ESSCOR

engineering sciences and systems corporation

ES-R-A007-01

October 1978

Assessment of Procedures and
Preliminary Software Design
for Fault Tree Synthesis

P.O. 2967409

Harold J. Payne

NOTICE
This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Department of Energy, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

Submitted to: Dr. Howard E. Lambert
Lawrence Livermore Laboratory
Livermore, California

... INFORMATION ON THIS DOCUMENT IS UNLIMITED

TABLE OF CONTENTS

<u>Section</u>	<u>Page</u>
1.0 INTRODUCTION.	1
2.0 GENERAL FUNCTIONAL SPECIFICATIONS	6
3.0 TECHNICAL SPECIFICATIONS.	9
3.1 Representation of the Material Control System. .	9
3.2 Digraph Representation	12
3.2.1 Unit Model Digraphs	12
3.2.2 System Digraph.	21
3.2.3 Digraph Conventions	24
3.3 Fault Tree Synthesis Procedures.	27
3.4 Finding Loops.	31
4.0 SOFTWARE FUNCTIONAL SPECIFICATIONS.	42
4.1 Decomposition of Functions	42
4.2 Data Structures.	43
4.2.1 SYSDIG Data Structures.	43
4.2.2 FTSYN Data Structures	43
4.2.3 Details of Data Structures.	45
4.3 Code Structure	48
4.4 Module Descriptions.	59
4.5 Software Standards	62
5.0 CONCLUSIONS	67
6.0 REFERENCES.	70

1.0 INTRODUCTION

A methodology for assessing the vulnerability of plutonium storage facilities to theft has been developed at Lawrence Livermore Laboratories (LLL) [1,2,3]. This methodology, termed the Safeguards Effectiveness Assessment Methodology, involves a series of analysis steps, as indicated in Figure 1-1. Our interest here is focused on the Material Control and Accounting (MC&A) system assessment, and more particularly, on the analysis task shown as "event set generation" in Figure 1-1.

The process of "event set generation" results in the itemization of all minimal sets of events, including actions of an adversary or adversaries and failures in the MC&A system, which are sufficient to lead to successful theft of material (minimal cut sets). One approach to event set generation being actively pursued at LLL, particularly by Dr. Howard E. Lambert, is to first create a representation of the MC&A system and potential adversary actions in the form of a directed network of a special type, hereafter termed the system digraph. Then a specific algorithm is applied to the system digraph to yield a fault tree, which is a compact representation of the minimal set of events discussed above. Further, a computer program, FTAP (Fault Tree Analysis Program) has been developed which can operate on a fault tree to produce, explicitly, the collection of minimal cut sets (as well as a number of other useful facts).

The two issues of concern here, then, are

- construction of the system digraph,
- fault tree synthesis from a system digraph.

These issues have both been recently explored by Lapp and Powers [5,7].

Figure 1-1. The LLL assessment procedure [2].

They have, in fact, developed software to construct a system digraph. However, the constructs of the system representation they use are not natural to the MC&A system, so that artificial associations of chemical terms used by Lapp and Powers must be made with the elements of the MC&A system. Moreover, the requirements for the system digraph to represent the MC&A system include representation of time delay, a feature not present in their software.

Lapp and Powers have also developed an algorithm for fault tree synthesis. Again, there are shortcomings in their work in so far as immediate application to the MC&A system assessment problem is concerned, due to the limitation on the complexity of the network their algorithm will handle.

However, the Lapp-Powers approach has promise for the MC&A system assessment problem. Consequently, current activities at LLL, particularly on the part of Dr. Howard E. Lambert, are directed toward development of this type of analysis procedure to produce the necessary methodology.

The effort reported here is intended to complement this activity. The objectives are:

- to provide an independent assessment of the Lapp-Powers approach to system digraph generation,
- to provide a preliminary software design, primarily to offer an assessment of the feasibility or difficulties in producing software to embody both system digraph generation and the fault tree synthesis procedures.

It was not our objective to participate in the necessary development of an algorithm for fault tree synthesis following the Lapp-Powers approach.

In the course of the development of the software design, a sequence of documents providing specifications for the software should be written, each in turn representing a detailing of some aspect previously described. The following set of four documents appears to be relevant to this particular project:

(1) General Functional Specifications

This document should describe, in general terms, the function or functions of the software system, the nature of the input data, and outputs to be developed, both as to generic types and media. The development of this document represents the first stage in the software design process.

(2) Technical Specifications

In certain applications, including the one being addressed here, there are a number of technical procedures or algorithms necessary to accomplish the functions of the software. These should be detailed in this second document.

(3) Software Functional Specifications

Having the first two documents, the next step is to develop a software design which explicitly accounts for the general functional specifications and for the nature of the procedures and algorithms as detailed in the technical specifications. This design should involve the development of a code structure, in top-down, modular fashion, to a point where all functions are accounted for and distinct procedures and algorithms are represented. Further, data structures should be delineated to the point where all necessary data types are identified. Finally, in connection with the code structure and data types identified, each module should be described in terms of input data, functions and output data.

(4) Detailed Software Technical Specifications

The final stage of the software design effort is the elaboration of the elements of the software functional specifications. First, the data structures must be detailed to define all variables. Next, a detailed description of each module is produced. This may take the form of "pseudo-code" or flow charts, accompanied by a listing of all input and output variables.

This document should also provide software standards, i.e., detailed descriptions of in-line documentation, and programming and style standards. Such standards can contribute significantly to simplifying the code debugging process, and will significantly aid subsequent software maintenance. Finally, this document should provide a detailed software test plan.

This sequence is generally found to be an iterative process, i.e., one in which developments at a more detailed level involve revisions to a higher level document. The result, however, is accurate documentation available in the course of the debugging and testing phases, and immediately at the conclusion of the software testing phase. This contrasts with a common practice of producing documentation only after the code is complete. A fifth document, the user's manual, can be constructed from the four documentation reports described above.

The next three sections of this report are devoted to preliminary versions of each of the first three documents identified, respectively. That these are preliminary versions necessarily follows from the incomplete nature of the technical specifications, particularly with regard to the fault tree synthesis procedures. Section 5 provides a concise summary of conclusions drawn. All references are collected in Section 6.

2.0 GENERAL FUNCTIONAL SPECIFICATIONS

The software requirements for automated generation of fault trees can best be met by dividing the software into two programs. The first program, the System Digraph Construction Program, would have the function of creating the system digraph and performing analysis of that digraph. This general function is further delineated in Figure 2-1.

The second program, the Fault Tree Synthesis Program, would have the general function of creating a fault tree from a system digraph. This general function is further delineated in Figure 2-2.

Completion of this specification requires statements concerning

- computer system(s) for which the code is to be developed,
- language (e.g., FORTRAN),
- size of problem to be handled (e.g., number of nodes and links in system digraph, number of nodes in fault tree).

Figure 2-1. Functions of the System Digraph Construction Program.

* FTAP is the existing Fault Tree Analys_is Program [8]

Figure 2-2. Functions of the Fault Tree Synthesis Program.

3.0 TECHNICAL SPECIFICATIONS

The technical specifications for the software system should include mathematical descriptions of the physical system being represented, and of the necessary algorithms and procedures. Not all necessary descriptions are presently available. Here, we discuss the system representation, e.g., the system digraph, the representation of fault trees, and discuss the procedures in so far as they have been developed to this time. This latter discussion focuses on the Lapp-Powers approach to fault tree synthesis.

3.1 Representation of the Material Control System

The material control system is conveniently represented in the form of a system flow sheet. A portion of the system flow sheet used in the Test Bed analysis is included here as Figure 3-1.

The system flow sheet consists of modules interconnected by specified directional streams. In Figure 3-1, some of the modules are:

SENS	a sensor
1B	labelled 1B
MNWAY	a manway
1B	labelled 1B
SEC	security
BOOTH	booth

Also there, stream 64 connects "SENS 1B" to "MNWAY 1B". Note that each stream connects a pair of modules in a particular direction.

The system flow sheet can then be represented by:

- (1) a list of modules, and
- (2) a list of connecting streams.

Figure 3-1. A portion of the system flow sheet for the Test Bed analysis.

The module list can take the form:

module #	module type	module label
1	SENS	1B
2	MNWAY	1B
3	SENS	BO
:	:	:
21	SEC	GATE
22	EXIT	GATE

The list of connecting streams can take the form:

stream #	upstream module			downstream module		
	#	type	label	#	type	label
1	21	SEC	GATE	22	EXIT	GATE
:	:	:	:	:	:	:
63	2	MNWAY	1B	3	SENS	BO
64	1	SENS	1B	2	MNWAY	1B

3.2 Digraph Representation

The events of the MC&A system, including adversary actions and actions of a responding security force, are represented at the most elementary level by sets of relationships pertaining to a module. This set constitutes a unit model. The representation is taken in the form of a directed graph (digraph) with labelled nodes and edge relationships.

The modeling of each module then takes the form of a unit model digraph. Through the use of the system flow sheet, a system digraph is assembled from the unit model digraphs.

In this section, we will first describe the digraph representation of modules, and then describe the process of assembling the system digraph.

3.2.1 Unit Model Digraphs

The elementary building block of a unit model digraph is the relationship between two variables, represented as an edge label:

The edge relationship describes the relationship of the values of "var 2" as they are caused by the values of "var 1".

For our purposes, variable values and their interpretations are as follows:

- $+1;\tau$ a positive disturbance, first occurring at time τ
- 0 a normal value
- $-1;\tau$ a negative disturbance, first occurring at time τ .

Certain variables are further restricted to values of +1 and 0 only.

For example, the variable "presence of an adversary in a manway" would have a value of +1 to indicate presence, and 0 to indicate absence.

The simplest edge relationships, and their interpretations, are as follows:

<u>op</u>	<u>interpretation</u>
+1	var 2 has the same value as var 1, i.e., a disturbance is propagated unchanged
0	var 2 has the value 0 for any value of var 1
-1	var 2 has the value equal to -1 times the value of var 1, i.e., a positive disturbance is transformed into a negative disturbance, and a negative disturbance is transformed into a positive disturbance.

Such a simple relationship is represented graphically in the form

If $\text{var } 1 = \text{val}$, we write $\text{var } 2 = \text{op} * \text{val}$.

Our interest in the MC&A system involves consideration of time delays in the effect of one variable on another. This can be represented graphically in the form

where τ is the time delay. The interpretation of this edge relationship is that "var 2" takes a value determined by "var 1" and "op" τ units of time

after var 1 takes a non-zero value. Generally, if $\text{var } 1 = \text{val}; \tau_1$, and the edge relationship is $\text{op}; \tau_2$, then $\text{var } 2 = \text{op} * \text{val}; \tau_1 + \tau_2$.

For example, consider the movement of an adversary through a manway. Suppose it takes 10 seconds (units of time) to traverse the manway. Then this can be represented by

If the adversary appears at the upstream end of the manway at time 20, this relationship implies that the adversary appears at the downstream end at time 30.

In addition, the relationship between variables may be conditional; this is generally represented in the form

where "cond" represents the conditions under which the indicated relationship holds. Conditions are phrased in terms of a logical expression involving one or more variables. Simple examples are "var 3 = +1", and "fire present in manway". The latter could also be phrased as "var 3 = +1" if var 3 is taken to represent the presence of fire in the manway. More complex conditions are also possible. In general, any logical expression involving variable values is possible.

To illustrate these relationships, consider the modeling of the movement of an adversary and security forces through a manway. The variables involved are:

- AP1 adversary presence at upstream end of manway
- AP2 adversary presence at downstream end of manway
- SP1 security force presence at upstream end of manway
- SP2 security force presence at downstream end of manway
- MB1 blockage of manway (= +1 for presence, 0 for absence).

The complete unit model digraph for the manway is then

Note that absence of a condition in the edge relationship is taken to mean "under normal circumstances", which here means that the manway is not blocked.

In representing unit model digraphs, it is convenient to arrange the parameters according to the output variables (AP2 and SP2, in the above illustration); for each output variable, a list of edge-oriented

data is given, each element of the list consisting of the input variable and the edge relationship. For the last example, we have

input variable	input variable	edge relationship
AP2	AP1	+1;10
	AP1	0: MB1 = +1
SP2	SP1	+1;10
		0: MB1 = +1

It can be seen from the system flow sheet, Figure 3-1, that modules of common type appear frequently. For example, there are several sensor modules and several manways. It will be convenient, in designing the software, to provide for generic types of modules, so that specific instances of each module can be represented as specializations of the general model. These specializations will involve:

- (1) specification of variables involved,
- (2) selection of certain parameters, e.g., values of time delay in the edge relationships.

In the code generated by Lapp-Powers, this approach has been accommodated by the implementation of distinct subroutines for each module type. Their code includes modules suitable for fault analysis of chemical processing systems. A portion of their code for a pipe is included here as Figure 3-2.

While one can, to a certain extent, use the functional relationships embodied in their code, the requirements for the MC&A system are sufficiently different that entirely new code is called for. The concept of individual subroutines for each module type should be retained.

Figure 3-2. (Three pages following) A portion of the Lapp-Powers subroutine for creating the unit model digraph for a pipe.

```

0100 SUBROUTINE PIPE1
0200 INTEGER CNP(2000),CN(5000),DGP(1000),DG(5000),STRM(1000),
0300 IISTR(1000),ISTRP(100),JD(2,1000),IP(50)
0400 INTEGER MX,C,VN,V(13)
0500 INTEGER PT(6),MDT(6),MSTR(6)
0600 INTEGER CL1(5)
0700 DIMENSION VML(6)
0800 DIMENSION VAL(1000),NODE(1000),IC(7),IDT(1000),GN(5000)
0900 LOGICAL LF,LCN
1000 REAL*8 UNT, USRID(100)
1100 COMMON /A27/ USRID,VAL,IDT,CN,IBGN,IEND,JEQ,Ik,I,NDGP,
1200 IIPN,MD,ITP,ISN,IMN,IMX,MAX,NCN,NDG,NBASE,IEQ,IAD,MP,
1300 ZMF,MC,CNP,CN,JD,DGP,DG,STRM,IISTR,ISTRP,IP
1400 EQUIVALENCE (VAL,NODE)

```

```

1500 C
1600 C DIGRAPH PIPE MODEL
1700 C STREAMS 1 & 2 ARE PROCESS STREAMS ENTERING AND EXITING
1800 C
1900 C
2000 C

```

```

2100 C
2200 C 1 ----->| |-----> 2
2300 C |-----|
2400 C
2500 C
2600 C
2700 C

```

```

2800 C VARIABLES TREATED IN THIS MODEL ARE THE FOLLOWING:

```

```

2900 C 1) "P" - PRESSURE
3000 C 2) "T" - TEMPERATURE
3100 C 3) "M" - MASS FLOWRATE
3200 C 4) "C" - CONCENTRATION
3300 C 5) "MR" - REVERSE MASS FLOW RATE
3400 C 6) "DEN" - DENSITY
3500 C 7) "LIQF" - LIQUID FRACTION
3600 C 8) "PLUG" - PIPE PLUGGED
3700 C 9) "TSUR" - TEMPERATURE OF THE SURROUNDINGS
3800 C 10) "NFLO" - NO FLOW CONDITION IN THE PIPE
3900 C 11) "LEAK" - LEAK TO OR FROM THE SURROUNDINGS
4000 C 12) "DTUB" - DIP TUBE
4100 C 13) "DT22" - DIP TUBE IN NON-STREAM OUTLET -?
4200 C
4300 C

```

```

4400 C PARAMETERS FOR THIS MODEL ARE THE FOLLOWING

```

```

4500 C TSUR = TEMPERATURE OF THE SURROUNDINGS
4600 C PSUR = PRESSURE OF THE SURROUNDINGS
4700 C CSUR = CONCENTRATION OF THE SURROUNDINGS
4800 C DENS = DENSITY OF THE SURROUNDINGS
4900 C LFSR = LIQUID FRACTION OF THE SURROUNDINGS
5000 C (ALL SURROUNDINGS PARAMETERS HAVE THE FOLLOWING VALUES

```

```

0600 C PIPE1,PIPEA,1,2
06200 C *PARAM
06300 C TSUR=0,PSUR=0,CSUR=0,DENS=0,LFSR=0,NLF=1
06400 C *TOP
06500 C P-1=10
06600 C *END
06700 C
06800 C
06900 DATA C/'C',MX/2,VN/13,UNT/'PIPE1',IFR/'FIRE',
07000 DATA V/'P','T','M','C','MR','DEN','LIQF','PLUG','TSUR','NFLD',
07100 I 'LEAK','DTUB','DTBZ'/
07200 DATA PT/1,2,3,4,5,MDT/'TSUR','PSOR','CSUR','DENS','LFSR','NLF'/
07300 I ,MSTR/5*-JEQ/
07400 ICI = JD(1,1)
07500 ICI = IDT(ICI)
07600 DECODE(5,100,ICI) (CL1(M),M=1,5)
07700 100 FORMAT(5A1)
07800 IF(CL1(1).EQ. C) V(4) = ICI
07900 NPT = 5
08000 MPT = 5
08100 CALL DATA(MDT,MSTR,VML,PT,RPT,MPT)
08200 CALL CODE(V,VN,UNT,MX,KODE)
08300 LF = .FALSE.
08400 LCN = .FALSE.
08500 GO TO (11,12,21,22,31,32,41,42,51,52,61,62,71,72,81,81,91,91,
08600 1 101,101,111,111,121,121,131,131),KODE
08700 C
08800 C STATEMENT NUMBER VARIABLE
08900 C -----
09000 C 11 PRESSURE IN
09100 C 12 PRESSURE OUT
09200 C 21 TEMPERATURE IN
09300 C 22 TEMPERATURE OUT
09400 C 31 MASS FLOW IN
09500 C 32 MASS FLOW OUT
09600 C 41 CONCENTRATION IN
09700 C 42 CONCENTRATION OUT
09800 C 51 REVERSE MASS FLOW IN
09900 C 52 REVERSE MASS FLOW OUT
10000 C 61 DENSITY IN
10100 C 62 DENSITY OUT
10200 C 71 LIQUID FRACTION IN
10300 C 72 LIQUID FRACTION OUT
10400 C 81 PLUG
10500 C 91 LEAK TO OR FROM THE SURROUNDINGS
10600 C 101 TEMPERATURE OF THE SURROUNDINGS
10700 C 111 NO FLOW CONDITION IN PIPE
10800 C 121 DIP TUBE
10900 C 131 DIP TUBE IN NON-STREAM OPENING
11000 C
11100 C PRESSURE IN
11200 11 GAIN = 1.0
11300 J = STREAM 2
11400 CALL ICASN(PRESSURE,J,1,PLUG,-JEQ,1,1,ICI)
11500 CALL GASN(GAIN,PRESSURE,J,IC,LF,4,LCN)
11600 ->IF(PSUR.EQ. 1.) GAIN = -1.0
11700 CALL GASN(GAIN,LEAK,-JEQ,IC,LF,1,LCN)
11800 GAIN = 10.0
11900 CALL GASN(GAIN,PLUG,-JEQ,IC,LF,1,LCN)
12000 GAIN = 0.0

```

12100	- - -	CALL GASN(GAIN,PLUG,-JEQ,IC,LF,3,LCN)	
12200		RETURN	
12300	C PRESSURE OUT		
12400	12	GAIN = 1.0	
12500		J = STREAM 1	
12600		CALL ICASN(PRESSURE,J,1,PLUG,-JEQ,1,1,IC)	
12700		CALL GASN(GAIN,PRESSURE,J,IC,LF,4,LCN)	
12800		IF(NORM LIQF .EQ. 1.) GO TO 13	
12900		CALL ICASN(TSURR,-JEQ,1,LIQFRACT,J,10,1,IC)	
13000		CALL GASN(GAIN,TSURR,-JEQ,IC,LF,3,LCN)	
13100		LCN = .TRUE.	
13200	13	CALL ICASN(TSURR,-JEQ,1,FLOW,J,-10,1,IC)	MULTIPLE
13300		CALL GASN(GAIN,TSURR,-JEQ,IC,LF,3,LCN)	EDGE
13400		LCN = .FALSE.	
13500		→ IF(PSUR .EQ. 1.) GAIN = -1.0	
13600		CALL GASV(GAIN,LEAK,-JEQ,IC,LF,1,LCN)	
13700		GAIN = -10.0	
13800		CALL GASN(GAIN,PLUG,-JEQ,IC,LF,1,LCN)	
13900		GAIN = 0.0	
14000		CALL GASN(GAIN,PLUG,-JEQ,IC,LF,3,LCN)	
14100		RETURN	
14200	C TEMPERATURE IN		
14300	21	GAIN = 1.0	
14400		J = STREAM 2	
14500		CALL ICASN(TEMP,J,1,REVFLOW,J,1,1,IC)	
14600		CALL GASN(GAIN,TEMP,J,IC,LF,3,LCN)	
14700		IF(TSUR .EQ. 0.) GAIN = -1.0	
14800		CALL ICASN(LEAK,-JEQ,1,REVFLOW,J,1,1,IC)	
14900		CALL GASN(GAIN,LEAK,-JEQ,IC,LF,3,LCN)	
15000		GAIN = 0.0	
15100		CALL ICASN(TEMP,J,1,REVFLOW,J,0,1,IC)	
15200		CALL GASN(GAIN,TEMP,J,IC,LF,3,LCN)	
15300		RETURN	
15400	C TEMPERATURE OUT		
15500	22	GAIN = 1.0	
15600		J = STREAM 1	
15700		CALL GASN(GAIN,TEMP,J,IC,LF,1,LCN)	
15800		CALL GASN(GAIN,TSURR,-JEQ,IC,LF,1,LCN)	
15900		→ IF(TSUR .EQ. 1.) GAIN = -1.0	
16000		CALL GASN(GAIN,LEAK,-JEQ,IC,LF,1,LCN)	
16100		GAIN = 0.0	
16200		CALL ICASN(TEMP,J,1,FLOW,STREAM 2,-10,1,IC)	
16300		CALL GASN(GAIN,TEMP,-JEQ,IC,LF,3,LCN)	
16400		CALL ICASN(TEMP,J,1,REVFLOW,J,1,1,IC)	
16500		CALL GASN(GAIN,TEMP,J,IC,LF,3,LCN)	
16600		RETURN	
16700	C MASS FLOW IN		
16800	31	GAIN = 1.0	
16900		J = STREAM 2	
17000		CALL ICASN(FLOW,J,1,PLUG,-JEQ,1,1,IC)	
17100		CALL GASN(GAIN,FLOW,J,IC,LF,4,LCN)	
17200		GAIN = 0.1	
17300		→ IF(PSUR .EQ. 1.) GAIN = -1.0	
17400		CALL GASN(GAIN,LEAK,-JEQ,IC,LF,3,LCN)	
17500		GAIN = -10.0	
17600		CALL GASN(GAIN,PLUG,-JEQ,IC,LF,2,LCN)	
17700		GAIN = 0.0	
17800		CALL GASN(GAIN,PLUG,-JEQ,IC,LF,3,LCN)	
17900		RETURN	
18000	C MASS FLOW OUT		

3.2.2 System Digraph

The system digraph is composed from the unit model digraphs by using the module specifications (to obtain specific unit model digraphs from the generic models) and the connectivity information contained in the system flow sheet. The result can be represented in the same form as a unit model digraph, as discussed in the previous section.

Before proceeding with this discussion, it is useful to distinguish amongst nodes in a digraph (both unit model digraphs and the system digraphs). Those nodes with no inputs are termed primal nodes; the variable represented by a primal node is termed a primary or basic variable. All other nodes are internal nodes; the variable represented by an internal node is termed an intermediate variable.

Values for basic variables must ultimately be specified; values for intermediate variables are implied by the values for the basic variables.

For each unit model digraph, the primal nodes are termed the module inputs. The remaining nodes are generally outputs, but it may be necessary to separately distinguish intermediate nodes in a unit model digraph which are not outputs. In any case, the term outputs will mean nodes (variables) which can serve as inputs to other modules, or which can serve as top events.

Consider the connection of two modules, say module A to module B by stream n:

In connecting these two modules, inputs for module B will be identified as certain outputs of module A. In designing the representation of modules, it will be necessary to provide a means for making this identification. In designing the software for making the connecting of modules, it will be necessary to make certain compatibility checks for necessary matches of inputs and outputs.

These requirements can be largely met by

- (1) categorizing variables by type, and
- (2) specifying variables appearing in a unit model by type and local label,

The connection process then involves

- (1) matching variables of like type, and
- (2) assigning global names to variables.

A simple scheme for global names was used by Lapp and Powers; the variable type name was appended with the stream number which labels the connection.

To illustrate this process, we will make use of the following variable types

AP	adversary presence
SP	security force presence
SS	sensor signal

Next, we introduce the following unit model digraph for a sensor

The manway module previously provided is to be connected to this sensor module (SENS 1),

In this case, the inputs to the sensor module can easily be associated with outputs of the manway module by relating variable types. If the stream number assigned is 10, say, the system digraph is represented as follows:

<u>output variable</u>	<u>input variable</u>	<u>edge relationship</u>
SS 1 [*]	AP 10	+1;2
	AP 10	0 : sensor failure
	SP 10	+1;2
	SP 10	0 : sensor failure
AP 10	AP 1 [*]	+1;10
	AP 1 [*]	0 : MBl = +1
	SP 1 [*]	+1;10
	SP 1 [*]	0 : MBl = +1

^{*}This is a (temporary) local number.

Note that in this connection process, any input which is not matched becomes a primal node for the system digraph. An important aspect of the system digraph construction software will be the characterization of nodes (as intermediate and primal, for example) to assist in checking out the specification of the system. Some primal nodes may be a result of an inadequate specification of unit modules.

3.2.3 Digraph Conventions

Where only one edge is incident to a node, the meaning of the digraph is clear. Certain conventions are required, however, where several edges are incident to node, particularly where one or more edges relate to disturbances and others relate to control responses.

To illustrate, consider the situation

If both var 1 and var 2 represent disturbances, the usual convention is that simultaneous disturbances which could create a normal condition for var 3 are ignored, as they would be extremely rare events. Further, this convention yields more conservative results. Thus, in working back from var 3, a normal condition (i.e., $\text{var } 3 = 0$) occurs only if both var 1 and var 2 are normal. The situation $\text{var } 1 = +1$, $\text{var } 2 = -1$, for example, is not considered.

Now suppose var 1 represents a disturbance, while var 2 is part of a control loop, that is, a loop designed to cancel the effect of a disturbance. In assessing how a disturbance could arise in var 3, one must now posit a failure in the loop of which var 2 is a part, as well as a disturbance in var 1. This sort of analysis involves more than the

immediate subnode var 2; it necessarily involves the specification of conventions pertaining to the analysis of loops which must ultimately be incorporated into the fault tree synthesis algorithm.

Lapp and Powers [3] have developed such conventions and corresponding procedures in their work. These conventions must be reexamined to determine necessary extensions to handle the more complex edge labels described in Section 3.2.2. That appears to be a reasonably straightforward task. (For example, delay around a feedback loop can render it inoperative.)

A third situation of interest occurs where var 1 corresponds to a disturbance as caused by an adversary action, and var 2 corresponds to an adversary action intended to cancel the effect of the disturbance as seen in var 3. In this instance, a normal condition var 3 can occur either due to no disturbance, or a disturbance together with an adversary induced feed forward cancellation of the effect of the disturbance. This convention is not at present embodied in the Lapp-Powers procedure [3]. In fact, it appears to be necessary to explicitly distinguish between the control-related feed forward paths and the adversary-related feed forward path due to the difference in conventions pertaining to them.

These considerations reveal the need to extract loops from the system digraph to allow for application of procedures which embody these conventions. Further, it should be noted that representation of adversary actions to cancel effects of disturbances is typically done after the system digraph representing the material control system is constructed, and analyzed for loop structure. These actions are in fact represented by the introduction of additional feed forward loops explicitly labelled as corresponding to adversary-related actions.

It is for these reasons that the software is envisioned to consist of three functionally distinct parts: (1) system digraph construction, (2) loop analysis, and (3) fault tree synthesis. With this separation, the capability for analyzing the system digraph prior to introduction of the adversary manipulation of variables is provided. Also, it provides for a manual labelling of loops to distinguish control-related and adversary-related loops.

3.3 Fault Tree Synthesis Procedures

A fault tree provides a compact representation of all combinations of events which lead to the occurrence of a specified (top) event. For example, consider the system represented by the digraph developed at the end of the last section (involving a sensor and a manway). How could the sensor become activated at time 0? The combination of events which produce this result is represented by the fault tree of Figure 3-3.

The fault tree to be produced by the synthesis procedure is of the form developed by the procedure of Lapp and Powers [3,5]. The tree involves nodes of two types: (1) gate nodes, which are further developed, and which correspond to "intermediate variables", and (2) terminal nodes, which correspond to "basic or primary variables"; and involves the operators "AND" and "OR". The "top" node is the gate node with no outgoing arcs. A fault tree can be generally represented in the form of a list.

<u>gate node</u> <u>variable</u>	<u>operator</u>	<u>immediate</u> <u>subnodes</u>
gate var 1	op 1	subnode list 1
⋮	⋮	⋮
gate var n	op n	subnode list n

Figure 3-3. A fault tree.

Synthesis of the fault tree from a system digraph involves

- (1) identification of gate node logical variables from the system digraph as logical expressions of the form "var = value",
- (2) identification of immediate subnodes to a gate node by examination of the links in the system digraph.

In the simplest instances, as illustrated by the last example, these identifications are immediate and the connecting operation is "OR".

Complications enter due to

- (1) presence of conditions on the edges in the system digraph,
- (2) existence of parallel paths (feed forward) or feedback loops in the system digraph.

Further, where edge conditions involve intermediate variables (rather than primal variables), "boundary conditions" must be recorded and accounted for as further feed forward paths may be involved.

Edge conditions of simple form involve only primal variables, i.e., variables which do not require further development as they are to be directly specified as to value in the subsequent fault tree analysis. For example,

Parallel paths arise either as a result of the presence of options to an adversary, or options to the MC&A response system, or as competing paths involving the adversary on one hand and the MC&A response on the other. Within the Lapp-Powers methodology [3], there is a technique for handling two parallel competing paths (a disturbance path and a control path, in their terminology). Extensions are necessary, however, to handle more general situations. Further, the element of time delay which appears in the system digraphs presented here must be handled. An option, however, is to merely detect situations for which an (automated) algorithm is not available, and to allow for manual analysis.

Feedback loops generally involve the use of a sensed signal to effect a control response in order to return an abnormal situation to a normal one. Again, Lapp and Powers [3] have developed an algorithm for developing nodes on a feedback loop. However, that algorithm does not handle general networks, e.g., nested loops. Further, the element of time delay must also be addressed. It appears, however, that the modeling of the MC&A system may be accomplished without having to involve feedback loops. If that proves to be so, the necessary algorithm development would be significantly simplified.

3.4 Finding Loops

In the Lapp-Powers procedure, and as expected in the extension of that procedure to the problem at hand, it is necessary to recognize feedback loops and feed forward loops (actually, alternative paths between two nodes). Here, we discuss procedures related to this subproblem.

As a preliminary to this, and also because the result is directly useful, nodes of the system digraph are first classified according to the four types of nodes. This classification is based on the edges incident to and incident from the node:

- (1) System input nodes have no edges incident to them. These nodes correspond to primal or basic variables.
- (2) System output nodes have no edges incident from them.
- (3) Intermediate nodes have edges both incident to and incident from them.
- (4) An isolated node has no incident edges.

Identifying the node type for each node in the system digraph is a simple process. First, each node is labelled as an isolated node. Next, the digraph representation, organized by output node is scanned. For each output node, the following changes are made:

<u>current label</u>	<u>new label</u>
isolated	system output
system output	no change
system input	intermediate
intermediate	no change

For each input node, the following changes are made:

<u>current label</u>	<u>new label</u>
isolated	system input
system output	intermediate
system input	no change
intermediate	no change

As we have previously noted, it appears that the model, i.e., the system digraph, can be restricted to having no feedback loops, or in standard graph terminology, no directed cycles. There is a simple scheme for testing whether this is so.

The following is taken from [6]:

"Proposition

"A directed graph is acyclic if and only if its nodes can be numbered in such a way that for all arcs (i,j) , $i < j$.

"In order to show that the nodes of an acyclic graph can be so ordered, one first observes that there is at least one node with in-degree zero. Such a node is found, numbered 1, and all arcs incident from the node are deleted. A node with in-degree zero is found in the resulting subgraph. This node is numbered 2, and all arcs incident from it are deleted, etc.

"This procedure can be implemented by a computation whose complexity is $O(n^2)$. We suppose that the graph is described by its adjacency matrix A and that the rows and columns of this matrix are ordered according to the given arbitrary numbering of the nodes. Let $v(j)$ denote the new number of node j .

"Initially $d_j^{(in)}$ is computed for all nodes j , by forming the sum of the entries in column j of matrix A . A node k for which $d_k^{(in)} = 0$ is found, and $v(k)$ is set to 1. The in-degrees are revised by subtracting the entries in row k of A , and the process is repeated. This is summarized below.

"Renumbering the Nodes of an Acyclic Digraph

" Step 0 (Start)

$$\text{"Set } d_j^{(in)} = \sum_{i=1}^n a_{ij}, j = 1, 2, \dots, n.$$

"Set $N = (1, 2, \dots, n)$

Set $m = 1$.

Go to Step 2.

"Step 1 (Revision of In-Degrees)

"Set $d_j^{(in)} = d_j^{(in)} - a_{kj}$, for all $j \in N$

"Step 2 (Detection of Node with Zero In-Degree)

Find $k \in N$ such that $d_k^{(in)} = 0$. If there is no such k , stop; the digraph is not acyclic.

Set $v(k) = m$.

Set $m = m + 1$

Set $N = N - k$.

If $N \neq \phi$, go to Step 1. Otherwise, stop; the computation is completed.

"The complexity of the computation is estimated as follows.

Step 0 requires only n^2 addition. Step 1 requires n subtractions. Step 2 requires at most n comparisons to determine k , and various other operations independent of n . Step 0, which is $O(n^2)$, is performed exactly once. Steps 1 and 2, which are both $O(n)$, are performed at most n times. It follows that the overall complexity is $O(n^2)$."

To illustrate this procedure, consider first the digraphs of Figure 3-4; the link incidence arrays for these two digraphs are provided in Table 3-1.

We now go through the steps for the first digraph:

Step 0: $d_1 = 3, d_2 = 2, d_3 = 1, d_4 = 0, d_5 = 0, d_6 = 0$

$N = (1, 2, 3, 4, 5, 6)$

$m = 1$

(a) Digraph without a feedback loop (cycle).

(b) Digraph with a feedback loop (cycle).

Figure 3-4. Digraph examples for feedback loop detection procedure.

Table 3-1. Link incidence arrays for digraph examples for feedback loop detection procedures.

(a) Digraph without a feedback loop (cycle).

$$[a_{ij}] = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(b) Digraph with a feedback loop (cycle).

$$a_{ij} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Step 2: $k = 4$

$$v(4) = 1$$

$$m = 2$$

$$N = (1, 2, 3, 5, 6)$$

Step 1: $d_1 = 3, d_2 = 1, d_3 = 2, d_4 = 1, d_5 = 0, d_6 = 0$

Step 2: $k = 5$

$$v(5) = 2$$

$$m = 3$$

$$N = (1, 2, 3, 6)$$

Step 1: $d_1 = 3, d_2 = 1, d_3 = 0, d_4 = 0, d_5 = 0, d_6 = 0$

Step 2: $k = 3$

$$v(3) = 3$$

$$m = 4$$

$$N = (1, 2, 6)$$

Step 1: $d_1 = 2, d_2 = 0, d_3 = 0, d_4 = 0, d_5 = 0, d_6 = 0$

Step 2: $k = 2$

$$v(2) = 4$$

$$m = 5$$

$$N = (1, 6)$$

Step 1: $d_1 = 1, d_2 = 0, d_3 = 0, d_4 = 0, d_5 = 0, d_6 = 0$

Step 2: $k = 6$

$$m = 6$$

$$v(6) = 5$$

$$N = (1)$$

Step 1: $d_1 = 0, d_2 = 0, d_3 = 0, d_4 = 0, d_5 = 0, d_6 = 0$

Step 2: $k = 1$

$v(1) = 6$

$N = \phi$ stop.

The reordering is defined by $v(k)$, and is illustrated in Figure 3-4(a) by numbers in boxes.

Now we consider application of the procedure to the second digraph example:

Step 0: $d_1 = 1, d_2 = 1, d_3 = 1, d_4 = 1, d_5 = 1, d_6 = 1$

$N = (1, 2, 3, 4, 5, 6)$

$m = 1$

Step 2: no k can be found such that $d_k = 0$, hence procedure stops.

Our need is to explicitly display feedback loops and multiple paths. A cursory review of the literature did not reveal a candidate procedure. Hence, we endeavored to develop one especially suited to the problem at hand. The following describes a procedure for this purpose which is based upon the technique of successively removing intermediate nodes.

As a preliminary, we introduce the notion of a bundle of edges. A bundle is a set of edges which connect the same pair of nodes. Such bundles are typically present in the system digraph as a device to represent different relationships between variables under different conditions. We generalize this so that bundles can include parallel paths generated in the node reduction process.

Edge conditions can also lead to loops and feedforward paths. To account for this, additional edges can be created. For example, if the relationship between var 1 and var 2 involves a condition on var 3, say,

as illustrated below;

for our purposes, one replaces this by

i.e., an additional edge is created between var 3 and var 2. This additional edge can be labelled so that its source, i.e., as a condition on the relationship between var 1 and var 2, can be recovered.

In the node reduction process, we need only retain connectivity information. However, we will associate eliminated nodes with bundles. Hence, edges will be in the form

output node, input node, bundle #

and bundles will be in the form

bundle #, ordered list of included edges and bundles
, ordered list of nodes on the path represented by
this edge.

The overall process is an iterative one, as illustrated by the flow chart of Figure 3-5. The major steps of the procedure are:

Figure 3-5. Flow chart for the loop analysis procedure.

- (1) elimination of an intermediate node with corresponding generation of new edges and bundles,
- (2) combination of parallel edges to form a new bundle, and
- (3) identification of feedback loops amongst the newly defined edges,

These steps are repeated until the list of intermediate nodes is exhausted.

Consider a general intermediate node, to be eliminated

Elimination of this node can be accomplished by introducing a set of new edges and bundles, which collectively represent all the paths from the input nodes $(i_1, \dots, i_n)$ to the output nodes $(k_1, \dots, k_m)$ via the intermediate node j. This set of edges is

<u>output node</u>	<u>input node</u>	<u>bundle #</u>
i_1	k_1	l_{11}
$\vdots$	$\vdots$	$\vdots$
$\vdots$	k_m	l_m
$\vdots$	$\vdots$	$\vdots$
i_n	k_1	l_{n1}
	$\vdots$	$\vdots$
	k_m	l_{nm}

The newly defined bundles are

<u>bundle #</u>	<u>included bundles</u>
l_{11}	bundle for edge (i_1, j) , node j , bundle for
$\vdots$	edge (j, k_1)
$\vdots$	
l_{nm}	bundle for edge (i_n, j) , node j , bundle for
	edge (j, k_m)

After this construction, parallel paths and loops may arise. Parallel paths are handled by defining a new bundle which is the collection of bundles involved in the parallel paths. Loops, which are manifested by the existence of an edge with the input and output nodes identical, are taken care of by deleting the edge and transferring the bundle index to a loop data structure. Examination of the bundle data structure will yield the details of the nodes on the loop.

4.0 SOFTWARE FUNCTIONAL SPECIFICATIONS

This section presents a preliminary version of the software functional specifications.

4.1 Decomposition of Functions

There are three major, distinct functions of the software system

- (1) system digraph construction,
- (2) digraph analysis (finding loops, characterizing nodes, etc.),
- (3) fault tree synthesis.

The first two are naturally grouped into a one program. It may be desirable to separate the third function into a second program, since it is generally desirable to examine the system digraph and the analyses of it before proceeding to the synthesis of the fault tree.

For purposes of the remainder of our discussion, it is assumed that the first two functions will be accomplished by the program SYSDIG (System Digraph), and the third function will be accomplished by the program FTSYN (Fault Tree Synthesis).

4.2 Data Structures

Data structures are identified below for each of the two programs in the form of groups of data items which are functionally related. The next step is to delineate individual data items in each group, and to associate FORTRAN variable names with each data item. A preliminary version is presented in Section 4.2.3, where individual data items for each group are listed.

4.2.1 SYSDIG Data Structures

Input data

- module definition parameters (a set for each module)
- system connectivity parameters (a coding of the system flow sheet)
- digraph analysis option parameters
- output option parameters

Working data structures

- *system digraph representation*
- node characterizations
- loop representations.

4.2.2 FTSYN Data Structures

Input data

- system digraph representation
- node characterizations
- loop representations
- top event(s)
- output option parameters

Data Group: system connectivity parameters

Items in this group are organized by the stream (or system flow sheet link) connecting a pair of modules:

- stream number
- input module designation (form to be decided, either module index or module name)
- output module designation.

Data Group: system digraph representation

Items in this group are organized by link:

- output node variable (global designation)
- input node variable (global designation)
- edge operator
- edge delay
- edge first condition variable (global designation)
- edge first condition value
- $\vdots$
- edge n^{th} condition variable (global designation)
- edge n^{th} condition value.

Data Group: node characterizations

This group provides connections between the system level description and module descriptions, and defines node types. These items are organized by global node:

- number of global nodes
- local global node correspondence

Working data structures

- fault tree representation
- auxiliary data structures necessary for synthesis algorithm.

4.2.3 Details of Data Structures

Data Group: module definition parameters

Within this group, there is firstly a subgroup which relates to permanent data, not being specific to particular module specifications.

- module type names (e.g., manway, sensor, etc.)
- variable type names (e.g., adversary presence)
- number of module types
- number of variable types
- module parameter names
- number of module parameter types.

The second subgroup provides for specifications of modules:

- module type indices (i.e., first module is type n_1 , second module is type n_2 , etc.)
- variable type indices, organized by module (i.e., for first module, variable types m_{11} , m_{12} , etc.)
- module link parameters, organized by module, then link within the module
- local variable designations, organized by module, then by variable within the module.

- global node type (output only, intermediate, input)
- special characteristics, i.e., first node upstream of branch point for parallel (feed forward) paths, or loop index on which this node is located.

Data Group: loop representations

These items are organized by loop:

- number of parallel (feed forward) paths
- number of feedback loops
- feed forward path definitions
 - index of path collection
 - upstream node
 - downstream node
 - nodes on first path
 - nodes on n^{th} path
- feedback loop definition
 - index of loop
 - normal loop gain and delay
 - ordered list of nodes on loop

organized by node pair

organized by loop

Data Group: fault tree representation

Since the fault tree is an output which also serves as a portion of the input to another computer program (FTAP), it is desirable to make the structure of this data group compatible with the input requirement of FTAP.

Items in this group are organized by branch in the tree:

- gate node name
- operation
- list of immediate subnode names.

Data Group: auxiliary data structures necessary for synthesis algorithm

The details of this data group can be developed when the details of the synthesis algorithm are finalized. If the Lapp-Powers procedure, as it presently exists, could be used, no auxiliary structures would be necessary, as the fault tree would be built up recursively and would itself serve as the working data structure.

There will be a need to maintain logical consistency in connection with "boundary conditions", i.e., conditional statements involving intermediate variables. How this will be done has not yet been determined, so details of the necessary data structure cannot be spelled out.

4.3 Code Structure

The following pages present the first few levels of code structure for each of the two programs. Further delineation requires prior definition of all procedures and would proceed in concert with development of the software technical specifications.

4.4 Module Descriptions

Preliminary to coding, detailed descriptions of each module must be developed. A suggested format for these module descriptions is illustrated in Figure 4-1, where the description for the top-most routine of the program SYSDIG is provided.

A second example is provided in Figure 4-2. This is the module description for MODDEF.

In the course of the development of detailed software technical specifications, these module descriptions would be extended by listing individual data items (by FORTRAN name) serving as inputs and outputs, and by converting the module description into either pseudo-code or a flow chart.

Module No: 1.0

Module Name: SYSDIG

Module Function: This is the executive routine for the system digraph construction and analysis program.

Module Description: The system digraph construction and analysis program is organized into four software functions. The first, data input and diagnostics, is accomplished by a call to module 2.0, INPSDG. The second, construction of the system digraph, is accomplished by a call to module 3.0, SDGCON. The third, system digraph analysis, is accomplished by a call to module 4.0, SDGANL. The fourth, the output of the system digraph and its characterizations, is accomplished by a call to module 5.0, OUTSDG.

Module Inputs: None

Module Outputs: None

Figure 4-1. Module description for SYSDIG.

Module No: 2.2

Module Name: MODDEF

Module Function: Process module specifications.

Module Description: This routine processes all input data related to the specification of modules to produce unit model digraphs defined in terms of local variables. Input data are processed in sets, each set corresponding to a single module specification. The first step in the processing of data in one set is to extract the module type index, by a call to module 2.2.1, MODTYP. This index is then used to control the call of the proper subroutine specific for the designated module. These steps are repeated until all sets of data for module specifications are processed.

Module Inputs: Input card data for module specifications, general module descriptions.

Module Outputs: Module definition parameters.

Figure 4-2. Module description for MODDEF.

4.5 Software Standards

Large software projects require standardization in the code and documentation in order to

- (1) facilitate management as the project proceeds,
- (2) facilitate coordination of effort by different software designers and programmers,
- (3) facilitate debugging and testing,
- (4) improve maintainability,
- (5) facilitate future revisions.

Necessary elements of the software standards include

- (1) documentation, as described in this report, and
- (2) coding standards for variable naming, COMMON areas, source style, and in code documentation.

Figure 4-3 presents a module which conforms to one particular standard for source coding. The important point is not that this form be used, but rather that some effective standard be adopted.

Figure 4-3. (Three pages following) A module which conforms to a standard for source coding.

```

C
C
C --- CODED      08-10-78 BY H. PAYNE
C --- REVISED      BY
C
C --- TITLE - MOVE VEHICLES ON FREEWAY SUBNETWORK - MODULE 3.2.4.2
C
C --- FUNCTION - CALLS ROUTINES NECESSARY TO MOVE VEHICLES ON FREEWAY
C ---              SUBNETWORK FOR ONE TIME SLICE, AND ACCUMULATES
C ---              EXITING FLOW RATES FOR EACH LINK
C
C --- ARGUMENTS - NONE

```

THIS ROUTINE MOVES ALL VEHICLES FOR ONE TIME SLICE. WITHIN THIS ROUTINE, OUTPUT VOLUMES ARE ACCUMULATED FOR ALL DYNAMIC LINKS. TO MOVE THE VEHICLES, FIRST MODULE 3242.1 IS CALLED TO COMPUTE EXITING VOLUMES AND DOWNSTREAM DENSITIES. NEXT, MODULE 3242.2 IS CALLED TO COMPUTE ENTRY VOLUMES AND SPEEDS. NEXT, MODULE 3242.3 IS CALLED TO COMPUTE NEW LINK DENSITIES AND SPACE-MEAN-SPEEDS. FINALLY, MODULE 3242.4 IS CALLED TO ACCUMULATE VEHICLES EXITING THE SUBNETWORK.

FREELO - MODULE 3.2.4

```
C      -----  
C  
C      DNVLIN - MODULE 3242.1  
C      UPVLSF - MODULE 3242.2  
C      UPDNHF - MODULE 3242.3  
C      UECNTE - MODULE 3242.4
```

	<u>SYMBOL</u>	<u>DEFINITION</u>
ILD	LINK	NUMBER OF FIRST DUMMY LINK
IL	LINK	NUMBER
INSTPS	NUMBER	OF INTEGRATION STEPS IN A TIME SLICE
IT	TIME	SLICE INDEX
NSTEP	NUMBER	OF INTEGRATION STEPS PER TIME SLICE
TDLNK	TOTAL	NUMBER OF DYNAMIC LINKS IN SUBNETWORK
TTLNK	TOTAL	NUMBER OF LINKS IN SUBNETWORK
VOLA	EXIT	FLOW RATE FOR AUTOS AND TRUCKS
VOLB	EXIT	FLOW RATE FOR BUSES
VOLC	EXIT	FLOW RATE FOR CAR POOLS
VOLINA	TOTAL	FLOW RATE OF AUTOS AND TRUCKS ENTERING LINK
VOLINB	TOTAL	FLOW RATE OF BUSES ENTERING LINK
VOLINC	TOTAL	FLOW RATE OF CAR POOL VEHICLES ENTERING LINK
VOLOUA	TOTAL	FLOW RATE OF AUTOS AND TRUCKS EXITING LINK
VOLOUB	TOTAL	FLOW RATE OF BUSES EXITING LINK
VOLOUC	TOTAL	FLOW RATE OF CAR POOL VEHICLES EXITING LINK

```

C
C      IMPLICIT INTEGER (A-Q, S-V, X), REAL (R, Z), LOGICAL (W, Y)
C
COMMON /NSTEP / NSTEP (1)
COMMON /TDLNK / TDLNK
COMMON /TTLNK / TTLNK
COMMON /VOLA / VOLA
COMMON /VOLB / VOLB
COMMON /VOLC / VOLC
COMMON /VOLINA/ VOLINA(1)
COMMON /VOLINB/ VOLINB(1)
COMMON /VOLINC/ VOLINC(1)
COMMON /VOLOUA/ VOLOUA(1)
COMMON /VOLOUB/ VOLOUB(1)
COMMON /VOLOUC/ VOLOUC(1)
COMMON /YTRACE/ YTRACE

C
C
C
C ----- FOR EACH DYNAMIC LINK, FIRST COMPUTE UPSTREAM
C ----- SPEEDS AND ENTERING FLOW RATES
C
      DO 20 IL = 1,TDLNK
        IF (YTRACE) CALL TRACE ( 440070, IL, 0, 0, 0 )
        CALL UPVLSP (IL)

C
C ----- FOR EACH INTEGRATION STEP, FIRST RESET EXITING FLOW RATES
C
        INSTPS = NSTEP(IL)
        DO 10 IT = 1,INSTPS
          VOLOUA(IL) = 0
          VOLOUB(IL) = 0
          VOLOUC(IL) = 0

C
C ----- UPDATE DENSITIES, SPEEDS AND EXITING FLOW RATES FOR THIS LINK
C
          CALL DNVLBN( IL )
          CALL UPBNSP( IL )

C
C ----- ACCUMULATE VEHICLES EXITING LINK
C
          VOLOUA(IL) = VOLOUA(IL) + VOLA
          VOLOUB(IL) = VOLOUB(IL) + VOLB
          VOLOUC(IL) = VOLOUC(IL) + VOLC

        10      CONTINUE

C
C ----- SCALE NUMBER EXITING TO GET FLOW RATES
C
          VOLOUA(IL) = VOLOUA(IL) / INSTPS
          VOLOUB(IL) = VOLOUB(IL) / INSTPS
          VOLOUC(IL) = VOLOUC(IL) / INSTPS
          IF (YTRACE) CALL HTRACE ( 440071, VOLOUA(IL), VOLOUB(IL),
            1      VOLOUC(IL), 0 )
        20      CONTINUE

C
C ----- UPDATE EXITING FLOW RATES FOR DUMMY LINKS
C

```

IF (TTLNK .EQ. TDLNK)

GO TO 40

ILD = TDLNK + 1

DO 30 IL = ILD,TTLNK

VOLOUA(IL) = VOLINA(IL)

VOLOUB(IL) = VOLINB(IL)

VOLOUC(IL) = VOLINC(IL)

IF (YTRACE) CALL TRACE (440072, IL, 0, 0. 0)

IF (YTRACE) CALL HTRACE (440073, VOLOUA(IL), VOLOUB(IL),

1 VOLOUC(IL), 0)

30 CONTINUE

40 CONTINUE

C

C ----- ACCUMULATE VEHICLES EXITING SUBNETWORK

C

CALL UPCNTE

RETURN

END

5.0 CONCLUSIONS

Here, we summarize conclusions in connection with the two major objectives addressed in this report.

Lapp-Powers approach to system digraph generation

- In order to represent the MC&A system with time delays, extensions to the elements of the system digraph used to represent the system are necessary. A simple means for incorporating an appropriate extension in the edge relationship has been presented here.
- Conventions are necessary in connection with the system digraph in order to properly reflect cancelling effects of feed forward paths and feedback loops. The standard conventions have been identified here. Procedures for these exist in the work of Lapp and Powers, except for the one (which should be straightforward to develop) pertaining to adversary-related feed forward paths.
- The system digraph can be constructed in a hierarchal fashion, in a way similar to that developed by Lapp and Powers. The necessary modeling extensions do not appear to be a hindrance to this approach. This hierarchal approach involves, firstly, construction of unit model digraphs, then connecting these digraphs (through the use of system connectivity data) to form the system digraph. The software design should involve separate routines for distinct model types, and the existing code of Lapp and Powers can be used as a guide for this purpose.

Fault tree synthesis

- The initial required step in a Lapp-Powers type of approach to fault tree synthesis is the identification of alternative forward paths and feedback loops. The sparse nature of the graph involved here, and the special constructs (edge relationships, conventions regarding cancelation of disturbances) seem to indicate that a procedure specific to this problem be developed. Hence, procedures were presented in outline form for, firstly, identifying node types (input, intermediate, and output), and then, in a node elimination technique, for identifying alternative forward paths and loops. More effort should be expended in this area to (1) explore the literature for relevant procedures, and (2) develop the procedures outlined here or otherwise identified.
- The Lapp-Powers procedure as it presently exists is not capable of handling the MC&A assessment problem. First, the feature of time delay must be accommodated. That appears to involve only a straightforward extension of the existing procedure. The more serious remaining technical problems are: (1) handling of general, or at least more complex, system digraphs, in so far as nested loops and complex multiple paths are concerned, and (2) handling of "boundary conditions", i.e., edge relationships which involve intermediate (non-primal) digraph variables. These boundary conditions can lead to complex multiple paths.

- Only a very high level software design for fault tree synthesis could be presented, as all the important decisions concerning data structures and code structures depend on details of the procedure. If the existing Lapp-Powers procedure were adequate, this design would not be complex, and the data structures presented here would suffice. But that is not the case, and further developments will be necessary.

6.0 REFERENCES

1. Fussell, J.B. and R.E. Lambert, "Quantitative Evaluation of Nuclear System Reliability and Safety Characteristics," IEEE Trans. Rel., Vol. R-25, No. 3, August 1976, pp. 178-183.
2. Lambert, Howard, "United States Safeguards Effectiveness Assessment Methodologies," seminar notes, Lawrence Livermore Laboratory, 1978.
3. Lambert, Howard, "Digraphs and Automated Fault Tree Analysis," seminar notes, Lawrence Livermore Laboratory, 1978.
4. Lambert, Howard, "Getting Down to the Real Nitty Gritty," or "Crucial Modeling Issues in the Digraph -- Fault Tree Analysis and the Matrix Procedure, Lawrence Livermore Laboratory, MC78-868, September 11, 1978.
5. Lapp, Steven A. and Gary J. Powers, "Computer-Aided Synthesis of Fault Trees," IEEE Trans. Rel., Vol. R-26, April 1977, pp. 2-13.
6. Lawler, Eugene L., Combinatorial Optimization: Network and Matroids, Holt, Rinehart and Winston, Inc.
7. Shaelwitz, Joseph A., Steven A. Lapp and Gary J. Powers, "Fault Tree Analysis of Sequential Systems," Ind. Eng. Chem., Process Des. Dev., Vol. 16, No. 4, 1977, pp. 529-549.
8. Willie, Randall, R., "Computer-Aided Fault Tree Analysis," Report ORC 78-14, Operations Research Center, University of California, Berkeley, August 1978.

"Work performed under the auspices of the
U.S. Department of Energy by the Lawrence
Livermore Laboratory under contract number
W-7405-ENG-48."

Reference to a company or product name does not imply approval or recommendation of the product by the University of California or the U.S. Department of Energy to the exclusion of others that may be suitable.

NOTICE

"This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Department of Energy, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately-owned rights."