Power-Hierarchy of Dependability-Model Types

Manish Malhotra
AT&T Bell Laboratories, Holmdel
Kishor S. Trivedi, Fellow IEEE
Duke University, Durham

Key Words — **Combinatorial-model type, dependability, fault-tree, generalized stochastic Petri net, Markov-model type, reliability block diagram, reliability graph, stochastic reward net.**

Reader Aids —
General purpose: Widen the state-of-the-art
Special math needed for explanations: Discrete mathematics
Special math needed to use results: Same
Results useful to: Reliability modelers

Summary & Conclusions — **This paper formally establishes a hierarchy, among the most commonly used types of dependability models, according to their modeling power. Among the combinatorial (non-state-space) model types, we show that fault trees with repeated events are the most powerful in terms of kinds of dependencies among various system components that can be modeled (which is one metric of modeling power). Reliability graphs are less powerful than fault trees with repeated events but more powerful than reliability block diagrams and fault trees without repeated events. By virtue of the constructive nature of our proofs, we provide algorithms for converting from one model type to another. Among the Markov (state-space) model types, we consider continuous-time Markov chains, generalized stochastic Petri nets, Markov reward models, and stochastic reward nets. These are more powerful than combinatorial-model types in that they can capture dependencies such as a shared repair facility between system components. However, they are analytically tractable only under certain distributional assumptions such as exponential failure- & repair-time distributions. They are also subject to an exponentially large state space. The equivalence among various Markov-model types is well known and thus only briefly discussed.**

1. INTRODUCTION[1]

Fault-tolerant computer systems are used in a variety of applications that require high reliability or availability. For instance, computer systems in flight-control in aircraft & spacecraft require that the system provide service, without failing, until the end of mission. Such systems have a high reliability requirement. On the other hand, computer systems in database applications and communication networks are required to be operational for as high a fraction of time as possible (there is no critical mission time in this case). Such systems are required to possess high availability. Laprie [1] coined the term dependability as a measure of the quality, correctness, and continuity of service delivered by a system. Dependability encompasses measures such as reliability, availability, and safety.

Over the years, several model types such as reliability block diagrams, fault trees, and Markov chains, have been used to model fault-tolerant systems and to evaluate various dependability measures. These model types differ from one another not only in the ease of use in a particular application but in terms of modeling power. For instance, a *series-parallel*[2] system is reasonably modeled by a *series-parallel* reliability block diagram. Similarly, fault trees are more intuitive in capturing how a component failure propagates into a higher level subsystem or system failure. Thus some model types lend themselves easily to model certain kind of behavior of systems. Modeling power of a model type is determined by the kinds of dependencies within subsystems that can be modeled and the kind of dependability measures that can be computed. For instance, if various components of a system share a repair person (repair dependency among components), then FT or RBD cannot easily be used to model the availability of this system. Markov chains and stochastic Petri nets can easily model such a repair dependency.

From a variety of model types, a particular model type is chosen to specify a model. The choice of a suitable model type is determined by factors such as:

Constraints

- Familiarity of the user with the model type
- The model type supported by the available modeling tool-kit

Choices

- Ease of use in a particular application
- The kind of system and system behavior to be modeled
- The measure of system behavior to be computed
- Conciseness and ease of model specification

This paper analyzes the *choices* category and ignores the *constraints* category (although it is obviously important in some situations). The modeler's decision process can be greatly simplified by comparing model types according to:

- modeling power
- conciseness of model specification.

Little has been done to compare formally the dependability-model types. Ref [2,3] summarize the dependability-model types. These studies informally discuss model types, the kinds of dependencies that can be modeled by them, and dependability measures that can be evaluated using these model types. However,

[1]Acronyms, nomenclature, and notation are given at the end of the Introduction.

[2]The terms, *series* / *parallel* are used in their logic-diagram sense, irrespective of the schematic-diagram or physical-layout.

to the best of our knowledge, there has been no formal comparative evaluation of model types except for the following studies.

- Using probabilistic arguments, Shooman [4] showed the equivalence of RBD & FT (without repeated events); *ie*, any system that can be modeled by RBD can also be modeled by FT and vice-versa.
- Hura & Atwood [5] showed how Petri net models can represent *s*-coherent fault trees. They showed that an equivalent Petri net representation allows study of dynamic behavior of the model and offers more insightful treatment of fault detection & propagation. However, they do not show that Petri nets can model certain systems which can not be modeled by FT. ◄

This paper is mainly concerned with the modeling power of the following dependability-model types:

- reliability block diagrams
- fault trees without repeated events
- fault trees with repeated events
- reliability graphs
- continuous-time Markov chains
- generalized stochastic Petri nets
- Markov reward models
- stochastic reward nets.

We compare the model types and establish a hierarchy of model types on the basis of their modeling power. For example, to compare model types A & B, we —

- either provide an algorithm that converts any instance of model type A to an equivalent instance of model type B (and vice-versa),
- or prove that not every instance of model type A can be converted to an equivalent instance of model type B.

Some of the relationships our study reveal are obvious and some are not so obvious. Our aim is to provide a modeler with a power-hierarchy of dependability-model types which enable the modeler to select from a variety of model types for a given problem.

Section 2 describes the fault-tolerant multiprocessor system that is the illustrative example in this paper. Section 3 describes combinatorial-model types. Section 4 establishes power-hierarchy among combinatorial-model types. Section 5 briefly discusses Markov-model types and compares them to combinatorial-model types. Section 6 shows the overall power hierarchy.

Acronyms[3]

CTMC	continuous time Markov chain
FT	fault tree (without repeated events)
FTRE	fault tree with repeated events
GSPN	generalized stochastic Petri net
MRM	Markov reward model
RBD	reliability block diagram
RG	reliability graph
SRN	stochastic reward net.

Notation

M_i, P_i	[memory, processor] module i
N	interconnection network
U, V	set of [nodes, edges] in a digraph
G	reliability graph
e_i, w_i	[edge, node] i in a reliability graph
G_i	gate i in a fault tree
$D_{i,j}$	disk j of processing subsystem i
e	number of edges in a reliability graph
$p_{x,y}$, $p_{i:x,y}$	[path, subpath i] from node x to node y in a reliability graph
s_i	state i of a CTMC
ω_i	initial probability of being in s_i
p_i	place i in a Petri net
$t_{i,j}$	transition from p_i ($i \neq 0$) to p_j in a Petri net
$r_{i,j}$	rate associated with $t_{i,j}$
$t_{0,i}$	immediate transition between place p_0 and place p_i
C_i	component i
λ_i, μ_i	[failure, repair] rate of C_i
r_i	reward rate associated with state s_i of CTMC.

Other, standard notation is given in "Information for Readers & Authors" at the rear of each issue.

2. FAULT-TOLERANT MULTIPROCESSOR SYSTEM AN EXAMPLE

A fault-tolerant multiprocessor system is a running example in this paper. Figure 1 shows the basic multiprocessor architecture; it consists of two processors P_1 & P_2, each with a private memory M_1 & M_2 respectively. A processor and its memory form a processing unit. Each processing unit is connected to a mirrored-disk system. This forms a processing subsystem. Both processing units are connected via an interconnection network N. The system is functional while N is functional and at least one of the processing subsystems is functional. For a processing subsystem to be functional, the processor, memory module, and at least one of the two disks must be functional. For simplicity & illustration, we restrict ourselves to this 2-processor system. This architecture and the corresponding models are easily scaled to many processors.

3. COMBINATORIAL MODEL TYPES

3.1 Reliability Block Diagrams

RBD fall into the category of *combinatorial* (also known as *non-state-space*) model types [2, 3]. They map the operational dependency of a system on its components and not the actual physical structure. In Shooman's [4] words, RBD represent the probability-of-success approach to system modeling.

[3]The singular / plural of an acronym are always spelled the same.

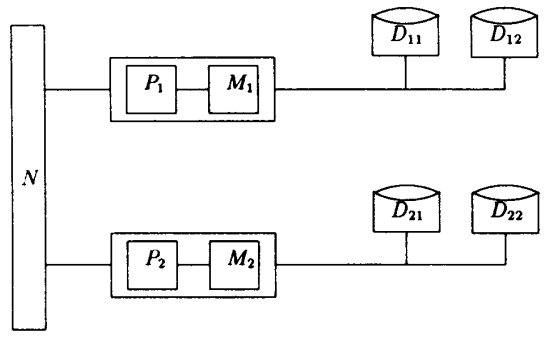


Figure 1. Fault-Tolerant Multiprocessor System

The subsystem representing *series* components implies that failure of any component results in failure of that subsystem. The subsystem representing *parallel* components implies that only the failure of all the components results in failure of that subsystem. Figure 2 shows the RBD model for the fault-tolerant multiprocessor.

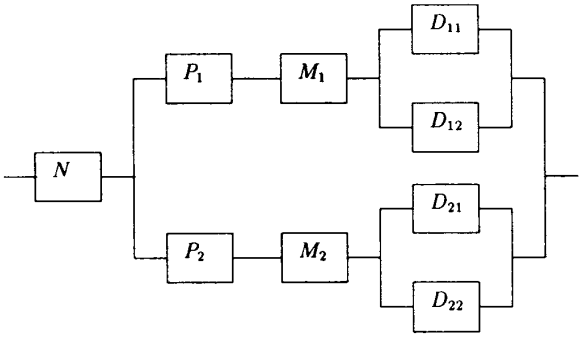


Figure 2. RBD Model of the Multiprocessor System

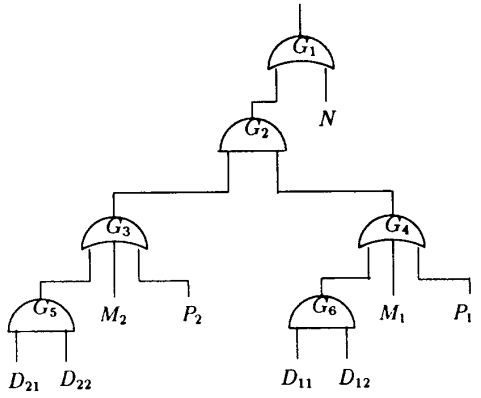


Figure 3. FT Model of the Multiprocessor System

Some researchers have used RBD with repeated blocks [6, 7]. However, we use RBD without repeated blocks.

3.2 Fault Trees Without Repeated Events

Like RBD, a FT is also a combinatorial-model type and maps the operational dependency of a system on its components. However, unlike RBD, FT represent a probability-of-failure approach to system modeling [4]. The phrase 'without repeated events' means that inputs to all the gates are distinct. Figure 3 shows the FT model for the fault-tolerant multiprocessor. Failure of a component implies that the corresponding input to the gate becomes *True*. The output of an OR gate is *False* iff all inputs are *False*. The output of an AND gate is *True* iff all inputs are *True*. The output of the top gate in the FT tells whether the system is operational or not. We allow only AND & OR gates in the fault trees (FT & FTRE) we consider in this paper.

Many authors have proposed extensions to FT, *eg*, allowing repeated events. Some of them have included gates such as NOT, EXOR, PAND (priority AND), kOfn, and some special gates such as cold spare, functional dependency, and sequence enforcing [8]. Some of these extensions enhance the modeling power of fault trees and some simply increase the ease of use. We consider fault trees with repeated events next.

3.3 Fault Trees with Repeated Events

Fault trees in which 'gates are allowed to share inputs' are FTRE, and are more general than an FT. Consider a simple variation on the base multiprocessor system shown in figure 1: there is a shared memory M_3 between P_1 & P_2. Figure 4 shows the new system. If memory module M_i fails, then processor P_i uses memory module M_3 and continues to work; M_3 can also be shared by both P_1 & P_2 if both M_1 & M_2 fail. Neither RBD nor FT, as we have described, can model the dependability of this system. FTRE can model the dependability of this system as shown in figure 5. This example shows that FTRE possess higher modeling power than FT or RBD because any RBD or FT model can also be modeled by FTRE. As mentioned in section 3.1, RBD with repeated blocks [6, 7] (analogous to the repeated events in a fault tree) have been used. RBD with repeated blocks have the same modeling power as FTRE.

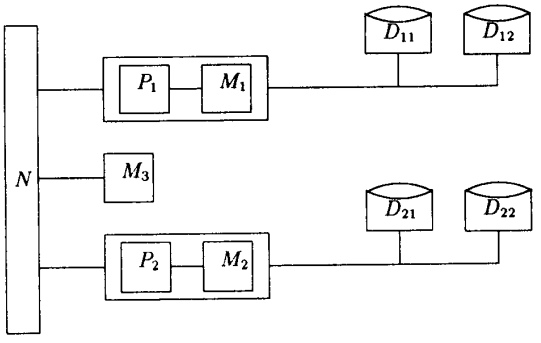


Figure 4. Multiprocessor System with Shared Memory

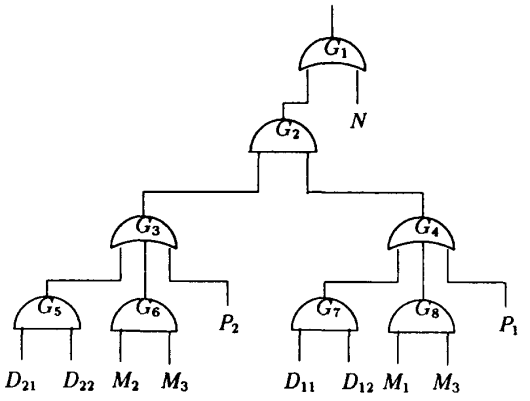


Figure 5. FTRE Model of the Multiprocessor System with Shared Memory

3.4 Reliability Graphs

Graphs have been extensively used as model types to model network reliability [9]. We consider a special class of digraphs as a model type in the software tool SHARPE [10]. A reliability graph $G = (U,V)$ is an acyclic digraph. There are two special nodes labeled *source* & *sink* in U. The *source* node has no incoming edges and the *sink* node has no outgoing edges. V has 2 kinds of edges: component-edges and ∞-edges. For each component, there is at most 1 edge in the RG, *ie*, repeated edges are not allowed. Failure of a component is indicated by failure of an edge; ∞-edges and nodes do not fail. The system modeled by an RG is operational as long as there is at least 1 path with no failed edge from the *source* node to the *sink* node. Figure 6 shows the RG of the multiprocessor system with shared memory.

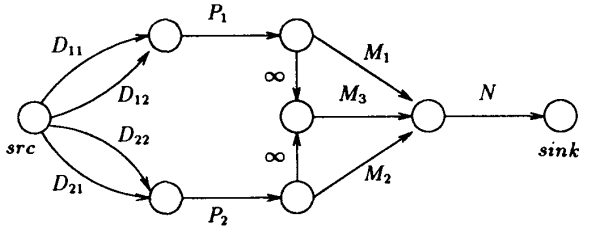


Figure 6. RG Model of the Multiprocessor System with Shared Memory

4. HIERARCHY AMONG COMBINATORIAL MODEL TYPES

This section establishes power-hierarchy among the 4 combinatorial-model types considered so far (RBD, FT, FTRE, RG) by proving either *a*) that every instance of a model type can be converted into an instance of another model type, or *b*) the counter-assertion. Proofs in case *a* are constructive, *ie*, we provide an algorithm for converting an instance of a model type into an instance of another model type. Proofs in case *b* are given by providing a counter-example, *viz*, showing an instance of a model type for which no equivalent instance of another model type exists.

4.1 FT to RBD

Shooman [4] has shown that a RBD is equivalent to a FT without repeated events, but did not provide a conversion algorithm. It is simple to do so. Comparison of the RBD in figure 2 and the FT in figure 3 reveals the similarities among the two model types. These similarities provide the algorithm for converting a FT model to an equivalent RBD model. The algorithm is given in PASCAL-like pseudo-code in figure 7. The FT is converted to the equivalent RBD by starting the algorithm as FT_to_RBD(*root*) where *root* is the root node of the FT.

```
Algorithm FT_to_RBD(x)
begin
  if (x ≠ NIL)
  then if (x == AND gate)
    then x ← PARALLEL construct
    else if (x == OR gate)
    then x ← SERIES construct
    foreach y ∈ child[x] do
      FT_to_RBD(y)
end
```

Figure 7. Conversion Algorithm for FT to RBD

This algorithm is simply a preorder traversal of the FT. If the node encountered is a gate (AND or OR), then it is converted to appropriate construct (PARALLEL and SERIES, respectively). If the node is a component (a leaf), then do nothing. This yields the RBD. There are n leaves in the FT; thus there are at most $n-1$ gates. Every step of the algorithm uses $O(1)$ time. Hence, the time complexity of this algorithm is $O(n)$.

4.2 RBD to FT

An RBD can be similarly converted to a FT in a reciprocal fashion. The algorithm is in figure 8. The time-complexity of this algorithm is $O(n)$ as well.

```
Algorithm RBD_to_FT(x)
begin
  if (x ≠ NIL)
  then if (x == PARALLEL construct)
    then x ← AND gate
    else if (x == SERIES construct)
    then x ← OR gate
    foreach y ∈ child[x] do
     RBD_to_FT(y)
end
```

Figure 8. Conversion Algorithm for RBD to FT

4.3 FT to RG

We now show that any FT can be converted to a RG. We prove this claim by an algorithm to convert a FT to an equivalent RG.

Assumption

1. In a FT, a node is either a gate or a component. ◀

The algorithm is in pseudo-code in figure 9. We briefly describe it below.

```
Algorithm FT_to_RG(x)
begin
  if (x ≠ NIL)
  then if (x = = root)
    then insert edge (source, sink) labeled root in V_r,g
    else if (x = = AND gate)
    then delete directed edge (u,v) labeled x from V_r,g
      foreach y ∈ child[x] do
        insert directed edge (u,v) labeled y in V_r,g
    else if (x = = OR gate)
    then delete directed edge (u,v) labeled x from V_r,g
      foreach y ∈ child[x] do
        if (y is the last remaining child of x) then w ← v
        insert directed edge (u,w) labeled y in V_r,g
        u ← w
    foreach y ∈ child[x] do
      FT_to_RG(y)
end
```

Figure 9. Conversion Algorithm for FT to RG

Initialize the RG consisting of only *source* & *sink* nodes connected by an edge labeled with the name of the root node (a gate) of the FT. Now perform a preorder traversal of the FT starting from the root node. The node at any step of the tree traversal is the current node. Let the current node be a gate labeled G with k inputs. If it is an AND gate, then replace the directed edge (u,v) (directed from u to v) labeled G in the partially constructed RG by k edges $e_1,e_2,\ldots,e_k$ between u & v with the same direction as the original edge. Assign the labels of the nodes (gates or components) from which the inputs to gate G are coming from to each of these edges. If it is an OR gate with k inputs, then replace the edge (u,v) labeled G in the partially constructed RG by a path $p = (e_1,e_2,\ldots,e_k)$ of k edges where $e_1 = (u,w_1)$, $e_2 = (w_1,w_2)$, ..., $e_k = (w_{k-1},v)$. Assign the labels of the nodes (gates or components) from which the inputs to gate G are coming, to each of these edges. If it is a component, then do nothing. Continue with preorder tree traversal until all the nodes have been traversed.

This algorithm yields an equivalent RG. For n components in the FT, there are at most n−1 gates (internal nodes). For each internal node, an edge in RG is inserted once and deleted once. For each external (leaf) node, an edge is inserted in the RG once. An edge in the partially constructed RG can be identified by its label in $O(1)$ time, by maintaining an array of pointers indexed by the label of the gate/component. Hence deletion & insertion of each edge in the partially constructed RG can be done in $O(1)$ time. This implies that for each node encountered in the preorder traversal of the FT, $O(1)$ work is done. The time complexity of this algorithm is therefore $O(n)$ since the FT consists of at most 2n−1 nodes.

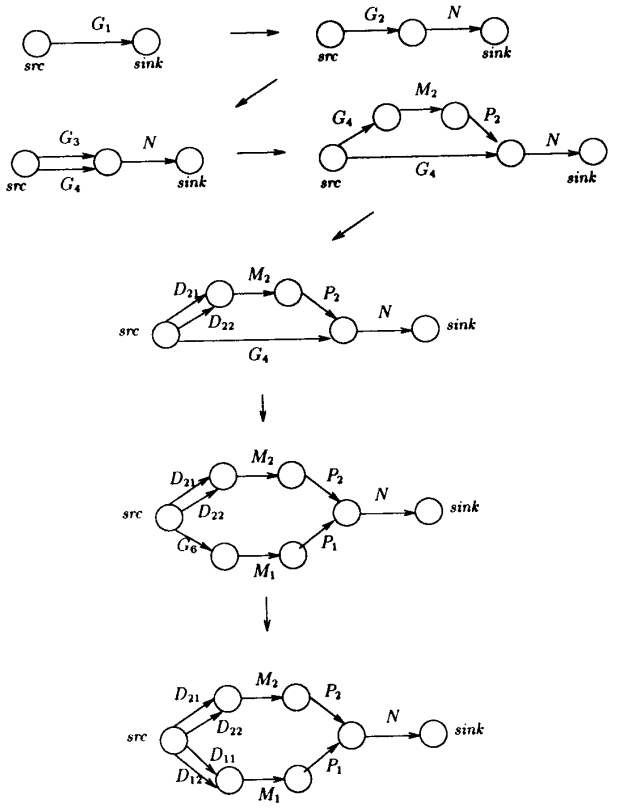


Figure 10. Conversion of the FT Model of Multiprocessor System to RG Model

For the FT model of the multiprocessor system in figure 3, figure 10 shows the steps taken by this algorithm. Due to preorder traversal of this FT, the nodes are looked at in the following order:

$$(G_1, G_2, G_3, G_5, D_{21}, D_{22}, M_2, P_2, G_4, G_6, D_{11}, D_{12}, M_1, P_1, N).$$

Omitting the nodes,

$$(D_{21}, D_{22}, M_2, P_2, D_{11}, D_{12}, M_1, P_1, N),$$

corresponding to the components (since we do nothing at these nodes), we are left with gates,

$$(G_1, G_2, G_3, G_5, G_4, G_6).$$

The first (top) element of figure 10 shows the initialization of the RG. Each subsequent element shows the partially constructed

RG at the end of each step when the gates are encountered in the order $(G_1, G_2, G_3, G_5, G_4, G_6)$.

4.4 RG to FT

We have shown that any FT can be converted to an equivalent RG. Since RBD & FT can be converted to one another, it implies that any RBD can also be converted to an equivalent RG. We now show that the converse is not true: not every RG can be converted to an equivalent FT. We prove this by showing that an example RG (shown in figure 11) cannot be converted to an equivalent FT.

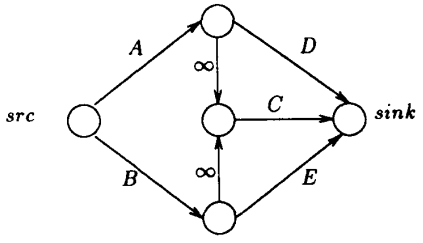


Figure 11. An Example RG Model Which Cannot be Converted to a FT

For every RG, there exists an equivalent Boolean expression that captures the operational dependence of the modeled-system on the system-components. The literals of this Boolean expression represent the system-components. In the RG in figure 11 the components are labeled A,B,C,D,E. Assign to each component a logical value of '0 when it is operational' and '1 when it is failed'; then the Boolean expression for this RG is:

$$S = (A+D)(A+C)(B+C)(B+E). \quad (1)$$

When the value of (1) is 0, the system is operational; otherwise the system is failed. Following the principles of minimal OR-AND (or AND-OR) realization (where the total number of inputs to a gate is minimized) of a Boolean expression [11], one can show that (1) cannot be reduced to an equivalent form in which all literals occur only once. Therefore (1) cannot be represented by a FT (which is similar to a combinational circuit) in which no input is repeated.

We have shown, by an example, that not every RG can be converted to an equivalent FT. Since FT are equivalent to RBD, this also proves that not every RG can converted to an equivalent RBD.

4.4 RG to FTRE

The basic idea behind this algorithm is to enumerate all the simple paths from the *source* node to the *sink* node. This is easily done using a breadth-first search [12]. For every path, construct an OR gate with inputs from all the components which appear on this path (∞-edges are ignored). Then construct an AND gate (root gate) such that the output of each OR constructed in the previous step is input to this gate. This is the equivalent FTRE for the RG. Events are repeated if different paths are not edge-disjoint, *ie*, different paths have edges in common. All the paths can be enumerated using a naive algorithm which takes exponential ($O(2^e)$) time (e is the number of edges in the RG) since there could be as many as $O(2^e)$ distinct paths. More sophisticated algorithms which yield more compact FTRE can be derived based on min-paths of the RG which can be computed using the Tarjan [13] algorithm. The pseudo-code of a generic algorithm to convert a reliability graph into an FTRE which uses some path enumeration algorithm is in figure 12. Our aim is simply to establish that a RG can be converted to a FTRE, not to provide optimal algorithms for conversion.

```
Algorithm RG_to_FTRE(RG)
begin
  enumerate all the paths from source to sink in RG
  let the paths be P1,P2,...,PK
  let Pi = {ei:1,ei:2,....,eiNi}
  foreach path Pi do
    construct gate Gi ← OR ei:1,ei:2,....,eiNi
  construct gate Gsys ← AND G1, G2, ...., GK
end
```

Figure 12. Conversion Algorithm for RG to FTRE

The correctness of this algorithm can be argued as follows. A system modeled by a RG is operational as long as there is a path from *source* node to the *sink* node. A failed edge in a RG blocks all the paths it appears on. This can be stated as: a system is operational iff there is at least one path which has no failed edge. This is precisely what the above constructed FTRE captures. For the RG of the base architecture with shared memory (figure 6), the paths possible from the *src* to the *sink* are:

D_{11}, P_1, M_1, N

$D_{11}, P_1, \infty, M_3, N$

D_{12}, P_1, M_1, N

$D_{12}, P_1, \infty, M_3, N$

D_{21}, P_2, M_2, N

$D_{21}, P_2, \infty, M_3, N$

D_{22}, P_2, M_2, N

$D_{22}, P_2, \infty, M_3, N$

Figure 13 shows the equivalent FTRE, as constructed by algorithm RG_to_FTRE. It must be realized however, that a shared edge among different paths in a RG does not *always* imply that an equivalent FT (without repeated events) does not exist. An example of this appears in figure 14, where the edge labeled C appears on both the paths from *src* to *sink*.

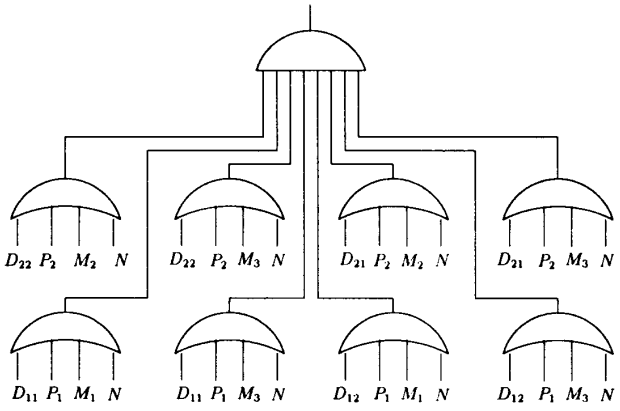


Figure 13. Conversion of the RG Model of Multiprocessor System to FTRE

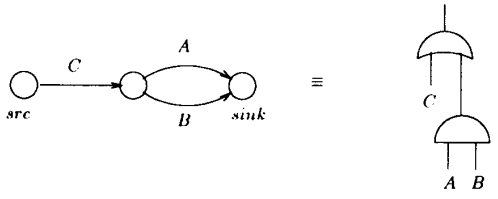


Figure 14. An Equivalent FT without Repeated Nodes for an RG with a Shared Edge

4.6 FTRE to RG

Section 4.5 proved that any RG can be converted to an equivalent FTRE. This section shows that the converse is not true: there does not always exist an equivalent RG for every FTRE. We prove this claim by means of a counter-example. Consider a TMR system with 3 components A,B,C. The system is operational as long as at least 2 components are operational. Figure 15 shows a FTRE model of this system.

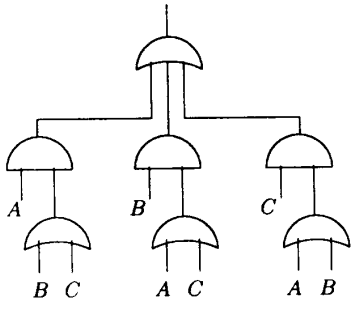


Figure 15. FTRE Model of a TMR System

This FTRE can not be converted to a RG. To prove this, assume that an equivalent RG does exist. Let the component-edges representing components A,B,C in this RG be $e_a = (u_a,v_a)$, $e_b = (u_b,v_b)$, $e_c = (u_c,v_c)$, Let $p_{a,b}$, $p_{b,c}$, $p_{a,c}$ be the paths from the *source* to the *sink* which includes edges (e_a,e_b), (e_b,e_c), (e_a,e_c) respectively. Thus,

$$p_{a,b} = (p_{1:a,b}, e_a, p_{2:a,b}, e_b, p_{3:a,b}),$$

$$p_{a,c} = (p_{1:a,c}, e_a, p_{2:a,c}, e_c, p_{3:a,c}),$$

$$p_{b,c} = (p_{1:b,c}, e_b, p_{2:b,c}, e_c, p_{3:b,c}),$$

$p_{i:a,b}$, $p_{i:b,c}$, $p_{i:a,c}$ $(i = 1,2,3)$ are paths consisting only of ∞-edges. The paths $p_{a,b}$, $p_{b,c}$, $p_{a,c}$ are not necessarily edge-disjoint, *ie*, there could be edges common to more than one path. Figure 16 shows this RG.

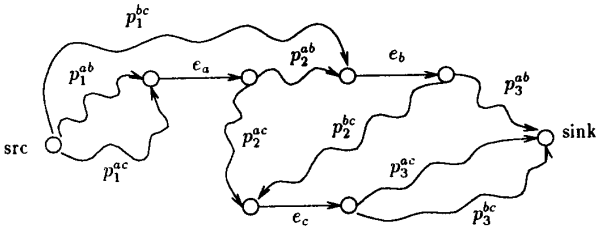


Figure 16. Counter-Example for Conversion of an FTRE to an RG

Given these 3 paths, there exists another path from *source* to *sink*:

$$p_b = (p_{1:b,c}, e_b, p_{3:a,b}). \quad (2)$$

This p_b consists of only 1 component-edge e_b; the rest are ∞-edges. This implies that there exists a path from the *source* to the *sink* such that it consists of only 1 component edge. This in turn implies that the system is operational if B is operational even though both A & C can have failed. This contradicts our assumption about the operational dependency of the system on A,B,C. Therefore, the assumption that an equivalent RG exists for this FTRE is incorrect. *Q.E.D.*

It can be similarly shown that any system which has a kOFn gate where $k >$ liub(½n) (at least k out of n components must be operational for the system to be operational) can not be modeled by a RG.

As another example, consider the multiprocessor system with shared memory. If we impose the constraint that memory module M_3 can be used by only one processor P_1 or P_2, then failure of only one memory module M_1 or M_2 could be tolerated, *ie*, failure of both M_1 & M_2 leads to system failure. It is easy to construct the FTRE model for such a system. However, we claim that a RG model for this system can not be constructed.

We have established the hierarchy among combinatorial-model types — from most powerful to least powerful:

FTRE

RG

RBD/FT (these are equivalent to each other).

5. MARKOV MODEL TYPES

This section considers some Markov-model types used for dependability modeling:

- continuous-time Markov chains
- generalized stochastic Petri nets
- Markov reward models
- stochastic reward nets.

Markov models can handle some dependencies, in a system, which combinatorial models cannot [2, 3]. For example, consider FTRE model of the multiprocessor system. If we wished to model the system availability when there is a single repair person for all the components, then we could not do so by an FTRE. Such repair-dependency can not be modeled by any combinatorial-model type, but a Markov chain or a GSPN can easily model such dependency. Dugan *et al* [14] show that an FTRE model can be converted to a CTMC. This establishes that CTMC are more powerful than FTRE.

Most of the algorithms for converting Markov-model types into each other are known because GSPN (SRN) are solved by converting them into CTMC (MRM) and solving them. However, for completeness, we briefly discuss various Markov-model types and the conversions between them. For a comparison of CTMC & MRM and a discussion of how MRM can be used for reliability analysis using special reward rate assignments, refer to [15].

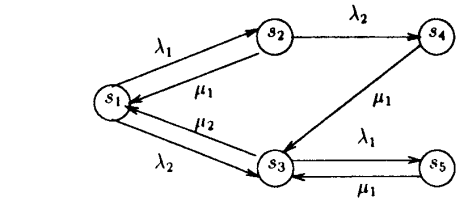
5.1 GSPN to CTMC

Ajmone-Marsan *et al* [16] showed that CTMC are equivalent to GSPN: for every GSPN model, an equivalent continuous-time Markov chain exists and vice-versa. They provided an algorithm which converts a GSPN to a CTMC. Basically, a reachability graph of the GSPN is constructed. Each tangible marking of this reachability graph becomes a state of the equivalent CTMC. For more details, refer to [16].

5.2 CTMC to GSPN

Converting a CTMC to a GSPN model is fairly obvious. For each state s_i of the CTMC, construct a place p_i. Replace each arc (s_i,s_j) with rate $r_{i,j}$ of the Markov model by a transition $t_{i,j}$ of rate $r_{i,j}$ with an incoming arc from p_i and an outgoing arc to p_j. If there is a single initial state s_{init} of the CTMC, then a single token must be put in p_{init}. If the CTMC has several initial states $s_k,s_{k+1},\ldots,s_m$ with probabilities $\omega_k,\omega_{k+1},\ldots,\omega_m$, then a new place p_0 is created with a single token, and p_0 is connected to $p_k,p_{k+1},\ldots,p_m$ via immediate transitions $t_{0,k},t_{0,k+1},\ldots,t_{0,m}$ with probabilities $\omega_k,\omega_{k+1},\ldots,\omega_m$.

Figure 17a shows the CTMC of a system that consists of 2 components C_1 & C_2 which share a repair facility with priority repair discipline. The failure rate of C_i is λ_i and the repair rate is μ_i. C_1 has repair priority over C_2: if C_1 fails while C_2 is being repaired, then repair of C_2 is preempted, repair of C_1 begins, and the repair of C_2 resumes after C_1 is repaired. Assume further that this system can be in one of states 1,2,3 at time zero with probabilities $\omega_1,\omega_2,\omega_3$, respectively; figure 17b shows the equivalent GSPN model.



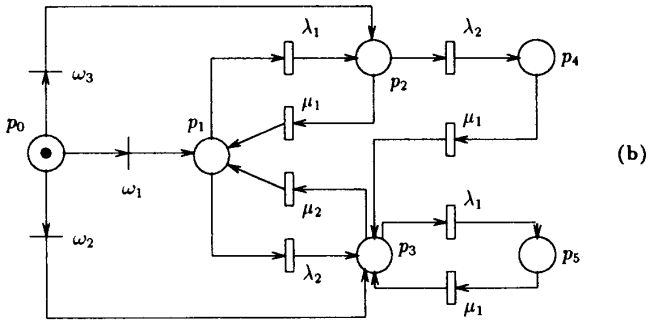


Figure 17. Converting a CTMC to a GSPN

5.3 SRN to MRM

Ciardo *et al* [17] formalize SRN and provide an algorithm to convert an SRN into an equivalent MRM. In an SRN, the rewards are specified using a reward function. A typical reward function is: "If there are x tokens in place p, then reward rate is r_x, else if there are y tokens in place p, then reward rate is r_y, else reward rate is r_z." Based on this reward function, a reward rate is associated with each tangible marking of the SRN. Each tangible marking corresponds to a state of the underlying CTMC. When reward rates are associated with each state of the CTMC, it becomes an MRM.

5.4 MRM to SRN

Conversion from MRM to SRN is divided in two parts:

1. Convert the states of the MRM to the SRN structure of places & transitions. This is done exactly the same way as a CTMC is converted to a GSPN (see section 5.2).
2. Convert the reward rate specification of the MRM to reward rate specification of the SRN. ◄

We describe this procedure using the example in figure 17. Let r_i be the reward rate associated with state s_i of the CTMC in figure 17a. According to the procedure in section 5.2 there is only one token in the resulting SRN, and for each state s_i of the MRM, there is a place p_i in the SRN. The equivalent reward rate function for the SRN is then straight forward. It has the following form for all possible values of i: "If there is one token in place p_i, then reward rate is r_i".

6. POWER HIERARCHY

Assumption

2. Failure & repair time distributions of system components are exponentially distributed. ◄

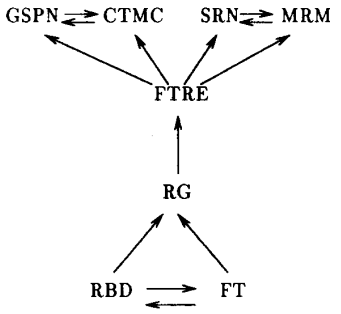


Figure 18. Power Hierarchy Among Dependability-Model Types

Figure 18 shows the overall hierarchy of dependability-model types as established in sections 3 - 5. However, tradeoffs exist between Markov & combinatorial-model types. For instance, to arrive at this power hierarchy, we added assumption 2. Whereas analysis of combinatorial-model types does not put any restrictions on the nature of distributions, the Markov-model types can be extended but usually become intractable under non-exponential distributions. Semi-Markov chains have often been used in reliability modeling to allow for non-exponential distributions but only in a restrictive manner. Markov regenerative processes hold more promise there [18].

There also exists a tradeoff in the kinds of dependability measures that can be computed from different model types. For example, availability & reliability can be computed relatively easily using combinatorial-model types. However, computing the system mean time to failure using combinatorial-model types can be quite complex if distributions are non-exponential. Availability of repairable systems can be modeled by combinatorial-model types only under the assumption that each system component has an s-independent repair person. However, Markov-model types can model the scenarios where a repair person is shared among various system components.

ACKNOWLEDGMENT

This work was done while the first author (M.M.) was at Duke University, and was supported in part by the US National Science Foundation under Grant CCR-9108114 and by the US Naval Surface Warfare Center under grant N60921-92-C-0161.

We are pleased to thank Dr. Robin Sahner for useful comments on an earlier draft of this paper. The referees made useful suggestions which resulted in important improvements. One of the referees pointed out that reliability block diagrams with repeated blocks have been used in the past.

REFERENCES

[1] J. Laprie, "Dependable computing and fault-tolerance: Concepts and terminology", *Proc. 15th Int'l Symp. Fault-Tolerant Computing*, 1985, pp 2-7.

[2] A. Johnson, M. Malek, "Survey of software tools for evaluating reliability, availability, and serviceability", *ACM Computing Surveys*, vol 20, 1988 Dec, pp 227-269.

[3] A. Reibman, M. Veeraraghavan, "Reliability modeling: An overview for system designers", *IEEE Computer*, 1991 Apr, pp 49-57.

[4] M. Shooman, "The equivalence of reliability diagram and fault-tree analysis", *IEEE Trans. Reliability*, vol R-19, 1970 May, pp 74-75.

[5] G. Hura, J. Atwood, "The use of Petri nets to analyze coherent fault-trees", *IEEE Trans. Reliability*, vol 37, 1988 Dec, pp 469-473.

[6] J. Buzacott, "Finding the MTBF of repairable systems by reduction of the reliability block diagram", *Microelectronics & Reliability*, vol 6, 1967, pp 105-112.

[7] J. Buzacott, "Network approaches to finding the reliability of repairable systems", *IEEE Trans. Reliability*, vol R-19, 1970 Nov, pp 140-170.

[8] J. Dugan, S. Bavuso, M. Boyd, "Dynamic fault-tree models for fault-tolerant computer systems", *IEEE Trans. Reliability*, vol 41, 1992 Sep, pp 363-377.

[9] C. Colburn, *The Combinatorics of Network Reliability*, 1987; Oxford University Press.

[10] R. Sahner, K. Trivedi, "A software tool for learning about stochastic models", *IEEE Trans. Education*, vol 36, 1993 Feb, pp 56-61.

[11] A. Friedman, P. Menon, *Theory and Design of Switching Circuits*, 1975; Computer Science Press.

[12] T. Cormen, C. Leiserson, R.L. Rivest, *Introduction to Algorithms*, 1990; MIT Press, McGraw Hill.

[13] R. Tarjan, "A unified approach to path problems", *J. ACM*, vol 28, 1981, pp 577-593.

[14] J. Dugan, K. Trivedi, M. Smotherman, R. Geist, "The hybrid automated reliability predictor", *AIAA J. Guidance, Control, Dynamics*, 1986 May-Jun, pp 319-331.

[15] K. Trivedi, M. Malhotra, "Reliability & performability techniques & tools", *Proc. 7th ITG/IG Conf. Measurement, Modeling, Evaluation of Computer & Communication Systems*, 1993 Sep, pp 27-48.

[16] M. Ajmone-Marsan, G. Conte, G. Balbo, "A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems", *ACM Trans. Computer Systems*, vol 2, num 2, 1984, pp 93-122.

[17] Ciardo, Blakemore, Chimento, Muppala, Trivedi, "Automated generation & analysis of Markov reward models using stochastic reward nets", *"Linear Algebra, Markov Chains, Queueing Models*, IMA volumes in Mathematics and its Applications (Meyer & Plemmons, Eds), vol 48, 1993; Springer-Verlag.

[18] H. Choi, V.G. Kulharni, K.S. Trivedi, "Markov regenerative stochastic Petri nets", *Proc. 16th IFIP W.G. 7.3 Int'l Symp. Computer Performance Modeling, Measurement, Evaluation* (Performance '93), 1993 Sep.

AUTHORS

Dr. M. Malhotra; AT&T Bell Labs; 101 Crawfords Corner Rd; Holmdel, New Jersey 07747 USA.
Internet (e-mail): manish@hogpa.att.com

M. Malhotra received the BTech (1989) in Computer Science from the Indian Institute of Technology, Delhi, and the MS (1990) and PhD (1993) in Computer Science from Duke University, Durham. Since 1993 June, he has been a member of technical staff at AT&T Bell Labs, Holmdel. His interests include reliability & performability modeling of telecommunication networks & services.

Dr. K.S. Trivedi; Dept. of Electrical Engineering; Duke University; Durham, North Carolina 27706 USA.

Kishor S. Trivedi (M'86, SM'87, F'92) received the BTech degree from the Indian Institute of Technology (Bombay), and MS & PhD in Computer Science from the University of Illinois, Urbana-Champaign. He is the author of *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. His research interests are in computer & communication system reliability & performance modeling. He has lectured & published extensively on these topics. He is a Fellow of the IEEE. He is a Professor of Electrical Engineering at Duke University, Durham. He has served as a Principal Investigator on various AFOSR, ARO, Burroughs, IBM, DEC, NASA, NIH, ONR, NSWC, Boeing, Union Switch & Signals, NSF, and SPC funded projects and as a consultant to industry and research laboratories. He was an Editor of the *IEEE Transactions on Computers* from 1983-1987. He is a co-designer of the widely circulated HARP, SAVE, SHARPE, and SPNP modeling packages.

Manuscript received 1994 March 18.

IEEE Log Number 94-06135 ◄TR►