Automatic Translation of Digraph to Fault-Tree Models

David L. Iverson; NASA-Ames Research Center, Moffett Field

Key Words: Digraph, Fault-tree, Translation, Cut set, Diagnosis

SUMMARY & CONCLUSIONS

This paper presents a technique for converting digraph models, including those models containing cycles, to a fault-tree format. A computer program has been developed which automatically performs this translation using an object-oriented representation of the models. The fault-trees resulting from translations can be used for fault-tree analysis and diagnosis. Programs to calculate fault-tree and digraph cut sets and perform diagnosis with fault-tree models have also been developed.

The digraph to fault-tree translation system has been successfully tested on several digraphs of varying size and complexity. Details of some representative translation problems are presented later in this paper. Most of the computation performed by the program is dedicated to finding minimal cut sets for digraph nodes in order to break cycles in the digraph. Fault-trees produced by the translator have been successfully used with NASA's Fault-Tree Diagnosis System (FTDS) to produce automated diagnostic systems.

1. INTRODUCTION

Fault-tree and digraph modeling are frequently used for failure analysis of large systems. Both types of models represent a failure space view of the system using AND and OR nodes in a directed graph structure. Each type of model has advantages. Digraph models can be derived from system schematics in a straightforward manner by associating a digraph node with each component in the schematic and augmenting the basic digraph with knowledge about component failure modes. (Ref. 1) In addition, digraphs allow any pattern of interconnection between graph nodes.

Fault-trees must have a strict tree structure and do not allow cycles (loops) in the graph. They are built using a top down hierarchical analysis of the system. Each subsystem failure is broken down into a set of lower level causes until basic component level failures are reached. The restrictions placed on fault-tree structure allow straightforward processing of fault-tree models using efficient techniques developed for tree data structures.

The similarity between digraphs and fault-trees suggests that the information in a digraph model could be represented in a fault-tree form. A method for translating digraph models, including those with cycles, to a fault-tree form has been developed. The nodes of a fault-tree produced by this translation scheme will have the same minimal cut set solutions as corresponding nodes in the input digraph.

This translation tool is useful for both digraph modelers and fault-tree modelers, as well as those who use fault-tree-like models as knowledge representations. There are several powerful fault-tree processing codes, such as cut set and quantitative solution codes, which could be used to help in the analysis of the translated digraph. If some parts of a system have been modeled using digraphs and some with fault-trees (e.g. a large project split over several contractors who used different modeling techniques), then the digraphs could be translated and incorporated into the fault-tree models. Also, in applications using graph models as knowledge or data, an acyclic fault-tree-like model will often be easier to use than an equivalent cyclic digraph-like model since the need to check for cycles is eliminated.

This paper describes a technique for translating digraph models into fault-tree models and details a system which uses this technique to automatically perform the translation. It also describes algorithms to calculate minimal cut sets for digraphs and fault-trees and briefly describes the use of fault-trees and translated digraph models in an automated diagnosis system.

2. DIGRAPH & FAULT-TREE MODELS

Digraph and fault-tree models were both developed for system reliability analysis. They are failure space models of a system where each node in the model represents a failure in the modeled system. A digraph model is a directed graph with nodes representing hardware failures, human actions, modes of operation, and other factors affecting system operation, connected by directed edges. The digraph edges represent flow connectivity, such as physical connections or the results of operator actions, where a
failure of a node could propagate through the directed edges and cause other nodes in the path to fail. Digraphs also contain specialized nodes representing AND gates which indicate that a failure must occur in each input of the gate in order to propagate through the output edges (Ref. 1). AND gates are drawn as bars in the digraph and regular nodes are drawn as circles (see fig. 1). Digraph nodes can be in one of two states, true or false. If a node is true, or marked, it means the item that the node represents has failed. If the node is false, or unmarked, then the item has not failed.

Fig. 1 shows a digraph model of a cooling system. Each node is a component of the system and the edges correspond to the flow of the coolant. Notice that a failure in the coolant reservoir or the coolant pipes could propagate through to cause the pumps to fail in their function of delivering coolant to the cooling unit. The bar in the digraph is an AND gate which indicates that both the primary and backup pumps must fail to operate before the cooling unit fails due to lack of coolant. Depending on the desired level of detail, additional nodes (e.g. nodes representing valves on the coolant pipes) could be added to this digraph.

A fault-tree model has a tree structure with nodes representing system failures and gates indicating how those failures interact. The inputs (children) of a gate represent failures that would cause or contribute to the failure of the gate's output (parent) node. In a traditional fault-tree the gates are either AND gates or OR gates. If a node is the output of an AND gate, then all the failure events which are children of that node (inputs to the AND gate) must occur before the parent event at the output of the gate occurs. If a node is the output of an OR gate, then only one of the child failure events must occur before the parent event will occur. The leaf nodes of a fault-tree are called Basic Event nodes. They usually represent individual component failures or replaceable unit failures.

Fig. 2 shows a fault-tree model of the same cooling system modeled in fig. 1. The fault-tree gates are shown as logic gates. OR gates have concave bottoms and AND gates have straight bottoms. Since each gate has only one output event (the event above the gate in the fault-tree) it is convenient to consider the gate and the output event as a single node in the fault-tree. This allows us to represent each fault-tree event
as an AND gate node, OR gate node, or Basic Event node in a directed graph structure.

The fault-tree in fig. 2 shows that a failure of the cooling system could be caused by a cooling unit failure or a failure of the coolant delivery system. The root node of this fault-tree is an OR gate representing failure of the cooling system and its child events are failure of the cooling unit and failure of the coolant delivery system. Coolant delivery system failure could be caused by a failure in the coolant supply or failure of the pumps, and so on down the tree. Notice that the 'pumps failed' node is an AND gate indicating that the primary and backup pumps must both fail before the 'pumps failed' event occurs.

3. MINIMAL CUT SETS

Digraph and fault-tree models are often used to calculate minimal cut sets (also called failure sets). A cut set for a given node is a group of failure events that will cause the failure represented by that node. For instance, the cooling system modeled in figs. 1 and 2 would fail if the coolant pipes failed or if both coolant pumps failed. One cut set for the cooling system failure node would contain the failure event 'coolant pipes failed'. Another cut set would contain two failure events, 'primary pump failed' and 'backup pump failed'. The minimal cut sets for a node are all the node's cut sets except for those which are proper supersets of any other cut set in the group. Cut set calculations can be used to find dependencies in the modeled system and to discover weak links and vital system components whose failure could cause a serious system failure.

In a digraph model, the cut sets for a node are the sets of nodes which must be marked (i.e. in a failed state) to cause the given node to be marked. For instance, in fig. 1 the minimal cut sets for the cooling unit node would be ['cooling unit', 'primary pump', 'backup pump', 'coolant pipes'], and ['coolant reservoir']. If all the nodes in any one of these cut sets were marked, then the cooling unit would fail. One way to think of the cut sets of a digraph node is to divide them into two parts: a singleton set indicating failure of that node, and other cut sets representing failures in the input paths of that node. This idea is used in the translation between digraphs and fault-trees.

Fault-tree cut sets are sets of basic failure events, or fault-tree leaf nodes, that could cause the failure represented by a fault-tree node to occur. Fault-tree cut sets can be calculated recursively by finding the cut sets of an event's children, then combining those cut sets into sets containing the basic failure events that could cause that event to occur. (Basic Event nodes have a single cut set containing the failure event represented by that node.) For instance, in fig. 2 the event 'pumps failed' could be caused by the occurrence of two other events, 'primary pump failed' and 'backup pump failed', so the cut sets of its child events would be combined to come up with a single cut set ['primary pump failed', 'backup pump failed']. Then its parent event 'coolant delivery system failure' would combine that cut set with the cut sets from its other child to come up with the cut sets ['reservoir empty'], ['coolant pipes failed'], and ['primary pump failed', 'backup pump failed']. Notice that the cut sets for an OR gate will include all the cut sets of each of its children, while an AND gate will combine its children's cut sets into new sets containing basic failure events that will cause every one of its child failure events to occur. As we propagate the cut sets up the tree, we find that the top fault-tree node, 'cooling system failure', in fig. 2 has the same cut sets as the 'cooling unit' digraph node in fig. 1.

4. DIGRAPH TO FAULT-TREE TRANSLATION

Translating digraph models into fault-tree models would be a straightforward task if cycles (loops) were not allowed in digraph models. The translation scheme presented addresses this problem and resolves those cycles by using repeated events in the fault-tree model. A flow chart of the translation algorithm is shown in fig. 3.

The translation begins with the user specifying a terminal node in the digraph. The resulting fault-tree will use this terminal node as a root event and will contain all the information in the digraph that is upstream (nodes that feed into the terminal node) of this terminal node. A fault-tree OR gate node representing the terminal node event is created. This OR gate is called an event node. It is the fault-tree node that corresponds to the digraph node being processed. A Basic Event node is then added as a child of that root event node. This Basic Event node is called a failure node. It represents the failure of the associated digraph component. Then, before any of the terminal node's inputs are processed, the terminal node is marked as a visited node.

Next, each input path into the terminal node is processed. If the input node is a regular digraph node (not an AND gate), a fault-tree OR gate event node representing that digraph node is created and made a child of the root node. The input digraph node is then processed in the same way that the terminal node was processed, using the new fault-tree event node as a subtree root node. A Basic Event failure node is added as a child, the digraph node is marked as visited, and its inputs are processed.

If the input to the digraph terminal node is an AND gate, the fault-tree node corresponding to that input is also an AND gate. No failure nodes are created for AND gates. The digraph AND gate's
Add a F.T. AND gate child; mark the digraph AND as visited; process the digraph AND's inputs

Add an event node child; add a failure node as its child; mark the digraph node as visited; process the digraph node's inputs

Add the subtree representing that node as a repeated subtree

Add a repeat node child to be resolved later

START
Is the digraph node marked as visited?

no

Is the digraph node an AND gate?

yes

Is an equiv. node an ancestor of the current F.T. node?

no

yes

Fig. 3: Flowchart of translation algorithm

input nodes are processed just like inputs into regular digraph nodes, using fault-tree event nodes or AND gates added as children of the fault-tree AND gate as subtree roots.

If we encounter an input node to a digraph node which is marked as already processed, we have found a cycle in the digraph. There are two possibilities in this situation. In the first case, there are no ancestors (nodes in the path above the current node) of the corresponding fault-tree node which are contained in the subtree representing the twice visited digraph node and its inputs. In this case we can safely add that subtree as a child of the corresponding fault-tree node without creating a cycle and destroying the tree structure. In the second case, one of the ancestors of the corresponding fault-tree node is contained in the subtree representing the twice visited digraph node and its inputs. In this case we cannot add this subtree as a child of the corresponding fault-tree node without creating a cycle in the tree. Instead, a new fault-tree node, called a repeat node, is created and added as a child. These repeat nodes are associated with twice visited digraph nodes and will be replaced by regular fault-tree nodes later in the conversion.

This first phase of processing will terminate when all the digraph input nodes either have no inputs themselves or have only previously processed nodes as inputs. After the first phase is finished, the only remaining task is to resolve the digraph cycles by replacing the fault-tree repeat nodes with subtrees equivalent to their associated digraph nodes.

5. RESOLVING DIGRAPH CYCLES

Each repeat node in the fault-tree corresponds to a node in the input digraph and must have the same minimal cut sets as that digraph node. This is accomplished by building a subtree from the minimal cut sets of the digraph node. The repeat node becomes an OR gate with a child for each minimal cut set of the digraph node. If a cut set has a single element, a single failure node representing that element is added as a child of the repeat node. If a cut set has multiple elements, an AND gate is created with a failure node for each of the cut set elements as children. This AND gate then becomes a child of the repeat node.

Minimal cut sets for a digraph node can be calculated nearly the same way as described above for fault-trees. The cut sets for a digraph node are a combination of the cut sets of its input nodes plus a singleton cut set containing the node itself. A node with no inputs will have one cut set containing itself, just like a fault-tree Basic Event node. An AND gate's cut sets will be a combination of its inputs' cut sets such that the failure of the events in each cut set will cause the failure of all the AND gate's input nodes. These AND gate cut sets would be the union of one cut set from the first input, one from the second input, and so forth. An AND gate cut set will be produced by each possible combination of one cut set from each of the gate's inputs. The minimal cut sets for a node are formed by removing all those cut sets which are supersets of other sets in the group of cut sets.

The digraph cut set algorithm calculates minimal cut sets for a node recursively. It will calculate the cut sets for a node's inputs, then calculate the node's cut
sets. During the cut set calculations, each digraph node that has been visited during a series of recursive calls is marked. If one of these nodes is encountered again (i.e. it is contained in a cycle), the recursion stops. Since this node has already been visited, all of its input nodes have been included in the cut sets calculated thus far. Any cut sets added by processing this node again would later be eliminated since they would not be minimal cut sets.

This cut set calculation algorithm will eventually stop since each path into a node will end either in a node with no inputs or in a cycle. Any supersets of other sets in a node's group of cut sets are eliminated, leaving us with minimal cut sets for that node.

To complete the conversion, the minimal cut sets for each digraph node corresponding to a fault-tree repeat node are converted into sets of failure nodes, a subtree is built with these failure nodes as described above, and the fault-tree repeat node is replaced with this subtree. After all the repeat nodes in the fault-tree have been processed in this manner the conversion is complete. We now have a fault-tree where each node in the tree has the same minimal cut sets as its corresponding node in the input digraph.

6. TRANSLATION EXAMPLE

As an example we will go through the translation of a digraph model of a portion of an aircraft flight control system. The example digraph is shown in fig. 4. The flight control system has redundant flight control computers. This is modeled by the nodes FCCA and FCCB feeding into an AND gate, indicating that both computers (or their inputs) must fail for the control system to fail. Notice that the digraph contains cycles representing feedback in the control system.

The nodes in the resulting fault-tree will be named by appending suffixes to the name of the digraph node to which they are related. Fault-tree failure nodes will end in '-F', repeat nodes will end in '-R', and event nodes will retain the name of the corresponding digraph node. AND gates created when building subtrees to resolve digraph cycles will use the name of the node being resolved with '-A' and an integer appended.

We will choose digraph node TRMM as our terminal node. The first step in the translation creates the fault-tree OR gate root node TRMM and a failure node TRMM-F representing the failure of the digraph node TRMM. TRMM-F is added as a child of TRMM. Since the LOFS node is not marked as visited, an event node LOFS is also added as a child of TRMM. The resulting partial fault-tree is shown in fig. 5.

Before proceeding, the digraph node TRMM is marked as visited.

In the next step, the digraph node LOFS is processed. The failure node LOFS-F and an AND gate AND1, representing the digraph AND gate, are added as children of LOFS (see fig. 6). LOFS is marked as visited and the input AND gate is processed. No failure node is created for the AND gate since digraph AND gates do not correspond to actual failures in the system. The event nodes FCCA and FCCB, representing the inputs to the AND gate, are added as children of AND1. The AND gate is marked as visited and processing continues with FCCA and its inputs, then FCCB and its inputs.

Assume we have just processed the FCCA input ROLS. We then move on to process its input CSGS and the CSGS input TRMM. We notice that TRMM has already been visited and a cycle has been found. We check to see if a fault-tree node corresponding to TRMM appears as an ancestor of the event node CSGS.
and discover that the root node TRMM is an ancestor. In this case, a repeat node TRMM-R is added as a child of the fault-tree node CSGS. This repeat node will be replaced with a subtree later when the digraph cycle is resolved.

Now assume we have finished processing the
inputs for FCCA and have moved on to the inputs of FCCB. When we try to process the digraph node YAWS, we notice that it has already been visited during the processing of FCCA’s inputs. We check to see if the event node YAWS is an ancestor of the event node FCCB. We see that it is not. This means we can safely place the already created subtree representing YAWS and its inputs as a repeated subtree under the node FCCB without creating a cycle in the fault-tree. Note that we only need to check for YAWS in the ancestors of FCCB, not for YAWS and all its inputs. Any input of YAWS appearing in the fault-tree as an ancestor of FCCB would have YAWS as its ancestor, and therefore YAWS would also be an ancestor of FCCB.

Processing continues in this way until all the nodes upstream of TRMM have been visited. When this is done, the first phase of the translation is complete.

The only task remaining in the translation is to resolve the digraph cycles that were discovered in the first phase. In this example the cycles containing the node TRMM must be resolved. We run the digraph minimal cut set algorithm described above to find the minimal cut sets for node TRMM. These cut sets are: TRMM, (LOFS), (CHNS), (ROLS), (CSGS), (LAFS), (RFSC), (RUDP), (RDFT), (YAWS), (PITS), and the doubleton (FCCA FCCB). Each singleton cut set is converted into a failure node and added as a child to the OR gate repeat node TRMM-R. Then a fault-tree AND gate, TRMM-A1, is created with failure nodes FCCA-F and FCCB-F as children. This subtree corresponds to the doubleton cut set of the digraph node TRMM. The subtree is added as a child of the repeat node TRMM-R. We can see that TRMM-R will have the same minimal cut sets as the digraph node TRMM. Any nodes above TRMM-R in the fault-tree, which correspond to nodes downstream of TRMM in the digraph, will incorporate the cut sets of TRMM in their minimal cut sets. Thus, any combination of failures that could cause the digraph node TRMM to fail and propagate its failure will also cause the TRMM event node failure in the fault-tree and the occurrence of any other failure events that the TRMM failure could cause.

The complete fault-tree produced by the conversion of the digraph in fig. 4 is shown in fig. 6. The triangles containing numbers in fig. 6 indicate that the subtree shown with that numbered triangle at its root should be inserted where the numbered triangle appears in the tree.

7. DESCRIPTION OF THE CONVERSION CODE

The conversion algorithm described above has been implemented in Common LISP using the Flavors object-oriented programming package and in the C programming language. This code, as well as code for the FTDS diagnosis system, is available through NASA’s COSMIC software distribution service.

The programs represent nodes in the digraphs and fault-trees with objects like those shown in fig. 7. The AND gate representation is similar to the OR gate representation. Each object has a name and a type (AND gate, OR gate, Basic Event) associated with it. Node interconnections are stored in the objects as pointers to objects representing the connected nodes.

The conversion algorithm takes a digraph model and a terminal node name as input. The object representing the terminal node becomes the fault-tree root node. A Basic Event object, representing the failure of the terminal node, and each unprocessed input node are added as children of the root node. When this is done, the ‘processed’ slot in the terminal node object is set to true. Next, the program follows the input pointers in the terminal node and processes the input nodes. If the ‘processed’ slot of a node’s input is set to true, the program checks to see if any ancestors of the node’s corresponding fault-tree node are equivalent to the input node by following the pointers listed in the parent slots of the fault-tree node and its ancestors. If no equivalent nodes are found, a repeated subtree is added as a child of the

---

| Name: | PP-F |
| Type: | Basic-Event |
| Event: | Primary Pump |
| Parents: | PFA1 |
| Equiv-to: | '(PP-F) |

Fault-Tree Basic Event Object

| Name: | PP |
| Type: | OR-Gate |
| Event: | Primary Pump |
| Inputs: | CP |
| Outputs: | AND1 |
| Parent: | NIL |
| Children: | NIL |
| Processed: | NIL |
| Cutsets: | NIL |

Fault-tree OR Gate/Digraph Node Object

---

360 1992 PROCEEDINGS Annual RELIABILITY AND MAINTAINABILITY Symposium
Table 1: Results of digraph conversions

| Digraph Name | # OR gates in Digraph | # AND gates in Digraph | # Cycles Broken | # Nodes in Fault-tree | Run Time on SPARC 1+ |
|--------------|-----------------------|------------------------|-----------------|----------------------|----------------------|
| FCS Ex.      | 13                    | 1                      | 1               | 29                   | 0.3 sec              |
| FIRM Ex.     | 29                    | 7                      | 1               | 69                   | 0.3 sec              |
| F18 FCS      | 182                   | 60                     | 1               | 654                  | 7.846.5 sec          |
| Shuttle M.E. | 41                    | 18                     | 8               | 2564                 | 396.5 sec            |

8. RESULTS

The digraph to fault-tree conversion system has been successfully tested on several digraphs of varying size and complexity. Some results are presented in table 1 showing the size of the input digraph, the number of digraph cycles broken for the translation, the number of unique nodes in the resulting fault-tree, and the CPU time required for the translation on a Sun SPARCStation 1+. Most of the computation performed by the program is dedicated to finding complete cut sets for digraph nodes to break cycles in the digraph.

The digraph translations appearing in this table were run on a Sun SPARCStation 1+ with 28MB of memory running Allegro Common LISP with Flavors. Digraph cut set calculations were done with a call to the C language cut set code. The run times were determined using the LISP and Unix time functions. The FCS example is the digraph shown in fig. 4. The FIRM example digraph appears in ref 3. The F18 Flight Control System digraph was produced at NASA Ames Research Center by Joe Conley. The partial Space Shuttle Main Engine digraph was provided by the SAIC Corporation.

9. USING CONVERTED DIGRAPHS FOR DIAGNOSIS

One possible interpretation of a fault-tree model is to consider it as a group of if-then rules. Each gate in the fault-tree corresponds to a causal rule containing the output failure event of the gate in the consequent and the input failure events in the antecedent. For instance, an AND gate with output event D and input events A, B, and C would correspond to the rule 'If A and B and C then D'. A diagnosis system has been developed (called FTDS for Fault-tree Diagnosis System) which exploits this node/rule duality. (Ref. 2) This system uses an object-oriented representation of an augmented fault-tree model as a knowledge base to diagnose basic event failures based on information provided about higher level failure events. The fault-trees used by FTDS are augmented by providing additional heuristic and temporal information about causality and fault propagation. The fault-tree models produced by the digraph to fault-tree translation system are adaptable for use by FTDS. Using these fault-trees as a knowledge base, FTDS can diagnose failures in the systems represented by the original digraph models. Currently, the full power of FTDS cannot be immediately employed with the translated digraphs since the digraph models do not contain proper heuristic and temporal information. To remedy this, digraph models can be enriched to include this information, or the resulting fault-trees can be further developed to provide a richer knowledge base which would allow FTDS to use its temporal and heuristic reasoning features more effectively. Several digraphs have been translated with this system and used successfully with FTDS in our laboratory.

REFERENCES

1. Sacks, Ivan J., "Digraph Matrix Analysis", IEEE Transactions on Reliability, pp. 437-445, Vol. R-34, No. 5, December 1985.

2. Iverson, D.L. and F.A. Patterson-Hine, "Object-Oriented Fault-tree Models Applied to System Diagnosis", Proc. of the SPIE Applications of Artificial Intelligence VIII conference, Orlando, FL, Vol. 1293, April 1990.

3. Becker, F.J., D.T. Bui, D.B. Hackler, "Failure Identification and Risk Management Algorithm Description and User's Guide", NASA Technical Report No. JSC-24448, June 1990.
BIOGRAPHY

David L. Iverson
NASA Ames Research Center
Mail Stop 244-4
Moffett Field, California 94035-1000 USA

David L. Iverson received a B.S. degree in Mathematics and Computer Science with a minor in Physics from the University of Puget Sound in 1988. His undergraduate thesis research explored artificial intelligence and machine learning. In 1991 he received his M.S. in Computer Science with an emphasis in symbolic and heuristic computation from Stanford University. He now holds a full time research position in the Information Sciences Division at NASA Ames Research Center. His interests include the application of artificial intelligence to fault diagnosis, system modeling and representation, and using system reliability models for automated diagnosis.