Computer-aided Synthesis of Fault-trees

Steven A. Lapp
Gary J. Powers

Key Words–Computer-aided, Fault-tree synthesis, Computer program.

Reader Aids–
Purpose: Advance state of the art
Special math needed: None
Results useful to: Reliability engineers and theoreticians

Abstract–An algorithm is presented for the synthesis of fault-trees. The fault-tree is deduced directly from a digraph (directed graph) model of the system being analyzed. The digraph describes the normal, failed, and conditional relationships which exist between variables and events in the system. A computer program which uses this algorithm is illustrated for a chemical processing system.

1. INTRODUCTION

Fault-tree analysis was originally developed by Bell Telephone Laboratories in 1961. Bell used the technique to evaluate the launch control system of the Minuteman Missile [1]. In applying the fault-tree method, one begins by identifying some undesirable event associated with the system. This event is termed the top event. For the Minuteman, examples might be 1) Inadvertent firing of the missile or 2) Failure to launch when called upon to do so. Once an undesirable event has been chosen, the analysis proceeds by asking "What could cause this?" In answering this question, one generates other events connected by the logic operators AND, OR, EOR (EOR is an abbreviation for "Exclusive Or"; it is the same as XOR which is often used. This operator has two inputs and will produce a TRUE output if one, but not both, of them is TRUE.) Successive events are then developed in a similar manner. The analysis terminates when events are encountered which cannot or need not be developed further. These events are called primal events (primals). The logic structure relating the top event to the primals is the fault tree.

Once constructed, the fault-tree can be of considerable value in determining the paths whereby primal events can propagate through the system to cause the top event. Algorithms currently exist [2] that determine which primal events, or combinations of primal events, will cause the top event for a given fault-tree. These sets of events are termed cut sets. If occurrence-rate data are available for the primal events, a number of useful statistics can be computed [3]. Among these are the probability and rate of occurrence of the top event, the probability that a given cut set will occur, and the relative *s*-importance of each primal event to the top event. Such statistics can provide valuable information as to which part of the system should be modified to decrease the probability of the top event. An excellent example of the use of the fault-tree technique and resulting statistical analysis can be found in the Rasmussen Report on Nuclear Reactor Safety [4].

While much of the statistical and cut set analysis has been automated, actual construction of the fault-tree is usually done by hand. Manual construction of the tree can be extremely time consuming; a substantial fraction of 25 man-years of effort was required in the Rasmussen Study [4]. In addition to the time involved, the possibility exists that different analysts will produce different fault-trees [5] either by incorrect logic or omission of certain events. A computer aided synthesis technique would prove valuable in alleviating both of the above problems. Any system of constructing fault-trees should have the following 4 characteristics.

1) Handle complex systems efficiently. A complex system is one with feedback or feedforward loops and over 20 components.
2) Consider system topology as well as actual components in constructing the tree.
3) Handle multivalued logic; i.e. consider the direction and magnitude of deviations in process variables in addition to component failures.
4) During fault-tree construction, make checks to ensure consistency among events. For example, an increase in the temperature of some stream cannot be caused by a simultaneous decrease in the same stream's temperature. (More on this in the next section.)

Although other criteria certainly exist, these four represent basic requirements for any synthesis system. Table 1 summarizes previous and current work in terms of these criteria.

The issue of a formal synthesis method has been addressed by Fussell [6]. His technique, known as the synthetic tree model, uses transfer functions as models for component failures to construct the final fault-tree. The system under analysis is segmented by the user into parts (coalitions) and it is through this grouping that consistency requirements arise. The intermediate event under consideration along with any consistency requirements determine which component transfer function is used.

Taylor's method [7] uses algebraic models for components with qualifiers to indicate which equations describe the operation or failure of the component. These qualified equations are then written for each component and the resulting collection forms the system model. This model can then be used to determine the consequences of any deviation in the input variables. The method involves more cause-consequence than fault-tree analysis.

TABLE 1
Summary of Fault-tree Synthesis Work

	Complex Systems	Topology Considered	Multivalued Logic	Consistency Checks	Computer Program	Analyses Performed (1975)	Largest Problem (No. Gates in tree) (1975)
Fussell (Fault-tree analysis)	No	Partially (Component Coalitions determined by user.)	No	Yes (Not Allowed Boundary Conditions)	Yes (Certain Types of Electrical Systems)	<5?	22
Taylor (Cause-consequence analysis)	No	Yes	Yes	Yes	No	<5?	Not Applicable (Equivalent Tree would contain about 15 gates.)
Tompkins & Powers (Fault-tree Analysis)	No	Partially	No	No	No	<5?	15
Lapp & Powers	Yes	Yes	Yes	Yes	Yes	>1∅	143

Tompkins and Powers [8] have suggested a method using input-output models for equipment. These models convey information regarding variable relationships when the components are working as well as the effects of component failures. Construction of the fault-tree begins with the identification of important deviations in process variables. The process is then searched for sources of these deviations and it is through this search that the fault-tree is built.

The methods discussed so far have dealt with organizing construction of the fault-tree. Actual use of these techniques is just beginning and there are many important problems to solve. Our work in this area is described below. In order to develop a program which synthesizes fault-trees, it is first necessary to develop some means of system modeling which is suited for computer processing. This representation must also be general so that any type of process can be analyzed. With the system thus modeled, the next step is to develop an algorithm for fault-tree synthesis. These problems of suitable representation and subsequent fault-tree generation form the core of this paper.

2. FAULT-TREE SYNTHESIS–A PROTOCOL

As an example of how one might manually generate fault-trees, consider the flowsheet shown in Fig. 1. The function of this process is to cool a hot nitric acid stream before reacting it with Benzene to form Nitrobenzene. One top event for the system is a high temperature in the nitric acid reactor feed since this could cause a reactor runaway. Consider constructing a fault-tree for this event. The following notation is used

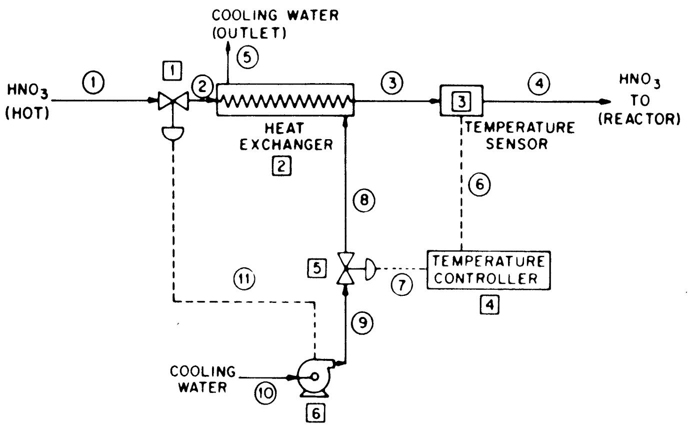


Fig. 1. Nitric acid cooler with temperature feedback and pump-shutdown feedforward loops.

to describe deviations in process variables: *T*, *P*, *M* denote deviations in temperature, pressure, and mass flow respectively; "+" and "–" denote directions of the deviation (positive or negative); ∅, 1, 1∅ denote magnitudes of the deviation (none, moderate, or very large). Hence, to represent a large decrease in the mass flow rate of stream 8, write M8(–1∅).

An engineer would begin by asking "What could cause *T*4(+1)?" Since Stream 4 is directly connected to Stream 3 through the temperature sensor, he might reason that *T*3(+1) is the cause; so the first step in constructing the tree would be to place an OR gate under *T*4(+1) with *T*3(+1) as its input:

```
                T4(+1)
                  *
                  OR
                  *
                T3(+1).
```

His next question would then be "What could cause *T*3(+1)?" The answer in this case is more complex. From a knowledge of heat exchanger performance, four causes emerge: *M*2(+1), *T*2(+1), *M*8(−1), and *T*8(+1). A question now arises as to how these causes should be logically connected. Our engineer might reason that since any one of these events would result in *T*3(+1), the proper operator is an OR gate; so his resulting structure would be as follows.

```
                T4(+1)
                  *
                  OR
                  *
                T3(+1)
                  *
                  OR
                  *
   *******************************************
   *        *                      *      *
 M2(+1)   T2(+1)                M8(−1)  T8(+1).
```

He then checks this tree to see if it is correct. The tree suggests that *M*2(+1) would cause *T*4(+1). Obviously this isn't true since the negative feedback control loop would act to cancel the effect of *M*2(+1). Similar arguments can be made to show that *T*2(+1) and *T*8(+1) will not cause *T*4(+1). On the other hand, *M*8(−1) will cause the top event. The reason for this is that *M*8 is itself part of the temperature control loop. Under conditions when *T*4(+1), the appropriate response for the control loop is *M*8(+1). *M*8(−1) is an indication that the control loop is actually working to promote the top event instead of cancelling it out, and this is why *M*8(−1) will cause *T*4(+1). This discussion suggests that if either *M*2(+1), *T*2(+1), or *T*8(+1) is to cause *T*4(+1), the control loop must either take no action to cancel the disturbance or promote it. If deviations in *M*2, *T*2, or *T*8 are very large, however, the control loop might not be able to cancel them out. Hence, these large deviations would cause *T*4(+1). With these arguments, the above tree results.

```
                 T4(+1)
                   *
                   OR
                   *
                 T3(+1)
                   *
                   OR
                   *
     *************************************
     *       *          *         *        *
    AND   M2(+1∅)   T2(+1∅)   T8(+1∅)   M8(−1)
     *
     *****************
     *               *
     OR        IMPROPER CONTROL
     *           LOOP ACTION
 *****************
 *       *       *
M2(+1)  T2(+1)  T8(+1)
```

He continues by focusing attention on *M*8(−1). Assuming that the cooling water control valve is AIR TO OPEN, possible causes are *P*7(−1) or *P*9(−1). Suppose, however, that this assumption is incorrect, i.e. the valve has been installed backward. In this case *P*7(+1) will be the cause but note that *P*7(+1) is a normal response to *T*4(+1). Hence, three causes really exist: *P*7(−1), *P*9(−1), and VALVE REVERSED. Consider that *P*7(−1) and VALVE REVERSED are on the control loop, and apply the logic suggested when expanding *T*3(+1); then the following expansion for *M*8(−1) results.

```
                         M8(−1)
                           *
                           OR
                           *
            ***********************
            *          *          *
           AND      P9(−1∅)      EOR
            *                     *
   **************                **********
   *            *                *        *
 P9(−1)   IMPROPER CONTROL     VALVE    P7(−1)
            LOOP ACTION       REVERSED .
```

The EOR gate is necessary because *P*7(−1) and VALVE REVERSED together cancel one another out resulting in *M*8(+1).

Proceeding with *P*7(−1), our engineer might trace this to *P*6(−1) and subsequently to *T*4(−1). This, however, cannot be since he knows *T*4(+1) to be the case. *T*4(−1) is a consistency violation and must be dropped from consideration. These are always found when negative feedback loops (such as control loops) are encountered in a process. One must be careful to exclude any events which are consistency violations. If *T*4(−1) is the only cause of *P*6(−1), then *P*6(−1) must also be dropped, and so on with *P*7(−1). *P*7(−1), however, can also be traced to a low air-pressure at the controller. Continuing the analysis, he might trace disturbances in *M*2, *T*2, and *T*8 to similar disturbances in *M*1, *T*1, and *T*1∅. Decreases (−1 or −1∅) in *P*9 can be traced to either decreases in *P*1∅ or a shutdown of the pump. Although PUMP SHUTDOWN stops the flow of cooling water, it also activates a system which stops the flow of hot nitric acid. Hence, in order for PUMP SHUTDOWN to cause *T*4(+1), a failure of the nitric acid shutdown system is also required. With this, his final tree becomes the following.(See at the top of next page.)

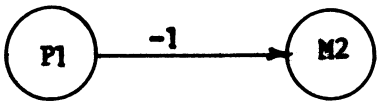
The analysis presented here is highly simplified in that many events have been excluded from the tree. It does, however, demonstrate the basic method of manual fault-tree synthesis. Later in this paper, a more detailed fault-tree for this same system will be constructed using a computer program.

```
                                      T4(+1)
                                        *
                                        OR
                                      T3(+1)
                                        *
                                        OR
                                        *
                 ***********************************************
                 *       *        *        *                   *
                AND   M2(+1Ø)   T2(+1Ø)   T8(+1Ø)              *
                 *                                             *
                 ***************                               *
                 *             *                            M8(-1)
                OR       IMPROPER CONTROL                      *
                 *         LOOP ACTION                         *
          *******************                                  *
          *      *      *                                      OR
       M2(+1)  T2(+1)  T8(+1)                                  *
                                 *********************************** *
                                 *          *          *             *
                              LOSS OF     P9(-1)    P1Ø(-1Ø)         *
                            INSTRUMENT      *                        *
                                AIR         OR                     VALVE
                                            *                     REVERSED
                        **********************
                        *                   *
                       AND                 AND
                        *                   *
          ******************                ******************
          *                *                *                *
        PUMP            FAILURE          P1Ø(-1)          IMPROPER
      SHUTDOWN       OF NITRIC ACID                     CONTROL LOOP
                    SHUTDOWN SYSTEM                        ACTION
```

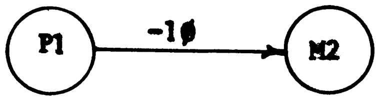
3. PROCESS REPRESENTATION–DIGRAPHS

In order to synthesize fault-trees automatically, it is first necessary to develop a satisfactory representation of the system under study. This representation must be general so that any type of process may be analyzed. At the same time, it must be suited for ease of computer processing. One means of system representation which meets both of the above standards involves the use of digraphs (directed graphs). A digraph is a set of nodes connected by directed edges. The nodes of digraphs used in fault-tree synthesis represent process variables and certain types of failures. In the case of chemical systems, these would be temperatures, pressures, flow rates, etc. Relations among the various nodes are embodied in the edges connecting them. If a deviation in one variable causes a deviation in a second variable, then a directed edge is drawn from the node representing the first variable to the node representing the second. A number is assigned to the edge depending on the direction and magnitude of the second deviation relative to the first. If a moderate deviation in the first variable causes a moderate deviation in the second, a value of "1" is assigned to the edge. On the other hand, if the second deviation is very large compared to the first, a value of "1Ø" is assigned. If the second deviation is very small compared with the first, no edge connects the nodes. The sign of the number reflects the relative direction of the deviations. If they are in similar directions, the number is positive, otherwise it is negative.

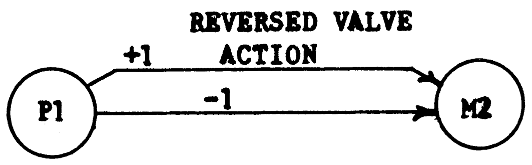
As an example, consider a control valve with spring action AIR TO CLOSE. We wish to represent the relationship between the air pressure on the valve (denoted by *P*1) and the flow rate of fluid through the valve (*M*2). Since a positive deviation in *P*1 causes a negative deviation in *M*2, the digraph is as follows.



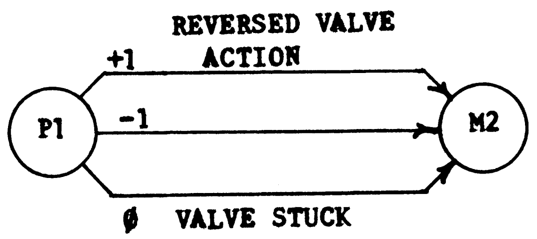
Suppose the valve has quick-closing characteristics. Then a positive deviation in *P*1 causes a very large negative deviation in *M*2, and the digraph is the following.



The number associated with the edge can be interpreted as a partial derivative ($\partial M2/\partial P1$) and is termed the gain between the variables. Thus far, we have concerned ourselves only with the usual relationship between variables. How do we model failures which alter these usual relationships? One answer is simply to include additional edges between the variables that represent the failed gains. Consider the valve again and suppose we wish to include the failure REVERSED VALVE ACTION. In this case, increasing $P1$ will increase $M2$. The following digraph embodies the additional failure:

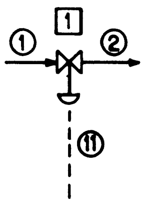


In addition, if we wish to include the failure VALVE STUCK (increasing $P1$ has no effect on $M2$), then we would use the following digraph.

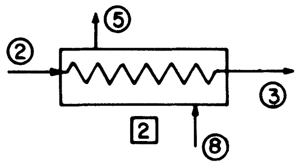


Edges with zero gains are drawn only if these edges represent failures. If zero-gain represents the normal relationship, no edge is drawn between the nodes.

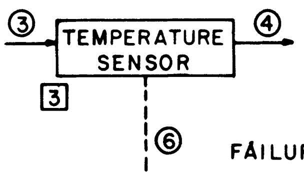
Let us now attempt to construct a digraph for an entire process using the methods discussed previously. Given a process, we first number the streams and assign nodes to represent the stream variables. The problem now becomes one of connecting the nodes with appropriate edges. In order to do this, we need models for the equipment in the process. Recall that in the previous example it was a model of the control valve which allowed us to relate $P1$ to $M2$. Equipment models must represent how variables are related both when the equipment is working and when it is failed. Consider now, the nitrobenzene process discussed in the previous section. Fig. 2a and 2b list models for the various pieces of equipment in this process. The models consist of a table with process variables listed against one another. The gain between two variables is found by locating the row and column corresponding to the first and second variables respectively. The gain is the entry in this location. The direction of the edge is from the row variable to the column variable. For example, the first row of the first control-valve model indicates that an edge of gain 1 is drawn from $P1$ to $M2$ while no edge is drawn from $P1$ to $T2$. Other relationships are obtained in a similar manner. Failures which change the relationship between variables are listed under the usual relationships. The first model indicates that two directed edges are drawn from $P11$ to $M2$. One should have a gain of $-1\emptyset$ (normal) while the other should have a gain of $+1\emptyset$



OUT / IN	M 2	T 2
P 1	+ 1	0
P 11	- 10	0
T 1	0	+ 1
FAILURE : REVERSED VALVE ACTION	P11 TO M2 GAIN CHANGES TO +10	

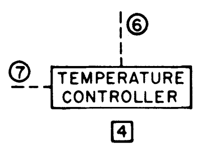


	M 5	T 3	ETC
M 2	0	+ 1	
T 2	0	+ 1	
M 8	+ 1	- 1	
FAILURE: EXTERNAL FIRE	0	+ 1	

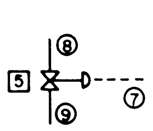


	T 4	P 6
T 3	1	1
FAILURE: SENSOR BROKEN	CHANGE T3 TO P6 GAIN TO 0 (ZERO)	

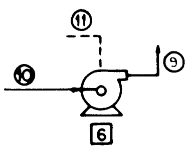
Fig. 2a.



	P 7
P 6	+ 1
FAILURE : EXTERNAL FIRE	+ 1
LOW AIR PRESSURE	- 1
INSTALLED WITH REVERSE ACTION	CHANGE P6 TO P7 GAIN TO - 1
CONTROLLER BROKEN	CHANGE P6 TO P7 GAIN TO 0 (ZERO)



	M 8
P 9	+ 1
P 7	+ 1
FAILURE : REVERSED VALVE ACTION	CHANGE P7 TO M8 GAIN TO - 1



	P 9	P 11
P 10	+ 1	0
FAILURE : PUMP SHUTDOWN	- 10	+ 1
LINE 11 PLUGGED AT PUMP	CHANGE SHUTDOWN TO P11 GAIN TO 0 (ZERO)	

Fig. 2b.

Figs. 2a & 2b. Input-output models for equipment in the nitric acid cooler process.

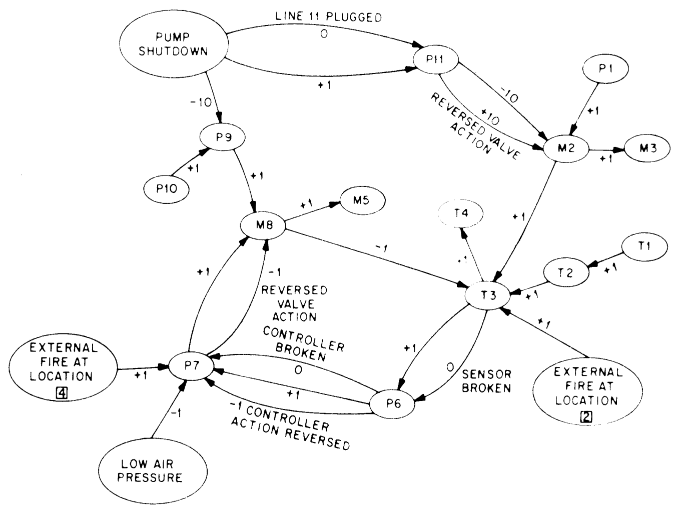


Fig. 3. Digraph for the output variable *T*4 in the nitric acid process.

(REVERSED VALVE ACTION-failure). These models also contain information on another type of failure. This second type of failure is one which causes a deviation in some other process variable instead of changing a relationship. As an example, consider the failure EXTERNAL FIRE in the heat exchanger model. The model indicates that the gain between EXTERNAL FIRE and *T*3 is +1. On a digraph, EXTERNAL FIRE would be assigned a node and an edge of gain +1 drawn between it and *T*3. Using the models listed in Figs. 2a and 2b, we can now construct a digraph for the nitrobenzene process, Fig. 3. The digraph is really a map of how the variables in this process are related. It reflects the behavior of the equipment involved as well as general system topology. In the next section, we shall see the role this structure plays in the development of a fault-tree synthesis algorithm.

4. AN ALGORITHM FOR FAULT-TREE SYNTHESIS

The problem of fault-tree synthesis can be formulated using a state space representation. Problems represented in the state space are solved by transforming a given initial state into a desired goal state. Transformation is accomplished through the use of appropriate operators which transform one state into another. The initial state in fault-tree synthesis is a definition of the top event along with a description of the process. The process description is in the form of a digraph. The goal state is a fault-tree connecting the top event to events which are not developed further. These events are called primal events. The task now becomes one of defining operators necessary to perform the synthesis. In order to do this, consider the methods used in manual fault-tree construction (See Section 2.) In developing each event, one asks "What could cause this?" On a digraph, this corresponds to asking "Which nodes are inputs to the node representing the current event?" Once these events are identified, the next task is to determine how they ought to be interconnected logically to form part of a fault-tree. In answering the question, "What could cause this?", it seems natural to connect the events by using an OR gate. As noted previously, however, this is sometimes incorrect. The problem lies in the fact that the use of an OR gate represents only a partial answer to what the causitive events are. In stating that one event causes another, it is necessary to include the qualification: Nothing else happens which will cancel the original effect. This important assumption, although not explicitly stated, is made at each step in the synthesis of the fault-tree.

Constructing a fault-tree requires explicitly stating this assumption at each step. Consider the previous example. In particular, examine the causes of *T*3(+1). Recall that this event can be traced to four causes; *M*2(+1), *T*2(+1), *M*8(−1), *T*8(+1). If any one of these events is to cause *T*3(+1), then none of the other variables can deviate in such a manner as to cancel the effect. In expanding *T*3(+1), one can use the OR gate except that now the inputs to the gate are '*M*2(+1) AND (NOT(*T*2(−1) OR *M*8(+1) OR *T*8(−1)))' instead of '*M*2(+1)' alone, '*T*2(+1) AND (NOT(*M*2(−1) OR *M*8(+1) OR *T*8(−1)))' instead of '*T*2(+1)' alone, etc. Hence, replace each of the NOTs by its Boolean equivalent structure, i.e., 'NOT *T*2(−1)' is the same as '*T*2(+1) OR *T*2(∅)', etc. This considerably complicates the tree and a simplification is necessary. *T*2(∅) can be traced to *T*1(∅) which means *T*1 UNCHANGED. *T*1, however, is a primal variable (deviations in it are primal events). Since primal events are normally considered to have low probabilities, *T*1(∅) is nearly always true. A similar argument shows that *M*2(∅) and *T*8(∅) are also true with high probabilities. Hence NOT *T*2(−1), NOT *M*2(−1), and NOT *T*8(−1), are also true with high probabilities and can be eliminated from the AND gate.

Now consider *M*8(∅). Unlike the other events discussed, *M*8(∅) is not normally true. Recall that since *T*3(+1) is true (this is the event being developed), *M*8 will increase due to the action of the control loop. Thus NOT *M*8(+1) is normally not true and remains on the AND gate explicitly, or in its equivalent form of '*M*8(−1) OR *M*8(∅)'. The event *M*8(−1) is not even part of an AND gate since all of its COMPLEMENTARY NOTs are normally true. *M*8(−1) is an indication that the control loop actually is responsible for the disturbance since *M*8 is the variable which ought to increase to cancel *T*3(+1).

This assumption of ALL OTHER THINGS BEING THE SAM is crucial. In the following developments it is assumed true for all events that are not interconnected by negative feedback or feedforward loops in the digraph model. Hence, it is quite important that the model capture any interactions which might occur in the system.

Consider now, the same events with '1∅' values. As with '1' values one can argue that *T*2(∅), *M*2(∅), and *T*8(∅) are normally true. Since a '1∅' indicates a very large disturbance, assume that the control loop cannot cancel it. In terms of these variables, this means that *M*8 cannot increase sufficiently to overcome the disturbance. Stated otherwise, NOT *M*8(+1∅) is normally true. Hence, all of the NOTs associated with the '1∅' values are normally true, so no AND gates are necessary. All of these results are embodied in the following tree for *T*3(+1).

```
                        T3(+1)
                          *
                          OR
                          *
            *************************************
            *      *        *         *        *
           AND  M2(+1∅)  T2(+1∅)   T8(+1∅)  M8(-1)
            *
            *************************
            *                       *
           OR                       OR
            *                       *
    *****************          ***************
     *          *     *         *             *
M2(+1)     T2(+1)  T8(+1)    M8(-1)         M8(∅)
```

This tree is the same as that previously developed manually except that '*M*8(−1) OR *M*8(∅)' has replaced IMPROPER CONTROL LOOP ACTION. This is fine since one of the causes of *M*8(−1) OR *M*8(∅) is IMPROPER CONTROL LOOP ACTION because the loop ought to send *M*8(+1).

From a functional point of view, a disturbance propagates through the control loop if:

1) The disturbance is extremely large in magnitude; or
2) The disturbance is caused by the control loop itself; or
3) An external disturbance enters the system and the control loop does not act to cancel it.

These statements are general and apply to any negative feedback loop (of which a control loop is a good example). Hence, these statements can be used to determine the causes of a disturbance when a negative feedback loop is involved. The generalized operator used for negative feedback loop variables is shown here.

```
                           E
                           *
                           OR
                           *
              **************************
              *            *            *
             AND         Large       Loop Variable
              *        Disturbance   Causes Disturbance
              *        Enters Loop
      **********************
      *                    *
External Disturbance    Loop Variable Fails
    Enters Loop         To Cancel Disturbance
```

(*E* represents the event currently being developed.)

This operator can then be used whenever a negative feedback loop is encountered. The structure of this operator arises from considering how the NOT condition is handled on a negative feedback loop.

Now consider an operator for negative feedforward loops. This is another case where the NOT OTHER THINGS CHANGING assumption is important. A negative feedforward loop is a structure within a process whereby one process variable affects another with opposite gains on different paths within the process. An example of this is the effect of PUMP SHUTDOWN on *T*3. On one path, PUMP SHUTDOWN tends to send *T*3(+1∅) while on another *T*3(−1∅). In the former case the sequence 'PUMP SHUTDOWN–*P*9(−1∅) –*M*8(−1∅) –*T*3(+1∅)' occurs while in the latter case 'PUMP SHUTDOWN–*P*11(+1) –*M*2(−1∅) –*T*3(−1∅)'. Thus to trace *T*3(+1∅) to a PUMP SHUTDOWN, one must include the additional condition that disturbances on the other paths of the loop are not present. In this particular problem, the condition is embodied in NOT *M*2(−1∅). Hence, the tree in this case looks like the following one.

```
                      T3(+1∅)
                         *
               Intermediate Variables
                         *
                        AND
                         *
          ****************************
          *                          *
     Pump Shutdown                   OR
                                *************
                                *           *
                             M2(+1∅)      M2(∅).
```

The generalized feedforward loop operator is then

```
                            E
                            *
                           AND
                            *
******************************************
*                                        *
Disturbance Which              Disturbances On
Propagates Down Both           Alternate Paths
Loop Paths                     Fail To Cancel
                               One Another.
```

(*E* represents the event currently being developed.)

This operator is applied when the point common to both sides of the loop is reached.

For events which involve neither negative feedback nor feedforward loops, the operator used is quite simple. It is just an OR gate since, in the absence of the negative loops, all of the NOT conditions are normally true.

Generalizations of these operators have been made to handle variables which are on combinations of negative feedback and feedforward loops. Given these operators, the local cause and effect relations in the process, and a list of the negative feedback and feedforward loops, it is possible to construct fault-trees. The digraph is useful here. Local cause and effect relations are embodied in the edges connecting the nodes. Negative feedback and feedforward loops can be determined by traversal of the graph.

Another important feature in fault-tree synthesis is consistency. Consistency means that two mutually exclusive events cannot occur at the same time. For example, consistency requires that *T*3(+1) not be traced to *T*3(−1) nor to *T*3(∅). Any inconsistent events that are generated in the course of the synthesis must be deleted. One might conclude then that any generated events must be checked for consistency against all events which have been developed. Fortunately, this is not true. Consistency is only a problem when feedback or feedforward loops are involved. This is the only way the variable can come back on itself in the graph. Hence, the only events which need to be checked are those which involve either of these types of loops. In the case of feedback loops, the event being developed is stored for later consistency checks. The event common to all paths of the feedforward loop is stored when the feedforward operator is applied.

A general algorithm is presented below; it is based on the concepts developed in this section. In the next section, this algorithm is applied to parts of the nitric acid problem. After that, a computer program based on the algorithm is described.

Fault-tree Synthesis Algorithm

1. Generate digraph and find all negative feedback and feedforward loops.
2. Select node representing top event.
3. Determine local causes of this event by noting the inputs to the node of the digraph.
4. Delete any local causes which violate consistency.
5. Select the appropriate operator depending on whether negative feedback or feedforward loops pass through the current node. Use this operator to connect logically the remaining local causes. If negative feedback or feedforward loops are involved, store the appropriate event for later consistency checks.
6. Select a node corresponding to an undeveloped event and return to step 3. If only primal events remain, stop.

5. ALGORITHMIC FAULT-TREE SYNTHESIS—AN EXAMPLE

The algorithm will be demonstrated with parts of the nitric acid problem. The digraph is shown in Figure 3. First, notice the one negative feedback loop and one negative feedforward loop. The feedback loop is comprised of the variables *T*3-*P*6-*P*7-*M*8-*T*3. The two paths of the feedforward loop are 'PUMP SHUTDOWN-*P*11-*M*2-*T*3' and 'PUMP SHUTDOWN-*P*9-*M*8-*T*3'. The top event is *T*4(+1); so begin at the node labeled *T*4. In the section on manual synthesis, the events *T*4(+1) and *T*3(+1) were analyzed. Consider the analysis from that point by examining *M*8(−1). *M*8 has three inputs, two from node *P*7 and one from *P*9; *M*8 is on a negative feedback loop and *P*7(−1) and REVERSED VALVE ACTION are also on the feedback loop. *P*9(−1) is an external cause since it is not on the feedback loop. Even though *P*9 is on one path of the negative feedforward loop, do not employ the feedforward operator since it is not common to both loop paths. At this point, only one event need be checked for consistency and that is *T*3(+1) which was developed one step previous to *M*8(−1). *T*3(+1) has been stored because it is on the negative feedback loop. None of the three local causes [*P*9(−1), *P*7(−1), or REVERSED VALVE ACTION] violate the consistency criterion. Therefore, employ the negative feedback operator and arrive at the following sub-tree.

```
                         M8(-1)
                            *
                           OR
                            *
             *****************************
             *      *              *      *
             *   P9(-1Ø)           *   P7(-1Ø)
            AND                  EOR (Gate A)
             *                     *
     **************         ****************
     *            *         *              *
  P9(-1)         OR       P7(-1)     REVERSED VALVE
                  *                      ACTION
            **************
            *            *
          P7(Ø)     (Same as Gate A)
```

The EOR is necessary in Gate A since a simultaneous occurrence of *P*7(−1) and REVERSED VALVE ACTION will cancel one another out resulting in *M*8(+1).

Now consider the event *P*9(−1Ø). There are two local causes: *P*1Ø(−1Ø) and PUMP SHUTDOWN. In this case, one of the causes (PUMP SHUTDOWN) is common to both paths of the feedforward loop. Therefore, employ the feedforward operator which results in

```
                          P9(-1Ø)
                             *
                            OR
                             *
               *****************************
               *                           *
              AND                       P1Ø(-1Ø)
               *
     ******************
     *                *
   PUMP              OR
 SHUTDOWN             *
            ******************
            *                *
         M2(+1Ø)           M2(Ø) .
```

In addition, store PUMP SHUTDOWN for later consistency checks.

As an example of a consistency violation, consider the event *P*6(−1) which would have resulted from the development of *P*7(−1). According to the digraph, the only cause of *P*6(−1) is *T*3(−1), but this is in violation of the consistency criterion, *T*3(+1). Hence, with the simple models used here, *P*6(−1) has no causes. Because of this, *P*6(−1) itself, must be deleted anywhere it appears in the tree.

The rest of the fault-tree can be developed using this algorithm. The final fault-tree is the same as was obtained using the manual technique except that events such as IMPROPER CONTROL LOOP ACTION are traced to more basic causes. A listing of this more complete fault-tree is given in the next section, which also contains the description of a computer program which synthesizes fault-trees.

6. FAULT-TREE SYNTHESIS PROGRAM

A computer program called FTS (Fault-Tree Synthesis) has been written which constructs fault-trees using the algorithm discussed in Sections 4 & 5. The operations performed by the program are exactly those described in Section 4. Before using the computer, however, an easy way must be developed for conveying all of the information in the digraph to the program.

What different pieces of information are contained in the digraph? The nodes of the digraph represent process variables and events. For example, nodes such as *T*4 denote variables and those like PUMP SHUTDOWN represent events. During the course of the synthesis, these nodes will take on values. The value associated with a process-variable node is the deviation in that variable while the value associated with an event-node denotes whether or not the event occurs. Nodes having inputs are termed connecting-nodes. Some nodes have no inputs. These correspond to either primal variables or events. The edges of the digraph show how the nodes (variables or events) are connected. Gains associated with the edges determine how one node affects others. In addition, certain types of edges are conditional upon other events (REVERSED VALVE ACTION is an example).

Therefore, classify each node according to two different criteria: 1) whether it corresponds to an event or variable, and 2) whether it is primal or connecting. Edges can be grouped into two classes, 1) conditional on some other event and 2) otherwise. In order to identify the types of nodes and edges on the digraph, numbers are assigned to them according to the following table.

TABLE 2
Digraph Coding Key

Name	Type	Number
Edge	Unconditional	(Not Numbered)
Edge	Conditional	$1 - M$
Node	Primal Event	$(M + 1)$–1ØØØ
Node	Connecting Variable or Connecting Event	1ØØ1 $- N$
Node	Primal Variable	$(N + 1)$ and up

Thus one first assigns all conditional edges values of 1 through *M*. Next any nodes corresponding to primal events are numbered beginning with $M + 1$. Then any nodes denoting connecting variables or events are assigned values of 1ØØ1 through *N*. Finally any remaining nodes corresponding to primal variables are numbered beginning with $N + 1$.

Input to FTS then takes the following form.

1) The values of *M* and ($N -$ 1ØØØ) are listed. These provide information on how many conditional edges and connecting nodes exist.

2) The inputs to each connecting node are listed along with their appropriate gains. If input is via an unconditional edge, the value of the node from which the edge originates is listed.

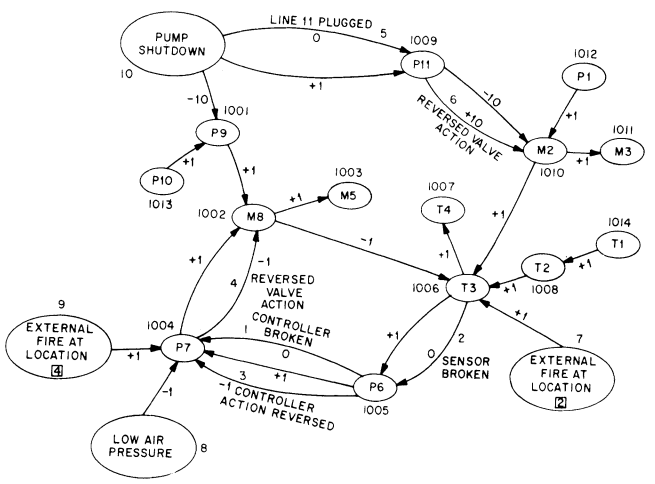


Fig. 4. Labeled digraph for nitric acid cooler process.

If it is by a conditional edge, the value of the edge itself is used. The gain associated with each edge is listed immediately below its value. This block of data allows FTS to determine the connectivity within the graph.

3) The nodes from which each conditional edge originates are listed. Together with the first block of data, this allows the program to determine special cases of 'conditional connectivity'.

4) The node corresponding to the top event is listed along with its value. This tells FTS where to begin on the digraph.

Fig. 4 shows the digraph for the nitric acid problem numbered for input to FTS. It required approximately one half hour to prepare the input for this problem. Run time for this problem was approximately two CPU seconds on an IBM 36Ø/67 computer. The output from FTS is shown in Fig. 5. Note the feedback structure in Gate 1 and the feedforward construction in Gate 13.

FTS has been used on larger problems with good results. The largest problem handled so far involved a 120 node digraph with 135 conditional edges. In that case, a 143 gate tree was synthesized in 25 CPU seconds. This tree contained 143 gates when no duplication of branches was considered. If the tree were expanded with duplication, a tree with well over 1000 gates would result. Factors affecting run times are digraph size as well as the number of negative feedback and feedforward loops in the process. Nesting of loops is also important. At present, appreciable time savings are accomplished by having the program recognize duplicate events and develop them only once. This is especially important when attacking larger problems.

FTS appears to be a very useful tool in system safety analysis since it allows the rapid synthesis of high quality fault-trees from process digraphs. As a process becomes more complex, however, development of the digraph can be tedious. A computer program is now being developed which will aid in generating the digraph. Associated with this program is the development of equipment models. These models are used by the program in assembling the digraph from a piping and instrumentation diagram. The current goal is a system which will aid the engineer in safety analysis much as general process

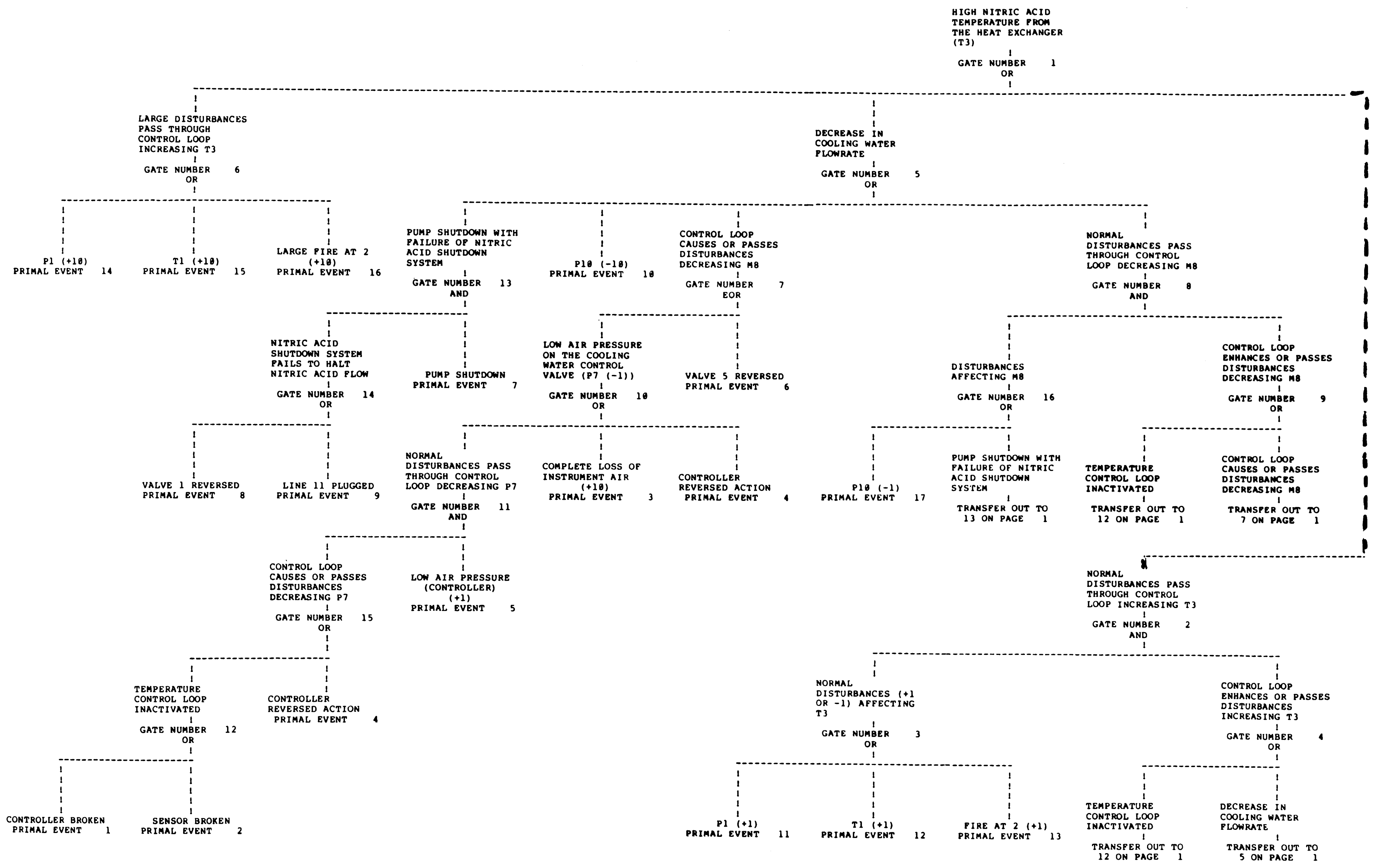

HIGH NITRIC ACID TEMPERATURE FROM THE HEAT EXCHANGER (T3) GATE NUMBER 1 OR
LARGE DISTURBANCES PASS THROUGH CONTROL LOOP INCREASING T3 GATE NUMBER 6 OR
DECREASE IN COOLING WATER FLOWRATE GATE NUMBER 5 OR
P1 (+10) PRIMAL EVENT 14
T1 (+10) PRIMAL EVENT 15
LARGE FIRE AT 2 (+10) PRIMAL EVENT 16
PUMP SHUTDOWN WITH FAILURE OF NITRIC ACID SHUTDOWN SYSTEM GATE NUMBER 13 AND
P10 (-10) PRIMAL EVENT 10
CONTROL LOOP CAUSES OR PASSES DISTURBANCES DECREASING M8 GATE NUMBER 7 EOR
NORMAL DISTURBANCES PASS THROUGH CONTROL LOOP DECREASING M8 GATE NUMBER 8 AND
NITRIC ACID SHUTDOWN SYSTEM FAILS TO HALT NITRIC ACID FLOW GATE NUMBER 14 OR
PUMP SHUTDOWN PRIMAL EVENT 7
LOW AIR PRESSURE ON THE COOLING WATER CONTROL VALVE (P7 (-1)) GATE NUMBER 10 OR
VALVE 5 REVERSED PRIMAL EVENT 6
DISTURBANCES AFFECTING M8 GATE NUMBER 16 OR
CONTROL LOOP ENHANCES OR PASSES DISTURBANCES DECREASING M8 GATE NUMBER 9 OR
VALVE 1 REVERSED PRIMAL EVENT 8
LINE 11 PLUGGED PRIMAL EVENT 9
NORMAL DISTURBANCES PASS THROUGH CONTROL LOOP DECREASING P7 GATE NUMBER 11 AND
COMPLETE LOSS OF INSTRUMENT AIR (+10) PRIMAL EVENT 3
CONTROLLER REVERSED ACTION PRIMAL EVENT 4
P10 (-1) PRIMAL EVENT 17
PUMP SHUTDOWN WITH FAILURE OF NITRIC ACID SHUTDOWN SYSTEM TRANSFER OUT TO 13 ON PAGE 1
TEMPERATURE CONTROL LOOP INACTIVATED TRANSFER OUT TO 12 ON PAGE 1
CONTROL LOOP CAUSES OR PASSES DISTURBANCES DECREASING M8 TRANSFER OUT TO 7 ON PAGE 1
CONTROL LOOP CAUSES OR PASSES DISTURBANCES DECREASING P7 GATE NUMBER 15 OR
LOW AIR PRESSURE (CONTROLLER) (+1) PRIMAL EVENT 5
NORMAL DISTURBANCES PASS THROUGH CONTROL LOOP INCREASING T3 GATE NUMBER 2 AND
TEMPERATURE CONTROL LOOP INACTIVATED GATE NUMBER 12 OR
CONTROLLER REVERSED ACTION PRIMAL EVENT 4
NORMAL DISTURBANCES (+1 OR -1) AFFECTING T3 GATE NUMBER 3 OR
CONTROL LOOP ENHANCES OR PASSES DISTURBANCES INCREASING T3 GATE NUMBER 4 OR
CONTROLLER BROKEN PRIMAL EVENT 1
SENSOR BROKEN PRIMAL EVENT 2
P1 (+1) PRIMAL EVENT 11
T1 (+1) PRIMAL EVENT 12
FIRE AT 2 (+1) PRIMAL EVENT 13
TEMPERATURE CONTROL LOOP INACTIVATED TRANSFER OUT TO 12 ON PAGE 1
DECREASE IN COOLING WATER FLOWRATE TRANSFER OUT TO 5 ON PAGE 1

Fig. 5. Fault-tree for the event $T4(+1)$ for the nitric acid cooler process.

simulators (PACER, FLOWTRAN, ECAP, etc.) now aid in performing mass and energy balances and electric circuit analyses. The digraph generator along with FTS would form the core of such a system.

ACKNOWLEDGMENTS

One of the authors (SAL) was supported by a Rockwell International Fellowship during this work. Mr. Douglas Johnston supplied the graphics program for 'drawing' the fault-tree on the line printer. The Carnegie-Mellon University Computation Center supplied computing support.

REFERENCES

[1] D.F. Haasl, "Advanced concepts in fault-tree analysis", System Safety Symposium, 1965 June 8-9, Seattle: The Boeing Company.

[2] W.E. Vesely, R.E. Narum, "PREP and KITT: Computer codes for the automatic evaluation of a fault-tree", IN-1349, August (1970).*

[3] J.B. Fussell, W.E. Vesely, "A new methodology for obtaining cut sets for fault-trees", *Transactions of the American Nuclear Society,* vol 15, 1972, pp 262-263.

[4] *Reactor Safety Study–An Assessment of Accident Risks in U.S. Commercial Nuclear Power Plants, WASH-14ØØ (NUREG-75/Ø14),* U.S. Nuclear Regulatory Commission, Washington, D.C., 1975 October.*

[5] G.J. Powers, "Advanced process synthesis" - Course Notes, Carnegie-Mellon University, 1974.

[6] J.B. Fussell, "Synthetic tree model–a formal methodology for fault-tree construction", Aerojet Nuclear Report ANCR Ø1Ø98, March (1973).*

[7] J.R. Taylor, "A formalisation of failure mode analysis of control systems", Danish Atomic Energy Commission, RISO-M-1654, September (1973). Available from Library of the Danish Atomic Energy Commission; (Atomenergikommissionens Bibliotek); Risø; DK-4000 Roshilde, DENMARK.

[8] G.J. Powers, F.C. Tompkins, "Fault-tree synthesis for chemical processes", *AICHE Journal,* vol 20, 1974 March, pp 376-387.

*Available from National Technical Information Service; Springfield, VA 22161 USA.

Comments on Some Referees' Questions

Steven A. Lapp
Gary J. Powers

1. Is the digraph unique? Is the fault-tree derived from a given digraph unique?

The digraph for a complete system is constructed from a flowsheet of the system and input/output models for the components in the system. If the models for the components are standardized (i.e. we agree as best we can on how valves, pumps, switches, controllers, human operators, etc. fail and/or propagate signals), then the digraph for the system will be unique. If we have different ideas about: 1) how the system is interconnected, or 2) what the models for the components within the system are then different digraphs will result.

Given a particular digraph, the fault-tree derived from it is unique. The algorithm is completely deterministic and given one digraph, it will develop only one tree. Of course, other digraphs could give rise to the same tree. Hence, the mapping from digraph to fault-tree is unique, but the reverse is not.

2. Will computer-aided fault-tree synthesis do away with analysis by hand? What about the very important interactions which go on between the fault-tree analyst and the system designers and operators? Won't they be lost?

I certainly would not do a complex fault-tree by hand now that I have this program. I would make too many mistakes; it would take too long; and I would get bored out of my mind. If I had a small problem, I would run the algorithm by hand on a digraph which I generated (by hand) for the system.

The important interactions with the designers and operators come during the flowsheet selection and development of the input/output models for the components within the system. It is in these models that we try to capture knowledge and experience. The flowsheet and modeling can be done at several levels of complexity. Simple flowsheets and models might be used initially to find the important parts of the system. Additional detail could be added to the flowsheets and models as more insight into the system behavior is required. The art resides in being able to pick the level of system flowsheet and modeling that is appropriate to the problem. Too little detail may overlook important failures. Too much detail may overwhelm the analyst. The fact that we have the algorithm allows us to add considerably more detail on operator actions, external events like fires, explosions, floods, vibrations, etc. without getting bogged down in the analysis. The discussions with the designers and operators are obviously still necessary to decide where to concentrate the effort.

3. Loopfinding. In the examples, the feedback and feedforward loops were found by inspection. For complex examples will the computer be able to find all the loops?

Yes. Fortunately, computers can find loops in networks of the size we are considering with very little trouble. To find 50 loops in a network of over 300 nodes requires less than 20 seconds of IBM 360/67 time.

4. Are loops always associated with control components, or can they occur due to natural process flows?

Most of the negative feedback and negative feedforward loops in a chemical process are due to the control structures. However, there are some natural loops of this type. For example, endothermic chemical reactions stabilize the temperature at which they occur by a feedback loop from temperature to reaction rate to heat of reaction and back to temperature. Similarly, the negative temperature coefficient of a nuclear reactor is a negative feedback loop that occurs naturally and would be found in a system digraph containing that component. I might also add that some loops have power while others do not. E.g., if a negative feedback loop (NFBL) involving a controller is converted into a positive feedback loop (PFBL), the loop will be unstable and drive itself to an extreme value. However, the PFBL between the inlet and outlet flows in a pipe does not have power and will not drive itself to an extreme.

5. Will this algorithm ignore failures which are sequence dependent?

The algorithm will handle sequential behavior provided that this behavior appears in the component models for the system. We capture this behavior by making certain relationships (digraph edges) time dependent.

6. Are digraphs the same as signal flow graphs?

The digraphs discussed are not signal flow graphs. The difference lies in the fact that digraphs do not perform algebra. Signal summation at a node, for instance, is not done. Only the discretized cause and effect relationships using the ∅, ± 1, ± 1∅ variable values are defined. This is why fault-trees constructed for digraphs do not indicate the exact amount of variable deviation necessary to produce a given effect.

7. What about combined inputs? (e.g. brittle fracture due to both high pressure and low temperature) Will digraphs handle these cases?

Digraphs can handle this type of event in two ways. First, the multivalued logic could be used. The digraph for large changes in pressure and temperature might be as shown.

$$\begin{tikzcd} \textcircled{P} \arrow[r, "+1", "P=(+1\emptyset)"'] & \boxed{\text{Fracture}} & \textcircled{T} \arrow[l, "-1"', "T=(-1\emptyset)"] \end{tikzcd}$$

If the pressure is very high, that is all it takes. No need to worry about *T* being different from its normal condition. And, if *T* is very low, there is no need to worry about *P*.

If normal deviations in *P* and *T* can combine to cause fracture, the model is this one.

$$\begin{tikzcd} \textcircled{P} \arrow[r, "+1", "T=(-1)"'] & \boxed{\text{Fracture}} & \textcircled{T} \arrow[l, "-1"', "P=(+1)"] \end{tikzcd}$$

8. What is the rule for deciding whether an edge is conditional on an event or other variable?

If the relationship between two variables is dependent on one (or more) other variables or events, make the edge conditional on these variables or events. This is a powerful feature of digraphs and allows the inclusion of wide classes of behavior in one model.

9. Why are consistency checks required in the algorithm only for feedback and feedforward loops?

Because these are the only situations in which variables can come back on themselves (i.e. cause themselves in an inconsistent manner).

10. Will the Fault-Tree Synthesis (FTS) program handle electrical systems?

Yes. The analogy between electrical circuits and fluid flow systems is strong and digraphs can be easily constructed for circuits containing resistors, switches, relays, batteries, capacitors, etc.

Steven A. Lapp; Department of Chemical Engineering; Carnegie-Mellon University; Pittsburgh, PA 15213 USA

Steven A. Lapp is a Rockwell Graduate Fellow in Chemical Engineering at Carnegie-Mellon University. He is studying the systematic synthesis of fault-trees for chemical processing systems. He has worked for the Exxon Company and is co-lecturer in a Fault-Tree Analysis short course given at Carnegie-Mellon University.

Gary J. Powers; Department of Chemical Engineering; Carnegie-Mellon University; Pittsburgh, PA 15213 USA

Gary J. Powers is Professor of Chemical Engineering and Director of the Design Research Center at Carnegie-Mellon University. Following a BS ChE degree from the University of Michigan, he received his PhD in Chemical Engineering from the University of Wisconsin. He has worked for the Dow Chemical Company, taught at the Massachusetts Institute of Technology, coauthored the text *Process Synthesis*, and consulted for numerous companies and governmental agencies.

Manuscript received 1976 February 10; revised 1976 September 17.

¤ ¤ ¤

Manuscripts Received

For information, write to the author at the address listed; do NOT write to the Editor.

"A method for estimating the reliability of IC components", D.E. Brodie; Dept. of Physics; University of Waterloo; Waterloo, Ontario N2L 3G1 CANADA.

"A 4-unit redundant system subject to common-cause failures", Balbir S. Dhillon; CAE Electronics Ltd; PO Box 1800; Saint-Laurent, Montreal, Quebec H4L 4X4 CANADA.

"Reliability of a system with two dissimilar modules", M. Sambandham; Dept. of Mathematics; A.C. College of Engineering & Technology; Karaikudi - 623 004 INDIA.

"Predicting the confidence of passing life-tests", R.E. Schafer; Bldg 606, MS K-217; Hughes Aircraft Company; PO Box 3310; Fullerton, California 92634 USA.