIME}) \\
X3 &= \text{INSW}(-L, 0., A) \\
FA &= \text{LIMIT}(0., 0.2 \times M1 \times X3 \times \text{TIME}) \\
\text{TIMER} &= \text{FINITI} = 10., \text{DELT} = 0.01., \text{OUTDEL} = 0.1 \\
\text{METHOD} &= \text{RKSFX} \\
\text{PARAM} &= M1 = 25., M2 = 2., K1 = 1000., K2 = 500., D1 = 100., A = 0.0833., T1 = 0.05 \\
\text{PRINT} &= X1, X1D, X2, X2D, IRD, IR, L, X3, FA \\
\text{END}
\end{align*}
\]

Figure 7.2

CSMP program for Example 1
### Table for Analog Circuit

<table>
<thead>
<tr>
<th>Output Comp Label</th>
<th>Maximum Value</th>
<th>Device Type</th>
<th>Input(s) Comp Label</th>
<th>Problem Parm</th>
<th>Gain Value</th>
<th>Scale Factor</th>
<th>Net Input Comp Label</th>
<th>Pgt Target(s) Comp Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>500</td>
<td>0.6250E-03</td>
<td>SOR</td>
<td>6 -x1 S101</td>
<td>0.1000E 01</td>
<td>0.1000E 01</td>
<td>0.1000E 01</td>
<td>2 -SO</td>
</tr>
<tr>
<td>2</td>
<td>-1RD</td>
<td>0.6250E-02</td>
<td>MULT</td>
<td>11 TIME S101</td>
<td>0.1000E 01</td>
<td>0.1000E 01</td>
<td>0.1000E 01</td>
<td>15 -SO</td>
</tr>
<tr>
<td>3</td>
<td>L</td>
<td>0.1000E 01</td>
<td>COMP</td>
<td>15 -PRET</td>
<td>0.5000E-01</td>
<td>0.1000E 00</td>
<td>0.5000E-02</td>
<td>4 -xZ</td>
</tr>
<tr>
<td>4</td>
<td>x3</td>
<td>0.6330E-01</td>
<td>SW</td>
<td>3 L</td>
<td>0.1000E 01</td>
<td>0.1000E 01</td>
<td>0.1000E 01</td>
<td>9 -x2D</td>
</tr>
<tr>
<td>5</td>
<td>FA</td>
<td>0.5000E 01</td>
<td>LIMIT</td>
<td>15 -PRET</td>
<td>0.5000E-01</td>
<td>0.1000E 00</td>
<td>0.5000E 01</td>
<td>7 -xZ</td>
</tr>
<tr>
<td>6</td>
<td>-x1</td>
<td>0.2500E-01</td>
<td>INT</td>
<td>7 PRET</td>
<td>0.1000E 01</td>
<td>0.1600E 02</td>
<td>0.1000E 01</td>
<td>3 -SO</td>
</tr>
<tr>
<td>7</td>
<td>xZ</td>
<td>0.4000E 00</td>
<td>INT</td>
<td>9 -x2D</td>
<td>0.4000E 01</td>
<td>0.1000E 01</td>
<td>0.1000E 02</td>
<td>6 -xZ</td>
</tr>
<tr>
<td>8</td>
<td>xZ</td>
<td>0.1000E 00</td>
<td>INT</td>
<td>7 PRET</td>
<td>0.1000E 01</td>
<td>0.1600E 02</td>
<td>0.1000E 01</td>
<td>7 -xZ</td>
</tr>
<tr>
<td>9</td>
<td>-x2D</td>
<td>0.3000E 01</td>
<td>INT</td>
<td>14 X2</td>
<td>0.4000E 02</td>
<td>0.2500E 02</td>
<td>0.1000E 02</td>
<td>6 -xZ</td>
</tr>
<tr>
<td>10</td>
<td>IR</td>
<td>0.2000E-02</td>
<td>INT</td>
<td>2 -xRD</td>
<td>0.1000E 01</td>
<td>0.1000E 01</td>
<td>0.1000E 01</td>
<td>2 -IR</td>
</tr>
<tr>
<td>11</td>
<td>TIME</td>
<td>0.1000E 02</td>
<td>INT</td>
<td>15 -PRET</td>
<td>0.1000E 01</td>
<td>0.1000E 01</td>
<td>0.1000E 01</td>
<td>2 -IR</td>
</tr>
<tr>
<td>12</td>
<td>-SO</td>
<td>0.6250E-03</td>
<td>INV</td>
<td>15 -SO</td>
<td>0.1000E 01</td>
<td>0.1000E 01</td>
<td>0.1000E 01</td>
<td>7 -xZ</td>
</tr>
<tr>
<td>13</td>
<td>x1</td>
<td>0.2500E-01</td>
<td>INV</td>
<td>15 -TIME</td>
<td>0.1000E 01</td>
<td>0.1000E 01</td>
<td>0.1000E 01</td>
<td>2 -IR</td>
</tr>
</tbody>
</table>

No time scale factor needed

**Figure 7.3**

Patching Information for Example 1
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

\[
\begin{align*}
\dot{S} &= D_r - K_1 S \\
\dot{G} &= K_1 S - K_2 S \\
\dot{P} &= K_2 G - U \\
\dot{A} &= F - K_3 A \\
A_c &= A/P \\
K_2 &= 3(C_1 O + C_2)P
\end{align*}
\]

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
\[ D_r = \text{drinking rate (liter/hour)} \text{ defined as follows:} \]
\[
D_r = \begin{cases} 
6 & \text{if } t \leq 0.167 \text{ hours} \\
0 & \text{otherwise}
\end{cases}
\]

\[ C_1 = \text{intestinal loss rate constant} = -0.0097 \]

\[ C_2 = \text{intestinal loss rate constant} = 3 \]

\[ F = \text{functional relationship between the rate of ADH release and} \]
\[ \text{the plasma volume defined as follows:} \]
\[
F = \begin{cases} 
420-133P & \text{if } P \leq P_2 = 3 \\
170-50P & \text{if } P_2 < P \leq P_3 = 3.4 \\
0 & \text{if } P > P_3
\end{cases}
\]

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 S=0,G=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=INTEGR(L,S,DR=K1*S)
G=INTEGR(GZ,K1*S-Z)
P=INTEGR(PZ,Z-0.06*U)
A=INTEGR(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=CMPAR(P3,P)
L2=CMPAR(P2,P)
F=F1S+F2S

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

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

TIMER FINTIM=10,DELT=0.01,OUTDELT=0.1
PRITPLT AC,A,P,F,U
METHOD RKS
END

Figure 7.6

CSMP program for Example 2
Figure 7.7

Patching Information for Example 2
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 \( \theta \) 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:

\[
\begin{align*}
Y_t &= -G \\
\dot{X}_t &= V_t \cos \theta \\
\dot{Y}_t &= V_t \sin \theta \\
X_c &= X_t - X_c \\
Y_c &= Y_t - Y_c \\
D_x &= \sqrt{D_x^2 + D_y^2} \\
H &= \frac{D_x}{H} \frac{D_y}{H} \\
\cos \theta &= D_x / H \\
\sin \theta &= D_y / H
\end{align*}
\]

with initial conditions

\[
\begin{align*}
X_c (0) &= 0 \\
Y_c (0) &= 2200 \\
X_t (0) &= 3000 \\
Y_t (0) &= 2000 \\
\dot{X}_c (0) &= V_c \cos \theta (0) \\
\dot{Y}_c (0) &= V_c \sin \theta (0)
\end{align*}
\]
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, VCD=100, VTD=400
DX=X*T-XC
DY=Y*T-YC
VXD=VCD*costht
VYD=VCD*sinht
costht=DX/H
sinht=DY/H
H=(DX**2+DY**2)**0.5
XCD=INTEGRAL(VX0*T*costht)
YCD=INTEGRAL(VY0*T*sinht)
X=INTEGRAL(0,XCD)
Y=INTEGRAL(0,YCD)
X=T=INTEGRAL(3000,VTO)
Y=T=INTEGRAL(2000,YTD)
SIGNL=COMPARE(H,100)
TIMER DELT=.01,FINTIM=5
END

Figure 7.10

CSMP program for Example 3
<table>
<thead>
<tr>
<th>Output Comp</th>
<th>Output Device</th>
<th>Output Comp Value</th>
<th>Output Device Type</th>
<th>Input(s) Comp</th>
<th>Input(s) Label</th>
<th>Problem Gain Value</th>
<th>Scale Factor</th>
<th>Key Input GAIN</th>
<th>Footnote Target(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 DX</td>
<td>0.35000E 04</td>
<td>SUMMER</td>
<td>15 -XT</td>
<td>S101</td>
<td>0.10000E 01</td>
<td>0.14299E 01</td>
<td>0.14299E 01</td>
<td>1 4 S00</td>
<td>1 17 -DX</td>
</tr>
<tr>
<td>2 DY</td>
<td>0.20000E 03</td>
<td>SUMMER</td>
<td>16 -YT</td>
<td>S101</td>
<td>0.10000E 01</td>
<td>0.10000E 02</td>
<td>0.10000E 02</td>
<td>3 3 S01</td>
<td>4 18 -DY</td>
</tr>
<tr>
<td>3 50I</td>
<td>0.40000E 05</td>
<td>SQRT</td>
<td>12 -DY</td>
<td>S101</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>5 -502</td>
<td>5 -502</td>
</tr>
<tr>
<td>4 500</td>
<td>0.12255E 08</td>
<td>SQRT</td>
<td>17 -DX</td>
<td>S101</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>5 -502</td>
<td>5 -502</td>
</tr>
<tr>
<td>5 -502</td>
<td>0.10000E 08</td>
<td>SUMMER</td>
<td>4 S00</td>
<td>S101</td>
<td>0.10000E 01</td>
<td>0.12255E 01</td>
<td>0.12255E 01</td>
<td>4 S00</td>
<td>5 6 M</td>
</tr>
<tr>
<td>6 M</td>
<td>0.31622E 04</td>
<td>SQRT</td>
<td>5 -502</td>
<td>S101</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>7 S101</td>
<td>6 S101</td>
</tr>
<tr>
<td>7 S101</td>
<td>0.10000E 01</td>
<td>DIVO</td>
<td>10 -DY</td>
<td>S101</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>7 11 S00</td>
<td>7 11 S00</td>
</tr>
<tr>
<td>8 S101</td>
<td>0.10000E 01</td>
<td>DIVO</td>
<td>10 -DY</td>
<td>S101</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>7 11 S00</td>
<td>7 11 S00</td>
</tr>
<tr>
<td>9 S101</td>
<td>0.10000E 01</td>
<td>COMP</td>
<td>6 H</td>
<td>S101</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>7 11 S00</td>
<td>7 11 S00</td>
</tr>
<tr>
<td>10 -XCD</td>
<td>0.25000E 04</td>
<td>INT</td>
<td>8 -S101</td>
<td>VCD</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>7 11 S00</td>
<td>7 11 S00</td>
</tr>
<tr>
<td>11 -YCO</td>
<td>0.40000E 03</td>
<td>INT</td>
<td>7 -S101</td>
<td>VCG</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>7 11 S00</td>
<td>7 11 S00</td>
</tr>
<tr>
<td>12 XCC</td>
<td>0.60000E 04</td>
<td>INT</td>
<td>10 -XCD</td>
<td>S101</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>7 11 S00</td>
<td>7 11 S00</td>
</tr>
<tr>
<td>13 YCO</td>
<td>0.25000E 04</td>
<td>INT</td>
<td>11 -XCD</td>
<td>S101</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>7 11 S00</td>
<td>7 11 S00</td>
</tr>
<tr>
<td>14 YCO</td>
<td>0.30000E 03</td>
<td>INT</td>
<td>12 -XCD</td>
<td>S101</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>7 11 S00</td>
<td>7 11 S00</td>
</tr>
<tr>
<td>15 -XCO</td>
<td>0.50000E 04</td>
<td>INT</td>
<td>14 -XCD</td>
<td>S101</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>7 11 S00</td>
<td>7 11 S00</td>
</tr>
<tr>
<td>16 -YCO</td>
<td>0.20000E 04</td>
<td>INT</td>
<td>15 -XCD</td>
<td>S101</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>7 11 S00</td>
<td>7 11 S00</td>
</tr>
<tr>
<td>17 -XCO</td>
<td>0.35000E 04</td>
<td>INV</td>
<td>16 -XCD</td>
<td>S101</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>7 11 S00</td>
<td>7 11 S00</td>
</tr>
<tr>
<td>18 -YCO</td>
<td>0.20000E 03</td>
<td>INV</td>
<td>17 -XCD</td>
<td>S101</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>7 11 S00</td>
<td>7 11 S00</td>
</tr>
<tr>
<td>19 -XCO</td>
<td>0.31622E 04</td>
<td>INV</td>
<td>18 -XCD</td>
<td>S101</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>0.10000E 01</td>
<td>7 11 S00</td>
<td>7 11 S00</td>
</tr>
</tbody>
</table>

No time scale factor needed

Figure 7.11
Patching Information for Example 3
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

Al.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 Al.1.

Al.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 Al.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 Al.3.
Execute the program DIMEN to dimension the arrays in the main program, CONTRL, using the data supplied by the following the DIMEN.NSIZE DD card.

Compile the resultant program, CONTRL, and link it with other CATS subroutine.

Read the CSMP program specified by the TRANSLAT.INPUT DD card and store the pertinent data.

Decompose the CSMP structure statements into 'substatements'.

Figure A1.1
Flow Diagram of CATS Execution Model
A

Sort the CSMP structure statements

Create, compile and load the subroutine RHS

Using the subroutine RHS compute the values of the pnames and maximum values of the vnames

Determine the minimum number of inverters required and the polarities of the a/h device outputs

B

Figure A1.1 (cont'd)
Figure A1.1 (cont'd)
Figure A1.2
Skeleton program for COORTI
INTEGER J1,J4,Y(118)
INTEGER TEST(1) DOLLAR(1)
LOGICAL A(1) VECTOR(4240),SIZE(1472),LTEST(2)
EXTERNAL (TEST,LTEST)
CALL SMTSAUDU(444,472)
READ(5,1)N1,N2,N3,N4,N5
FORMAT(5,12)
J1=N1+3*N2+**N3+N5
J3=J1+1
J5=(4*N3+N4)*5
J7=(N2*N3*2)+80+320
J9=J6+3
J10=J7+320
J11=J10+1
K1=J1+1
K2=J7+319
I=4*N1
K3=J4
K5=K4+1
K6=K5+1
K7=K6+1
K8=K7+1
K9=K8+1
K10=K9+1
K11=K10+1
K12=K11+1
K13=K12+1
K14=K13+1
K15=K14+1
K16=K15+1
K17=K16+1
K18=K17+1
K19=K18+1
K20=K19+1
K21=K20+1
K22=K21+1
K23=K22+1
K24=K23+1
K25=K24+1
WRITE(5,2)J1,J2,J3,J4,J5,J6,J7,J8,J9,J10,J11,J12,J13,J14,J15,J16,J17,J18,J19,J20,J21,J22,J23,J24,J25
READ(5,99,1)DOLLAR(1)=1,472
READ(4,6)VECTOR(4),SIZE(1),N1,N2
VECTOR(K)=SIZE(4)
CONTINUE
WRITE(5,7)VECTOR(K),K=1,4240
FORMAT(8041)
STOP
FND

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 'SIZ', 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:

\[ N1 = 10, \quad N2 = 0, \quad N3 = 12, \quad N4 = 0, \quad N5 = 0. \]

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
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, VCD, 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.

Al.4 JCL for Executing CATS Job

The CATS job is divided into the following four distinct
steps:
1) DIMENSN - creation of CONTROL program
2) FORT - compilation of CONTROL program
3) LKED - linkage of CONTROL program
4) TRANSLAT- execution of CONTROL program

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

Table A2.1

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

Table A2.1 (cont'd)
<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEDLAG</td>
<td>decomposes the CSMP macro LEDLAG into an equivalent INTGRL form</td>
</tr>
<tr>
<td>LEXY</td>
<td>generates numeric codes for the source statements</td>
</tr>
<tr>
<td>LOCATE</td>
<td>locates positions of identifiers in the PNAME/VNAME vectors</td>
</tr>
<tr>
<td>LOGIC</td>
<td>determines the cost index of a policy applied to the modified dependency matrix</td>
</tr>
<tr>
<td>MAGSCL</td>
<td>computes the magnitude scale factor for pot settings</td>
</tr>
<tr>
<td>MAXVAL</td>
<td>computes the values of pnames and maximum values of vnames</td>
</tr>
<tr>
<td>MINI</td>
<td>minimizes the number of inverters for a/h computer program</td>
</tr>
<tr>
<td>MOVE</td>
<td>moves any misplaced Initial statement in the Dynamic segment into the Initial segment</td>
</tr>
<tr>
<td>NAMID</td>
<td>extracts the identifiers in the source CSMP program</td>
</tr>
<tr>
<td>NEWSA</td>
<td>determines the polarities of the outputs of the a/h devices</td>
</tr>
<tr>
<td>NEWSTM</td>
<td>generates new substatements from the source statements and stores them in SC</td>
</tr>
<tr>
<td>NONLIN</td>
<td>updates the inputs to non-linear devices</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>prints out the CATS output</td>
</tr>
<tr>
<td>OUTPT1</td>
<td>prints out the result of the minimization procedure</td>
</tr>
<tr>
<td>PATCH</td>
<td>updates the analog tableau after inserting inverters</td>
</tr>
<tr>
<td>PCM</td>
<td>predictor-corrector method to solve differential equations</td>
</tr>
<tr>
<td>PROCES</td>
<td>processes the data, control and translation statements of the CSMP program</td>
</tr>
<tr>
<td>READIN</td>
<td>reads and processes the input data which is the CSMP program</td>
</tr>
</tbody>
</table>

Table A2.1 (cont'd)
<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REALPL</td>
<td>decomposes the CSMP macro REALPL into an equivalent INTGRGL form</td>
</tr>
<tr>
<td>REDABS</td>
<td>decomposes the 'ABS' function into equivalent sub-statements, implementable on a/h computer</td>
</tr>
<tr>
<td>REDSTM</td>
<td>decomposes the source expression into subexpressions</td>
</tr>
<tr>
<td>RHS</td>
<td>executes the assignment statements of the CSMP program</td>
</tr>
<tr>
<td>SET$S</td>
<td>extracts a particular set during the inverter minimization operation of the modified dependency matrix</td>
</tr>
<tr>
<td>SORT</td>
<td>initializes the various arrays for sorting operation</td>
</tr>
<tr>
<td>SORT1</td>
<td>sorts the source statements into a natural order</td>
</tr>
<tr>
<td>SORT2</td>
<td>selects and repositions each source statement into a natural order</td>
</tr>
<tr>
<td>STARTR</td>
<td>a Runge-Kutta technique used to generate starting values for the PCM routine</td>
</tr>
<tr>
<td>STMTYP</td>
<td>determines and stores the source structure statements in SC</td>
</tr>
<tr>
<td>STRCUR</td>
<td>builds up the tableau for the a/h program from the source statements</td>
</tr>
<tr>
<td>SUBSET</td>
<td>determines if a set ( S_1 ) is a subset of ( S_2 )</td>
</tr>
<tr>
<td>SWITCH</td>
<td>repositions a source statement after a specified statement</td>
</tr>
<tr>
<td>TABLE</td>
<td>determines if a device output needs to be included in the magnitude scaling operation</td>
</tr>
<tr>
<td>TRANS</td>
<td>translates the storage format of an identifier</td>
</tr>
<tr>
<td>TYPE</td>
<td>determines the type of a/h device corresponding to the source statement</td>
</tr>
</tbody>
</table>

Table A2.1 (cont'd)
<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Calls to</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS</td>
<td>-</td>
</tr>
<tr>
<td>ADJUST</td>
<td>-</td>
</tr>
<tr>
<td>AMATX</td>
<td>-</td>
</tr>
<tr>
<td>APPLY</td>
<td>CRTSET</td>
</tr>
<tr>
<td>BEGIN</td>
<td>AMATX, BMATX, CRTRHS, DECOMP, DYNA, FORASM, GETINV, INCOND, INVERT, MAGSCL, MAXVAL, MINI, MOVE, OUTPUT, PATCH, READIN, SORT, STMTP, STRCUR</td>
</tr>
<tr>
<td>BUILD</td>
<td>TRANS</td>
</tr>
<tr>
<td>BMATX</td>
<td></td>
</tr>
<tr>
<td>CMPR</td>
<td>NAMID</td>
</tr>
<tr>
<td>CMPXPL</td>
<td>ADJUST, GENV, INSERT</td>
</tr>
<tr>
<td>CNTCRD</td>
<td></td>
</tr>
<tr>
<td>COMBTN</td>
<td></td>
</tr>
<tr>
<td>COMPT</td>
<td>SETSS, SUBSET</td>
</tr>
<tr>
<td>CONTRL</td>
<td>BEGIN</td>
</tr>
<tr>
<td>CRTEQV</td>
<td>TRANS</td>
</tr>
<tr>
<td>CRTIOS</td>
<td>TRANS</td>
</tr>
<tr>
<td>CRTRHS</td>
<td>CRTEQV, CRTIOS, CRTSTM</td>
</tr>
<tr>
<td>CRTSET</td>
<td></td>
</tr>
<tr>
<td>CRTSTM</td>
<td>BUILD</td>
</tr>
<tr>
<td>DECOMP</td>
<td>CMPXPL, LEDLAG, REALPL</td>
</tr>
<tr>
<td>DIMEN</td>
<td></td>
</tr>
<tr>
<td>DYNA</td>
<td>ADJUST, EXPON, LABEL, NEWSTM, REDSTM</td>
</tr>
<tr>
<td>ERRM</td>
<td></td>
</tr>
<tr>
<td>EXPON</td>
<td></td>
</tr>
</tbody>
</table>

Table A2.2

Subroutine Calls to
<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Calls to</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILESC</td>
<td>CNTCRD, INSERT</td>
</tr>
<tr>
<td>FORASM</td>
<td>IEYFORT, IEWL (FORTRAN compiler and loader)</td>
</tr>
<tr>
<td>GENP</td>
<td>ERRM, LOCATE, TRANS</td>
</tr>
<tr>
<td>GENV</td>
<td>ERRM, LOCATE, TRANS</td>
</tr>
<tr>
<td>GETINV</td>
<td>GENP</td>
</tr>
<tr>
<td>GETNB</td>
<td>SETB99</td>
</tr>
<tr>
<td>HASH</td>
<td>-</td>
</tr>
<tr>
<td>INCOND</td>
<td>-</td>
</tr>
<tr>
<td>INPUT</td>
<td>GENP, INSERT, SWITCH</td>
</tr>
<tr>
<td>INPUT1</td>
<td>INPUT</td>
</tr>
<tr>
<td>INPUT2</td>
<td>INPUT</td>
</tr>
<tr>
<td>INPUT3</td>
<td>INPUT</td>
</tr>
<tr>
<td>INSERT</td>
<td>ERRM</td>
</tr>
<tr>
<td>INVERT</td>
<td>ERRM, NONLIN</td>
</tr>
<tr>
<td>LABEL</td>
<td>GENP, GENV</td>
</tr>
<tr>
<td>LEDLAG</td>
<td>ADJUST, GENV, INSERT</td>
</tr>
<tr>
<td>LEXY</td>
<td>ERRM, GENP, GETNB, LOCATE, NAMID</td>
</tr>
<tr>
<td>LOCATE</td>
<td>ERRM, HASH</td>
</tr>
<tr>
<td>LOGIC</td>
<td>APPLY, SET$$, SUBSET</td>
</tr>
<tr>
<td>MAGSCL</td>
<td>TABLE</td>
</tr>
<tr>
<td>MAXVAL</td>
<td>ACCESS, FCM, RHS, SETB99</td>
</tr>
<tr>
<td>MINI</td>
<td>APPLY, COMETN, COMPT, LOGIC, OUTPT1, SET$$, SUBSET</td>
</tr>
<tr>
<td>MOVE</td>
<td>LOCATE</td>
</tr>
<tr>
<td>NAMID</td>
<td>ERRM, TRANS</td>
</tr>
<tr>
<td>NEWSA</td>
<td>-</td>
</tr>
</tbody>
</table>

Table A2.2 (cont'd)
<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Calls to</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEWSTM</td>
<td>INSERT, REDABS, TYPE</td>
</tr>
<tr>
<td>NONLIN</td>
<td></td>
</tr>
<tr>
<td>OUTPUT</td>
<td></td>
</tr>
<tr>
<td>OUTPT1</td>
<td>APPLY</td>
</tr>
<tr>
<td>PATCH</td>
<td></td>
</tr>
<tr>
<td>PCM</td>
<td>ACCESS, RHS, STARTR</td>
</tr>
<tr>
<td>PROCES</td>
<td>CNTCRD, GETNR, LOCATE, NAMID</td>
</tr>
<tr>
<td>READIN</td>
<td>CMPR, CNTCRD, FILLSC, LOCATE, NAMID, PROCES</td>
</tr>
<tr>
<td>REALPL</td>
<td>ADJUST, INSERT</td>
</tr>
<tr>
<td>REDABS</td>
<td>GENV, INSERT</td>
</tr>
<tr>
<td>REDSTM</td>
<td></td>
</tr>
<tr>
<td>RHS</td>
<td>CSMP functions</td>
</tr>
<tr>
<td>SET$S</td>
<td></td>
</tr>
<tr>
<td>SORT</td>
<td>ERRM, SORT1, SORT2</td>
</tr>
<tr>
<td>SORT1</td>
<td></td>
</tr>
<tr>
<td>SORT2</td>
<td>SWITCH</td>
</tr>
<tr>
<td>STARTR</td>
<td>RHS</td>
</tr>
<tr>
<td>STMTYP</td>
<td>INSERT, LEXY</td>
</tr>
<tr>
<td>STCUR</td>
<td>GEMP, INPUT1, INPUT2, INPUT3, INSERT, SWITCH</td>
</tr>
<tr>
<td>SUBSET</td>
<td></td>
</tr>
<tr>
<td>SWITCH</td>
<td></td>
</tr>
<tr>
<td>TABLE</td>
<td></td>
</tr>
<tr>
<td>TRANS</td>
<td></td>
</tr>
<tr>
<td>TYPE</td>
<td></td>
</tr>
</tbody>
</table>

Table A2.2 (cont'd)
<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Called from</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS</td>
<td>MAXVAL, PCM</td>
</tr>
<tr>
<td>ADJUST</td>
<td>CMPXPL, DYN, LEDLAG, REALPL</td>
</tr>
<tr>
<td>AMATX</td>
<td>BEGIN</td>
</tr>
<tr>
<td>APPLY</td>
<td>LOGIC, MIN, OUTPT1</td>
</tr>
<tr>
<td>BEGIN</td>
<td>CONTROL</td>
</tr>
<tr>
<td>BMATX</td>
<td>BEGIN</td>
</tr>
<tr>
<td>BUILT</td>
<td>CRTSTM</td>
</tr>
<tr>
<td>CMPR</td>
<td>READIN</td>
</tr>
<tr>
<td>CMPXPL</td>
<td>DECOMP</td>
</tr>
<tr>
<td>CNTCRD</td>
<td>FILLSC, PROCES, READIN</td>
</tr>
<tr>
<td>COMBTN</td>
<td>MINI</td>
</tr>
<tr>
<td>COMPT</td>
<td>MINI</td>
</tr>
<tr>
<td>CONTRL</td>
<td></td>
</tr>
<tr>
<td>CTRDQV</td>
<td>CRTRHS</td>
</tr>
<tr>
<td>CTREOS</td>
<td>CRTRHS</td>
</tr>
<tr>
<td>CRTRHS</td>
<td>BEGIN</td>
</tr>
<tr>
<td>CRTSET</td>
<td>APPLY</td>
</tr>
<tr>
<td>CRTSTM</td>
<td>CRTRHS</td>
</tr>
<tr>
<td>DECOMP</td>
<td>BEGIN</td>
</tr>
<tr>
<td>DIMEN</td>
<td></td>
</tr>
<tr>
<td>DYN</td>
<td>BEGIN</td>
</tr>
<tr>
<td>ERRM</td>
<td>GEN, GENV, INSERT, INVERT, LEXY, LOCATE NAMID, SORT</td>
</tr>
<tr>
<td>EXPO</td>
<td>DYN</td>
</tr>
<tr>
<td>FILLSC</td>
<td>READIN</td>
</tr>
</tbody>
</table>

Table A2.3

Subroutine Calls from
<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Called from</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORASM</td>
<td>BEGIN</td>
</tr>
<tr>
<td>GENP</td>
<td>GETINV, INPUT, LABEL, LEXY, STRCUR</td>
</tr>
<tr>
<td>GENV</td>
<td>CMPXPL, LABEL, LEDLAG, REDABS</td>
</tr>
<tr>
<td>GETINV</td>
<td>BEGIN</td>
</tr>
<tr>
<td>GETNB</td>
<td>LEXY, PROCES</td>
</tr>
<tr>
<td>HASH</td>
<td>LOCATE</td>
</tr>
<tr>
<td>INPUT</td>
<td>INPUT1, INPUT2, INPUT3</td>
</tr>
<tr>
<td>INPUT1</td>
<td>STRCUR</td>
</tr>
<tr>
<td>INPUT2</td>
<td>STRCUR</td>
</tr>
<tr>
<td>INPUT3</td>
<td>STRCUR</td>
</tr>
<tr>
<td>INSERT</td>
<td>CMPXPL, FILLSC, INPUT, LEDLAG, NEWSTM, REALPL, REDABS, STMTYP, STRCUR</td>
</tr>
<tr>
<td>INVERT</td>
<td>BEGIN</td>
</tr>
<tr>
<td>LABEL</td>
<td>DYNAA</td>
</tr>
<tr>
<td>LEDLAG</td>
<td>DECOMP</td>
</tr>
<tr>
<td>LEXY</td>
<td>STMTYP</td>
</tr>
<tr>
<td>LOCATE</td>
<td>GENP, GENV, LEXY, MOVE, PROCES, READIN</td>
</tr>
<tr>
<td>LOGIC</td>
<td>MINI</td>
</tr>
<tr>
<td>MAGSCL</td>
<td>BEGIN</td>
</tr>
<tr>
<td>MAXUAL</td>
<td>BEGIN</td>
</tr>
<tr>
<td>MINI</td>
<td>BEGIN</td>
</tr>
<tr>
<td>MOVE</td>
<td>BEGIN</td>
</tr>
<tr>
<td>NAMID</td>
<td>CMPR, LEXY, PROCES, READIN</td>
</tr>
<tr>
<td>NEW$A</td>
<td>BEGIN</td>
</tr>
<tr>
<td>NEWSTM</td>
<td>DYNAA</td>
</tr>
<tr>
<td>NONLIN</td>
<td>INVERT</td>
</tr>
</tbody>
</table>

Table A2.3 (cont'd)
<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Called from</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTPUT</td>
<td>BEGIN</td>
</tr>
<tr>
<td>OUTPT1</td>
<td>MINI</td>
</tr>
<tr>
<td>PATCH</td>
<td>BEGIN</td>
</tr>
<tr>
<td>PCM</td>
<td>MAXVAL</td>
</tr>
<tr>
<td>PROCES</td>
<td>READIN</td>
</tr>
<tr>
<td>READIN</td>
<td>BEGIN</td>
</tr>
<tr>
<td>REALPL</td>
<td>DECOMP</td>
</tr>
<tr>
<td>REDABS</td>
<td>NEWSTM</td>
</tr>
<tr>
<td>REDSTM</td>
<td>DYN</td>
</tr>
<tr>
<td>RHS</td>
<td>MAXVAL, PCM, STARTR</td>
</tr>
<tr>
<td>SET$S</td>
<td>COMPT, LOGIC, MINI</td>
</tr>
<tr>
<td>SORT</td>
<td>BEGIN</td>
</tr>
<tr>
<td>SORT1</td>
<td>SORT</td>
</tr>
<tr>
<td>SORT2</td>
<td>SORT</td>
</tr>
<tr>
<td>STARTR</td>
<td>PCM</td>
</tr>
<tr>
<td>STMTYP</td>
<td>BEGIN</td>
</tr>
<tr>
<td>STRCUR</td>
<td>BEGIN</td>
</tr>
<tr>
<td>SUBSET</td>
<td>COMPT, LOGIC, MINI</td>
</tr>
<tr>
<td>SWITCH</td>
<td>INPUT, SORT2, STRCUR</td>
</tr>
<tr>
<td>TABLE</td>
<td>MAGSCL</td>
</tr>
<tr>
<td>TRANS</td>
<td>BUILD, CRTEQV, CRTIOS, GENP, GENV, NAMID</td>
</tr>
<tr>
<td>TYPE</td>
<td>NEWSTM</td>
</tr>
</tbody>
</table>

Table A2.3 (cont'd)
APPENDIX 3

Restrictions on the use of CATS

The users are restricted in the following ways 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, AMAXO, AMAX1, MAXO, MAX1, AMINO, AMIN1, MINO, 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 Sd1, where d is a decimal digit.

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


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