Exploiting Petri Nets to Support Fault Tree Based Dependability Analysis

Andrea Bobbio[1], Giuliana Franceschinis[1], Rossano Gaeta[2], Luigi Portinale[1]
[1]Dipartimento di Scienze e di Tecnologie Avanzate, Università del Piemonte Orientale
Corso Borsalino 54, 15100 Alessandria, Italy
[2]Dipartimento di Informatica, Università di Torino
Corso Svizzera 185, 10149 Torino, Italy
{bobbio,giuliana,rossano,portinal}@di.unito.it

Abstract

This paper explores the possibility of converting Fault Trees (FT) *into the* Generalized Stochastic Petri Net (GSPN) *formalism. Starting from a slightly modified version of a conversion algorithm already appeared in the literature, the aim of the paper is to exploit the modeling and decision power of GSPN for both the qualitative and the quantitative analysis of the modeled system. The qualitative analysis resorts to structural properties and is based on a T-invariant analysis. In order to alleviate the state space explosion problem deriving from the quantitative analysis, the paper proposes a new formalism for FT, that is referred to as* High Level FT (HLFT), *in which replicated redundant units are folded and indexed. Starting from the HLFT formalism, a new conversion algorithm is provided that translates a HLFT into a* Stochastic Well-formed Net (SWN). *The computational saving of using SWN with respect to GSPN is carefully examined considering an example of a fault-tolerant multiprocessor system.*

1. Introduction

Model types used in dependability analysis can be divided in two classes: combinatorial models and state-space based models. Combinatorial models assume that components are statistically independent and have a poor modeling power coupled with higher analytical tractability. Among combinatorial models, *Fault Trees (FT)* have become very popular in the dependability analysis of large safety-critical systems [1, 2]. The goal of FT analysis (FTA) is to represent the combination of elementary causes that lead to the occurrence of an undesired catastrophic event, denoted as the *Top Event (TE)*. In FTA the analysis is carried out in two steps: a qualitative step in which the list of all the possible combinations of events (the *minimal cut sets* MCS) that give rise to the TE is determined. If probability values can be assigned to all the events appearing in the tree, the quantification step can be undertaken and the probability of occurrence of the TE, and of the other nodes of the tree, can also be calculated.

The qualitative analysis step is considered of crucial importance in dependability analysis and safety studies, since it allows the analyst to enumerate all the possible causes of failure for the system and to rank them according to a very simple severity measure given by the order of the cutset.

The major weak point of the FTA methodology (as well as of any combinatorial technique), is the fact that the events must be considered as statistically independent. State space based approaches (like Markov models) overcome this limitation, but require the solution to be computed over a set of equations whose number increases exponentially with the number of components. Moreover, the generation of the state space and of the associated infinitesimal generator, needs to be supported by automatic tools. The language of Generalized Stochastic Petri Nets (GSPN) can be conveniently used for this purpose [3] since it allows to describe large state spaces by a restricted number of model primitives (places, transitions and tokens).

Hura and Atwood in [4] have proposed the use of PN to analyze coherent fault trees; Malhotra and Trivedi have provided in [5], an algorithm to convert a fault tree into a GSPN or its variant Stochastic Reward Nets (SRN), and have shown how various kinds of dependencies can be accommodated into the so obtained model. However, in the methodology proposed in [5], two peculiarities of the FTA are lost: *i)* the qualitative analysis is not considered; *ii)* the quantitative analysis requires the generation of the complete state space (incurring easily in the space explosion problem).

The purpose of the present paper is twofold. First we want to show how to respond to the first point (the qualitative analysis), by exploiting structural properties of the GSPN. In particular, it is shown how the MCS can be generated resorting to a T-invariant analysis.

The second goal aims at alleviating the state explosion problem by observing that high dependability is obtained through replication that induces symmetries in the system layout. To account for symmetries, a new formalism for FT, referred to as High Level FT (HLFT), is proposed in which replicated units are folded and indexed. Starting from the new HLFT formalism, an algorithm is provided that translates a HLFT into a Stochastic Well-formed Net (SWN). The computational saving of using SWN with respect to GSPN is carefully examined. Moreover, using the wider modeling flexibility of SWN, we can include in the model dependencies that could not have been accommodated into the corresponding FT.

In order to illustrate the new achievements, the paper considers an example of a fault-tolerant multiprocessor system taken from [5]. The conversion from the FT to the GSPN representation is carried on in Section 2, while the qualitative analysis based on T-invariants is discussed in Section 3. Section 4 introduces the new HLFT formalism and its conversion into a SWN. Finally, Section 5 presents some numerical results, considering also the case where statistical dependencies, due to the presence of a coverage factor, are included into the model.

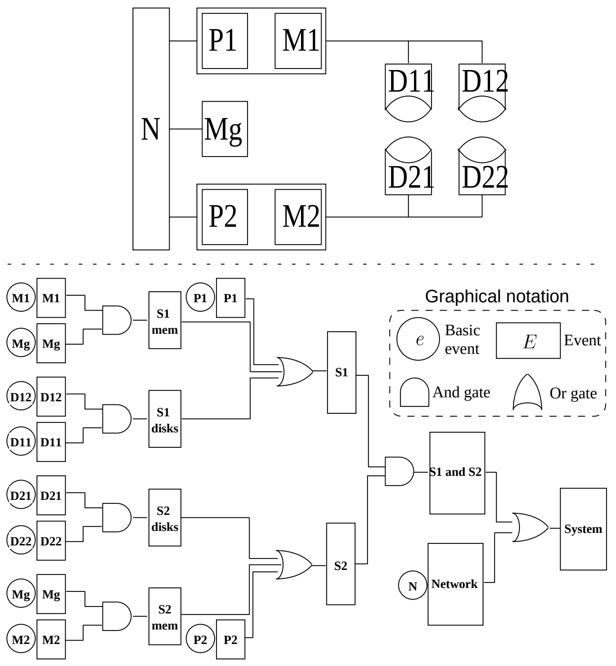

Figure 1. Diagram of the multiprocessor (top) and corresponding FT (bottom).

2. Conversion of a FT into a GSPN

In order to present the extension of HLFT, a formal definition of a FT is provided. The definition does not include the possibility that the FT contains the *NOT* gate and related gates (e.g. *XOR*).

Definition 1 (Fault Tree) *A fault tree (FT) is a tree comprising four types of nodes:* basic events, events, *and two types of elementary logical gates, the* AND *and the* OR *gate (see Figure 1).*

The root of a FT is an event called the Top Event (TE). Each event has exactly one successor which can be either a basic event or a gate. A gate has two or more successor events. The leaves of the tree are all basic events.

We apply the above definition to the following example [5] that will be used throughout the paper. A fault-tolerant multiprocessor system, whose block diagram is depicted in the top part of Figure 1, comprises two independent subsystems $S1$ and $S2$ with a shared common memory Mg. Each subsystem Si ($i = 1, 2$) is composed by one processor Pi one local memory Mi and two replicated disk units $Di1$ and $Di2$. A single bus N connects the two subsystems and the shared common memory.

Assuming as the TE the complete system failure, the FT modeling the system operation is given in the bottom part of Figure 1. Standard qualitative analysis provides the following logical expression for the TE:

$$\begin{aligned}TE = {} & \overline{N} + \overline{D}11\overline{D}12\overline{D}21\overline{D}22 + \overline{D}11\overline{D}12\overline{M}2\overline{Mg} + \\ & + \overline{D}11\overline{D}12\overline{P}2 + \overline{M}1\overline{Mg}\overline{D}21\overline{D}22 + \overline{M}1\overline{M}2\overline{Mg} + \\ & + \overline{M}1\overline{Mg}\overline{P}2 + \overline{P}1\overline{D}21\overline{D}22 + \overline{P}1\overline{M}2\overline{Mg} + \overline{P}1\overline{P}2 \qquad (1)\end{aligned}$$

where $\overline{C}$ means the failure of the component whose label is C. Each term in the right-hand side of (1) represents a MCS i.e. a combination of basic events whose simultaneous occurrence implies the TE.

A conversion algorithm from FT to GSPN has been presented in [5]. We propose slight variations that are summarized in Figure 2. For each basic element of the FT (event, ANDgate and ORgate), the top part of the figure shows the conversion rule proposed in [5] while the bottom part the new conversion rule. Applying the new rules, the FT of Figure 1 is converted into the GSPN of Figure 3 (except transition $S.f$ that will be needed later on). Places whose names have the suffix $.dn$ model components in the non-working condition. A token in place $S.dn$ models the TE. Transitions whose names have the suffix $.f$ model the fault of a component while all the other transitions model the combinatorial logic derived from the conversion from FT to GSPN. The initial marking of the net represents the multiprocessor having all working components.

The main advantage of the new rules is that the arc multiplicities required in [5] are no more needed. The elimination of arc multiplicities simplifies both the qualitative

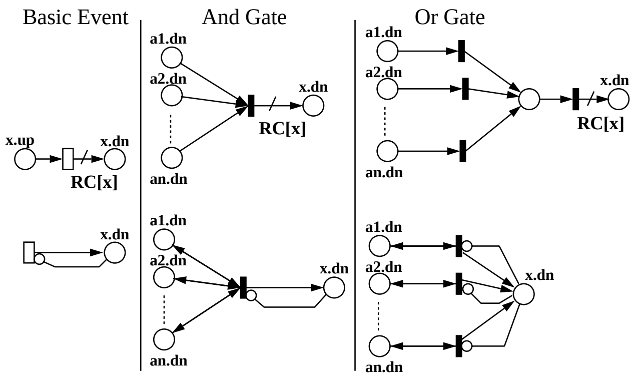

Figure 2. Modified FT to GSPN translation.

analysis in Section 3 and the high level description in Section 4. Other useful side-effects, not available in [5], are that the resulting nets are 1-bounded, and the failure markings encode the complete information on the subsystem failures occurred so far (in fact once a place $x.dn$ becomes marked, it will never lose its token).

3. Minimal cutset determination by T-invariant analysis

Qualitative analysis on FT is usually done by considering the *minimal cut sets* (MCS), representing the prime implicants of the TE (i.e. the system failure) expressed in terms of basic events (i.e. the failure of elementary system components).

Structural analysis of Petri nets offers a powerful tool for checking qualitative properties of the model. In this section, we will show that, the GSPN model obtained by the conversion procedure introduced in the previous section can be easily simplified in such a way that T-invariant analysis can be performed, in order to characterize the MCS of the corresponding fault tree. This exploits previous results obtained in Petri net analysis of logic programs [7, 9] and in diagnostic problem solving using Petri nets [8]. The basic idea is that if the Petri net model describes the logical dependencies among a set of entities, the transitive closure of such dependencies can be obtained by considering the reachability of certain markings; in particular, since T-invariant analysis deals with the *reproducibility* of markings, the above problems can be faced by considering how a given marking (or a set of markings) may be reproduced.

In particular, the approach in [8] generalizes some results obtained in [7, 9] to diagnostic analysis, that is to the computation of the primary causes that explain (in logical terms, that *entail* or *derive*) a set of abnormal observations. The event corresponding to the occurrence of a primary cause C is modeled by means of a source transition t_C with $t_C^\bullet = \{P_C\}$, where P_C is the place corresponding to C; the occurrence of an abnormal observation F is, on the contrary, modeled by means of a sink transition t_F with $^\bullet t_F = \{P_F\}$, where P_F is the place corresponding to F. The work in [8] shows that every set of source transitions $\{t_{C_1}, t_{C_2}, \dots t_{C_k}\}$ contained in a T-invariant support also containing t_F is such that $C_1 \wedge C_2 \wedge \dots C_k \models F$. Such a result holds whenever the Petri net represents a particular set of logical implications called *definite clauses* having the form $A_1 \wedge A_2 \wedge \dots A_k \rightarrow B$ where each A_i and B are logical propositions [8]. In particular, each logical proposition is associated with a place and each clause to a transition having in the input the places corresponding to the left-hand side of the clause and in the output the place corresponding to the right-hand side. This means that every transition t has at most one outgoing arc (i.e. $\forall t$, $|t^\bullet| \leq 1$).

In FTA, the abnormal observation is the TE while primary causes are identified with basic events; determining the basic events that entail TE means to determine the cut sets of the fault tree. By considering now a fault tree, it is easy to see that each boolean gate can be modeled as a definite clause; in particular, AND gates will correspond to a single clause having its input events in the left-hand side and the output event in the right-hand side, while an OR gate will correspond to n clauses, each one having one of the n input events in the left-hand side and the same output event in the right-hand side. If we now consider the GSPN model of a FT obtained by the conversion rules discussed in Section 2, it is easy to see that it can be simplified, in order to obtain a model like the one discussed above.
In fact, the following simplifications can be adopted: *i)* eliminate inhibitor arcs; *ii)* eliminate self-loops (i.e. transform bi-directional arcs into input arcs to transitions). The system failure is modeled by a transition having as input the failure events of some sub-systems and in the output the place $S.dn$ corresponding to the TE. In order to exploit T-invariant analysis as done in [7, 8], a further model adjustment is needed: the sink place $S.dn$ is transformed in a non-sink place by adding a sink transition $S.f$ in the output (see Figure 3).

The model obtained satisfies the conditions of theorem 4 of [8] and the following proposition can then be stated as a corollary of the above theorem (which is in turn derived from two theorems proved in [7] and [9] respectively).

Proposition 1 *Given a fault tree FT, let Σ_{FT} be the GSPN model obtained by: (*i*) applying the conversion procedure of Section 2; (*ii*) deleting inhibitor arcs; (*iii*) substituting self-loops with arcs from the place to the transition; (*iv*) adding a sink transition $S.f$ such that $^\bullet S.f = \{S.dn\}$.*

A set E of basic events is a cut set for FT (i.e. $E \models$ TE) iff there exists a positive T-invariant X of Σ_{FT} such that $X(S.f) \neq 0$ and the set restriction of the support of X to source transitions is equal to E.

The intuitive reason of the above result is that, as shown in [8], T-invariants model the reproduction of the empty marking, i.e. the firing of the system failure transition as a consequence of the firing of component failure transitions. In other words, T-invariants represent the sequence of transitions that, starting from the empty marking (i.e. the marking with every place having zero tokens), will bring the net to the empty marking again; because of the acyclic net structure, the only way of getting such a result is to suitably fire source transitions (corresponding to basic events), in order to fire the sink transition (corresponding to TE) at the end. The supports of such invariants are actually the derivational trace of the TE from basic events, so restricting one of such supports to source transitions will give us a set of basic events entailing TE (i.e. a cut set).

Property 1 holds for every cut set of the fault tree, however one is usually interested in MCS, i.e. cut sets that do not contain a subset of events that is still a cut set. It is well-known that the same notion of minimality is defined for T-invariants with respect to their support (*minimal support T-invariants* or MST). However, MST are not guaranteed to correspond to MCS; the reason is that the notion of minimality in MST is defined with respect to all involved transitions, while for having a minimal cut-set we need to have minimality with respect to source transitions. For example, if t_1 and t_2 are two source transitions, sets $\sigma_1 = \{t_1, t_3, t_5\}$ and $\sigma_2 = \{t_1, t_2, t_4, t_5\}$ can represent two minimal supports of T-invariants, but σ_2 is not minimal if restricted to source transitions[1].

In order to determine MCS from the net model we have then to perform two separate steps: (*i*) compute MST of the net and extract from their supports the subset of source transitions; (*ii*) filtering out from the sets of transitions obtained in the previous step, non minimal ones.

Example. Consider the GSPN for the multiprocessor system described in Figure 3; the net model for T-invariant analysis is obtained from that net by deleting inhibitor arcs and self-loops and by adding the circled transition $S.f$. The minimal supports for T-invariants of this net (computed by means of the *GreatSPN* package [14]) are the following:
$\{P1.f, M2.f, Mg.f, S12.f, S.f, t6, M2g.f, t3\}$
$\{P1.f, D21.f, D22.f, S12.f, S.f, t6, D2.f, t2\}$
$\{P1.f, P2.f, S12.f, S.f, t6, t1\}$
$\{D11.f, D12.f, Mg.f, M2.f, S12.f, S.f, t5, D1.f, M2g.f, t3\}$
$\{D11.f, D12.f, D21.f, D22.f, S12.f, S.f, t5, D1.f, D2.f, t2\}$
$\{D11.f, D12.f, P2.f, S12.f, S.f, t5, D1.f, t1\}$
$\{M1.f, Mg.f, M2.f, S12.f, S.f, t4, M1g.f, M2g.f, t3\}$
$\{M1.f, Mg.f, D21.f, D22.f, S12.f, S.f, t4, M1g.f, D2.f, t2\}$
$\{M1.f, Mg.f, P2.f, S12.f, S.f, t4, M1g.f, t1\}$
$\{N.f, S.f, t7\}$

By restricting the above supports to source transitions

[1]This example corresponds to a very simple fault tree where the $TE = E_1 + E_1E_2$, being E_1, E_2 two basic events.

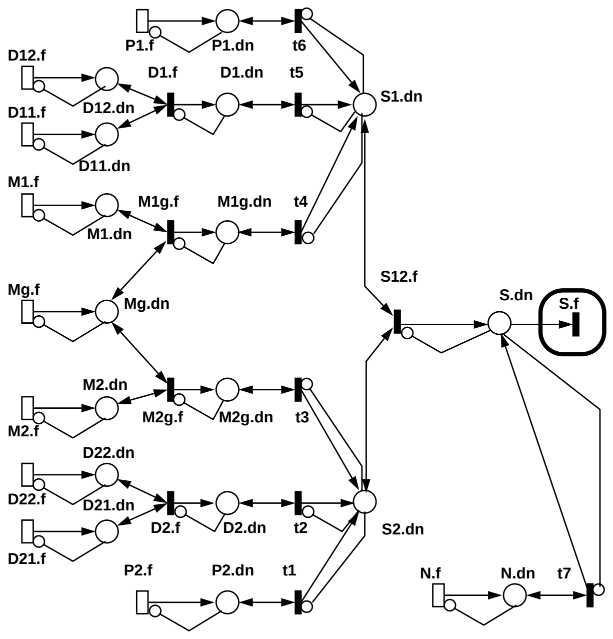

Figure 3. GSPN Model of the multiprocessor.

we obtain:
$\{P1.f, M2.f, Mg.f\}$ $\{P1.f, D21, f, D22.f\}$
$\{P1.f, P2.f\}$ $\{D11.f, D12.f, Mg.f, M2.f\}$
$\{D11.f, D12.f, D21.f, D22.f\}$ $\{D11.f, D12.f, P2.f\}$
$\{M1.f, Mg.f, M2.f\}$ $\{M1.f, Mg.f, D21.f, D22.f\}$
$\{M1.f, Mg.f, P2.f\}$ $\{N.f\}$
No further elaboration is then needed, since in this case it is easy to see that they actually correspond to the MCS i.e. to the terms of Equation 1.

4. High level fault trees

The basic idea behind HLFTs stems from the observation that often, due to the use of redundancy, a FT may contain several similar subtrees. In our running example we have two similar subtrees corresponding to the two subsystems $S1$ and $S2$, moreover, within each subsystem, there are two similar subtrees corresponding to the replicated disks.

To make the description more compact, we may *fold* the similar subtrees so that only one *representative* is explicitly included in the model. Such representative must abstract out from the identities of the events appearing in it: this can be done by using *parametric event identifiers*. For example, we may use $S(i)$, $P(i)$, $M(i)$, $D(i, j)$ as parametric identifiers for $S1$ and $S2$, $P1$ and $P2$, $M1$ and $M2$, $D11$, $D12$, $D21$ and $D22$. The information on how many copies of each subtree were present in the original FT can be expressed using an annotation associated with the root of each representative. To this purpose, we define a new type of node, called event *replicator* node.

Each event *replicator* node identifies a subtree representing several folded subtrees; such node is annotated with the *declaration* of one or more parameters, specifying their type. A *parameter type* is simply a finite, not empty set defining the range of possible values for the parameter. From the cardinality of parameter types declared in a given replicator node r, it is possible to derive the number of subtrees represented by the subtree of root r.

In our running example, the subtrees of root $S1$ and $S2$ can be folded into a subtree whose root is $S(i)$: this is an event replicator node where parameter i is declared, its type is $type(i) = C_1 = \{s_1, s_2\}$. Then we can fold the subtrees corresponding to events $D1(i)$ and $D2(i)$ obtaining a representative subtree whose root $D(i, j)$ is an event replicator node, including the declaration of parameter j whose type is $type(j) = C_2 = \{d_1, d_2\}$.

Observe that there might be basic events that are *shared* by some replicated subtrees, for example the same event Mg appears in both subsystems $S1$ and $S2$. The fact that the *same* shared memory appears in each subtree folded into the representative of root $S(i)$, is made explicit because the corresponding basic event node in the HLFT is *not* parametric. In Figure 4 the folded FT (i.e., the HLFT) of our running example is depicted.

Definition 2 (High Level Fault Tree) *A high level fault tree (HLFT) is a tree comprising five types of nodes:* basic events, events, event replicators, *and two types of logical gates, namely* AND *and* OR gates*.*

Basic events can be grouped into classes (of similar basic events): the generic basic event of a class can be expressed in parametric form using the notation name_of_class(list_of_parameters)*, where* (list_of_parameters) *is a list of (typed) parameters.*

An event is defined as in FTs. An event replicator node *is an event node which is annotated with the declaration of one or more parameters: an event replicator node is the root of a subtree representing several similar subtrees. If m parameters $v_1, \ldots, v_m$ are declared in a given event replicator, then it represents $|type(v_1)| \ \ldots \ |type(v_m)|$ events, one for each possible assignment of values to parameters. Without loss of generality we assume that a given parameter is declared in one and only one event replicator within a HLFT.*

Basic events, events and event replicators have an associated label that is used to uniquely identify them in the tree structure. This label can be parametric*: in this case it is expressed as a prefix followed by a list of parameters in brackets. A parameter v can be used in the label of some basic event, event or event replicator e only if v is declared in some event replicator on the path from the root of the HLFT to e (and in this case we say that the node corresponding to e is* in the scope[2] *of parameter v).*

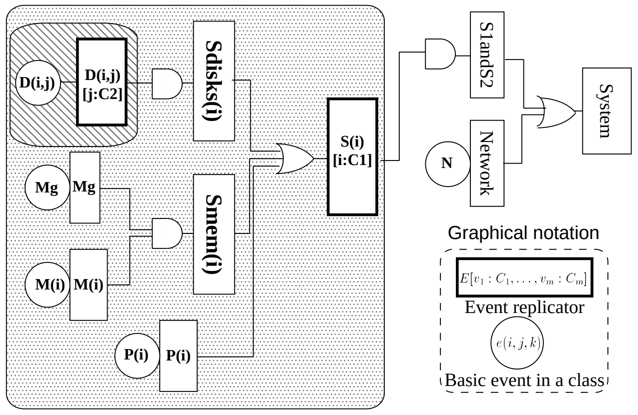

Figure 4. HLFT modeling the multiprocessor.

We define the set of free variables of a node n *of a HLFT, as the set FV_n of all parameters v such that n is in the scope of v.*

Logical gates in HLFT have the same notation and meaning as in FTs.

The root of a HLFT is an event, representing the TE. Each event in the tree has exactly one successor which can be either a basic event or a logical gate. In case the successor of an event is a basic event, the two nodes will have the same label and will be often considered as a unique indivisible basic event node *hereafter. The successor of an event replicator can be either a gate or a basic event. A logical gate has two or more successors, which are all events (possibly represented in a compact way by means of event replicator). The leaves of the tree are all basic events.*

In the HLFT of Figure 4 there are two event replicator nodes: $S(i)$ (subsystem i down), where parameter i is declared and $D(i, j)$, where parameter j is declared. The dotted area shows the scope of parameter i, while the dashed area shows the scope of parameter j.

HLFT unfolding. A HLFT can be *unfolded* into a FT by repeatedly applying a replication and substitution procedure to all subtrees whose root is an event replicator node. The unfolding algorithm first searches a subtree starting with an event replicator node *e* and such that it does not contain any other event replicator node; this subtree is replicated as many times as the possible assignments of values to the parameters declared in *e*. Each replica must be *appended* to the predecessor of *e* and is characterized by a specific assignment of values to the parameters, e.g. $v_1 = value_1, \ldots, v_m = value_m$. All basic event leaves of a given subtree replica that contain some parameter declared

[2]The event replicator e where a given parameter v is declared, is included in the scope of v.

in *e*, must be modified by substituting such parameters with the corresponding values characterizing the replica.

The above procedure must be iteratively repeated until no more event replicator nodes exist in the representation.

Applying the unfolding procedure to the HLFT of Figure 4, first the subtree within the dashed area is replicated twice, with the two replicas identified by the assignments $j = d_1$ and $j = d_2$. To complete this first step all variables j appearing in the leaves of the first (second) replica must be substituted with value d_1 (d_2). Then the procedure must be repeated for the subtree in the dotted area, that will be again duplicated, with the two replicas identified by the assignments $i = s_1$ and $i = s_2$. The result of the unfolding gives the FT of Figure 1, up to a relabeling adopted for the sake of labels readability ($M(s_1) \rightarrow M1$, $D(s_1, d_2) \rightarrow D12$, etc.).

Compact representation of minimal cut sets. The advantage of using HLFT representations is not only due to the more compact representation of the system structure, but also the possibility of performing a more efficient analysis by exploiting the symmetries made explicit by the HLFT representation.

In Section 5 we shall see how quantitative analysis can be made more efficient thanks to the high level representation. Here we discuss on our running example how qualitative analysis can take advantage of the system symmetries, that in our example are due to the presence of two similar subsystems, $S1$ and $S2$, and, within each subsystem, of two similar disks.

Equation 1 in Section 2 lists the ten MCS of the multiprocessor system, the same MCS have been obtained by using T-invariants in Section 3; however the ten MCS can be represented in a more compact way, reducing to seven. The difference is that in this case MCS can be parametric: (1) N, (2) $D11, D12, D21, D22$, (3) $M1, M2, Mg$, (4) $P1, P2$, (5) $P(i), M(k), Mg,\ \ i \neq k$, (6) $P(i), D1(k), D2(k),\ \ i \neq k$, (7) $Mi, Mg, D1(k), D2(k),\ \ i \neq k$. The last three MCS are parametric, and in fact they represent six ordinary MCS, that can be obtained by instantiating the parameters i and k (both of type C_1).

By allowing a parametric MCS representation, the list of MCS can reduce considerably, especially with highly redundant systems (if our system had three subsystems instead of two, the set of MCS would have 28 elements, while the set of parametric MCS would contain only 11 elements). The advantage, from the point of view of the user, is twofold: (1) a parametric representation allows to concentrate on the meaningful *patterns of system failure*, grouping all similar MCS into equivalence classes, (2) a shorter list of MCS can be understood and handled more easily. In our example the user is interested in knowing that one possible system breakdown cause is the simultaneous failure of the processor of one subsystem and of the two disks of the other

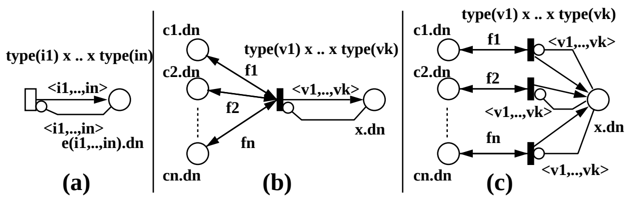

Figure 5. Translation rules.

subsystem, regardless the actual identity of the subsystem whose processor is down.

In the next section we shall describe a translation algorithm from HLFT to a high level Petri net formalism, namely Stochastic Well-Formed Nets (SWN): since parametric T-invariants have been defined for High level PNs [19, 18, 17], it is possible to apply the same structural analysis technique presented in Section 3 to the SWN models representing the HLFTs to obtain a parametric MCS representation.

4.1. Translating a HLFT into a SWN

In this section we present a new algorithm to generate SWN models from HLFTs, extending the algorithm in [5] (actually we extend the variant of the algorithm presented in Section 2). For space reasons we do not include the SWN definition here, the interested reader can find it in [16].

The definition of a SWN model starts with the definition of its *basic color classes*: the basic classes of a SWN model corresponding to a given HLFT are the types C_i of the HLFT parameters.

We now present the translation steps that allow to build the SWN model structure from the HLFT structure. Whenever a tuple of variables $\langle v_1, \ldots, v_m \rangle$ will appear as an arc labeling function in the SWN model, we assume, without loss of generality, that the variables appear in lexicographical order in the tuple.

The translation algorithm works in the same way as the one in [5], i.e., the HLFT is visited in postorder, and each node is then translated by applying the following rules.

Case of a basic event node
Rule BE1: If the event belongs to a class of basic events, and the class has not been translated yet, construct the subnet in Figure 5(a). Observe that the place has an associated *color domain* because the subnet represents in a compact way the possibility of a failure in *any* basic event in the class; the color domain, which defines the possible values that can be associated with the tokens contained in that place, corresponds to the set of all possible identifiers of the basic events in the class. Since the generic identifier of a basic event is described by a tuple of parameters $\langle i_1, \ldots, i_n \rangle$, then the colour domain of the place corresponding to the ba-

sic event class will be $type(i_1) \times \ldots \times type(i_n)$. The function on the transition output and inhibition arcs are simply identity functions.
Rule BE2: Basic event nodes that do not belong to a class are represented as in the FT to GSPN translation (Figure 2, bottom part of first rule).

Case of an *AND* gate
Rule AG1: The AND gate is translated as shown in Figure 5(b). The prefix x in the label of the subnet output place name, is the label identifying the event immediately preceding the AND node in the tree. The color domain of place, $x.dn$, is obtained from the set $FV_{AND} = v_1, \ldots, v_k$ of the free variables of the AND gate node, i.e., the color domain of $x.dn$ is $type(v_1) \times \ldots \times type(v_k)$. The function on $x.dn$ input and inhibition arcs is the identity function. There are as many input places $c_i.dn$ in the subnet as event nodes (of any kind) appended to the AND gate: let $c_1, \ldots, c_n$ be the labels of such nodes. Functions f_i associated with the input/output arcs of the subnet input places $c_i.dn$ are defined as follows:
Rule AGf1: if the node with label c_i is a basic event node, f_i is a tuple equal to the (possibly parametric) identifier of the event, labeling the node. If the basic event associated with the node does not belong to a class, then there is no need to associate a function with the arc (meaning that the arc has an associated constant function always resulting in one neutral token).
Rule AGf2: if the node with label c_i is an event, f_i is a tuple containing all the free variables of the event.
Rule AGf3: if the node with label c_i is an event replicator, and if $FV_{c_i} = v_1, \ldots, v_k$ are the free variables of the node, which shall include the set V_{c_i} of the parameters declared in it, then $f_i = \langle f_{v_1}, \ldots, f_{v_k} \rangle$ where $f_{v_i} = v_i$ if $v_i \in FV_{c_i} \backslash V_{c_i}$ else $f_{v_i} = S\ type(v_i)$ ($S\ C_i$ denotes the synchronization function on basic class C_i defined in the SWN arc expression syntax.)

Case of an *OR* gate
Rule OG1: The translation rule for the OR gate is shown in Figure 5(c). The output place color domain and arc functions are defined in the same way as for the AND gate. The input/output arcs of the subnet input places $c_i.dn$ have an associated function f_i which is exactly the same as in the AND gate for places representing failure of basic events or events, (i.e., Rules AGf1 and AGf2 are also used as Rules OGf1 and OGf2) while it changes for the event replicators.
Rule OGf3: In this case $f_i = \langle v_1, \ldots, v_k \rangle$, $v_i \in FV_{c_i}$.

Applying the above translation rules to the nodes of our running example (the translation order is determined by a postorder visit of the HLFT nodes) as summarized by the table in Figure 6 we obtain the SWN shown in Figure 7(a) . This model is constructed *gluing* the subnets in Figure 6 by superposing the places with same label.

Step	HLFT Node	Rules applied	Subnet
1	$D(i,j)$	BE1	(a)-left
2	$Sdisks(i)$	AG1,AGf3	(b)
3	Mg	BE2	(a)-right
4	$M(i)$	BE1	(a)-middle
5	$Smem(i)$	AG1,AGf1	(c)
6	$P(i)$	BE1	(a)-middle
7	$S(i)$ OR gate	OG1,OGf1,OGf2	(d)
8	*S1andS2*	AG1,AGf3	(e)
9	N	BE2	(a)-right
10	*System*	OG1,OGf1,OGf2	(f)

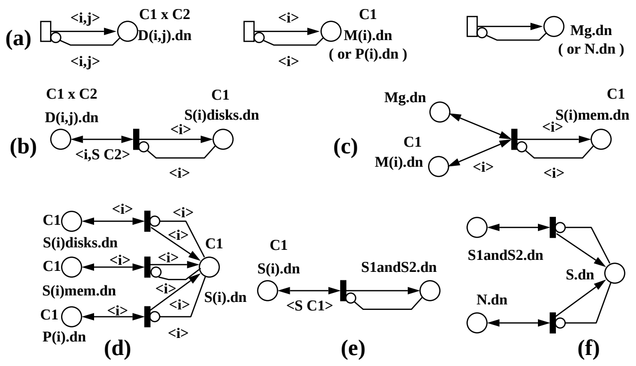

Figure 6. Summary of the example HLFT translation steps.

5. Fault tolerant multiprocessor analysis

In this section, we apply the proposed technique to the running example presented in Section 4.

For the sake of readability, we omit the drawing of the inhibitor arcs from place $S.dn$ to each transition, as discussed in [5]. Furthermore, we explicitly name only the places that model the state of components and the transitions representing a fault event. All the other places and transitions are left without names.

5.1. Model structural complexity

The SWN model obtained by applying the algorithm of Section 4 to the HLFT representing the fault tolerant multiprocessor is depicted in Figure 7(a). Two color classes named C_1 and C_2 have been defined; the former represents the subsystem objects while the latter represents the disk objects.

It can be easily noted that the SWN model is parametric in the number of subsystems and in the number of disks associated to each subsystem. The model has been obtained with the assumption of having exactly one shared memory and one single bus, but it can be easily modified to be parametric with respect to these two components as well. This observation leads to the first nice property of the proposed technique: the structural complexity of the resulting model, i.e., the number of places and transitions, does not depend on the number of replica inside each class. The number of

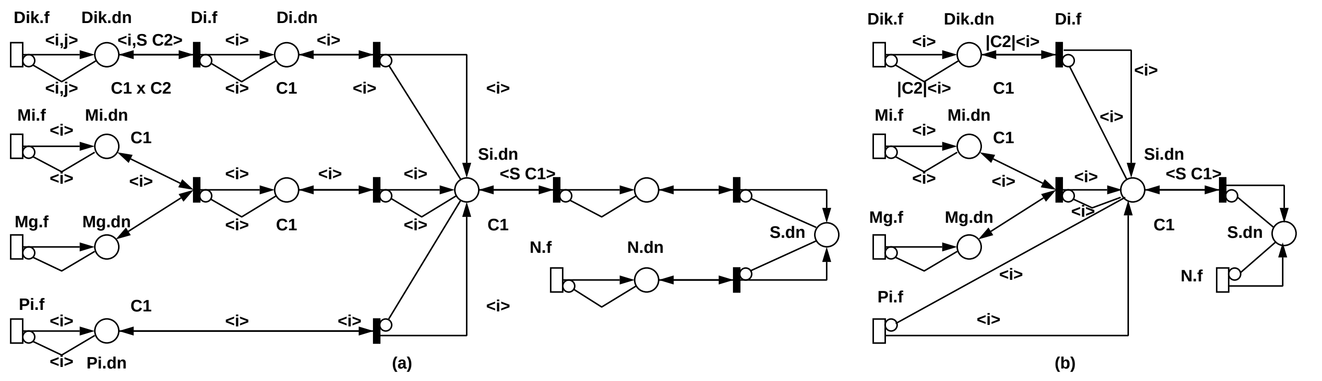

Figure 7. SWN model resulting from the application of the conversion algorithm of Section 4.

subsystems is reflected only in the cardinality of the corresponding color class. This is in sharp contrast with the conversion technique presented in [5] where the model complexity is highly dependent on the system size. The SWN model of Figure 7(a) comprises 10 places and 13 transitions. There are 6 places whose color domain is C_1 and one place whose color domain is $C_1 \times C_2$ therefore the corresponding GSPN model would have $|C_1| \cdot |C_2| + 6 \cdot |C_1| + 3$ places and $|C_1| \cdot |C_2| + 7 \cdot |C_1| + 5$ transitions (where $|C_i|$ is the cardinality of class C_i).

5.2. Model simplification

We can simplify the model of Figure 7(a) by realizing that it satisfies all the constraints imposed to perform the model decolorization technique as proposed in [13]. Roughly speaking, decolorization of the SWN model means stripping away all the informations about redundant color classes. In this case, color class C_2 that represents the disks used by each processor is redundant since the relevant information about this type of entities is their number. In this case, the decolorization requires a redefinition of the rate of transition $Dik.f$ that must be dependent on the marking of place $Dik.dn$ [15].

A further simplification is obtained by a structural reduction based on the elimination of "useless" pairs of places and transitions whose purpose is only to "pass forward" the tokens without affecting the global model behavior. The resulting model is depicted in Figure 7(b).

A similar simplification can be adopted also for the T-invariant computation mentioned in Section 3. Indeed, a simplification of the GSPN results into a simplification of its incidence matrix thus reducing the effort for computing the T-invariants.

5.3. Including dependencies: imperfect coverage

The main reason for converting a combinatorial model like a FT into a GSPN based model, is to exploit the larger modeling power of the last. A typical situation that forces to include dependencies into the model is the presence of an imperfect *coverage*. By this we mean that the failure of the i-th redundant subsystem may cause the failure of the entire system with probability $P_{coverage}$ (> 0).

In order to model an imperfect coverage at the subsystem level, one place and two immediate transitions (with priority higher than the priority of all the other immediate transitions) must be added. These transitions are named $Pcov$ and $Pnocov$ and are enabled whenever a new token is put in place $Si.dn$. Their weights are $P_{coverage}$ and $1 - P_{coverage}$ respectively. When $Pcov$ fires a token is deposited in place $S.dn$ thus modeling the occurrence of the TE. On the contrary, the firing of the conflicting transition $Pnocov$ models the situation where the fault of the i^{th} replica does not affect the failure of the entire system (unless the i^{th} was the last working component). The resulting SWN model is not shown due to space constraints.

5.4. Results

The number of aggregated states generated by the SWN models, that are used in the computation of the state probabilities, is extraordinarily smaller than the number of states generated by the corresponding GSPN model obtained by unfolding the SWN model. This induces drastic reductions of the computational complexity of the solution, and allows a much wider set of system configurations to be studied.

As an example, Table 1 compares the number of states for the SWN model of Figure 7(b) with the number of states of the corresponding unfolded GSPN model. The bottom part of Table 1 compares the number of states for the SWN model with imperfect coverage with the number of states of the corresponding GSPN. The table considers multiproces-

Table 1. Number of states of the SWN model and of the GSPN model of the multiprocessor (with and without coverage) for a variable number of subsystems, with two disks per processor.

	Tangible states		Vanishing states		Absorbing states	
N	SWN	GSPN	SWN	GSPN	SWN	GSPN
2	49	92	88	163	90	163
4	793	11,504	1,196	18,543	1,044	14,095
8	28,171	113,417,984	46,760	263,359,231	30,744	116,777,251
Data for the imperfect coverage SWN model						
2	49	92	192	371	94	171
4	793	11,504	3,316	56,687	1,225	17,199
8	28,171	113,417,984	148,576	953,690,879	43,471	184,902,399

sor systems with an increasing number of subsystems (Column 1) with a single bus, a single shared memory and two disks per processor.

It is quite evident that the degree of aggregation (that we define as the ratio between the number of GSPN states and the number of aggregated SWN states) is very high: for the data presented in Table 1, it ranges from 1.87 (for $N = 2$) to about 4026 (for $N = 8$) for the tangible states, from 1.85 to about 5632 for the vanishing states, and from 1.81 to about 3798 for the absorbing states. Similar numbers derive from the data of the imperfect coverage. Table 1 shows another nice property of the proposed conversion technique: the state space is automatically aggregated and its compactness allows the investigation of systems of a larger number of elements. Given the numbers shown in Table 1, an approach based on the conversion of FT into GSPN models would not allow the investigation of systems with more than five subsystems due the exceedingly large number of states generated by the resulting model.

As an example of the numerical results that can be obtained from the SWN models, we have evaluated the system unreliability as a function of time for different configurations. The time domain computations have been performed by resorting to a randomization technique for the transient solution of the Markov chain, associated with the SWN model. The system unreliability, i.e. the probability of occurrence of the TE is defined, in SWN models, as $P\{M(S.dn) = 1\}$.

For all the components, the failure distribution is assumed to be exponential with the following failure rates (in h^{-1}): processor = $5 * 10^{-7}$, disk = $8 * 10^{-5}$, local memory = $3 * 10^{-8}$, shared memory = $3 * 10^{-8}$, bus = $2 * 10^{-9}$. Figure 8 plots the unreliability of the multiprocessor system with 7 subsystems and two or three disks per subsystem. In the same figure this measure is plotted when an imperfect coverage probability $P_{coverage} = 0.01$ is included at the subsystem level.

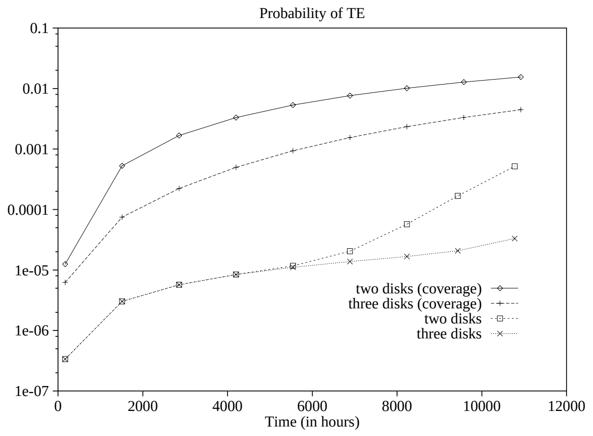

Figure 8. Probability of system fault as a function of time with and without coverage.

The observation of the numerical results leads to remark that increasing the number of disks per processor, that are the components with the highest failure rate, has a beneficial impact on the system reliability, especially for increasing mission times.

All numerical results were obtained with the *GreatSPN* package [14] running on a PC Pentium with 64 MBytes of main memory and running Solaris operating system. The computational cost for the derivation of the numerical results is rather small: each curve is composed of 9 points and the average CPU time needed for the computation of each point ranges from the minimum of 20 seconds (for the model with two disks) to a maximum of 300 seconds (for the model with three disks per processor and the imperfect coverage).

6. Conclusions and future works

Starting from the idea of converting a FT into a GSPN, this paper has reached several new achievements: *i)* the qualitative analysis of the system (i.e. MCS determination) through structural analysis (i.e. T-invariants) of the GSPN model; *ii)* the definition of a more compact FT representation called *High Level FT (HLFT)*; *iii)* automatic conversion of a HLFT into a SWN.

This last point has several consequences:

- converting a combinatorial model into a state-space based model increases the modeling and analytical power (at the cost of more expensive computations);
- current research in the area of FT is to find ways to include dependencies (see for instance [11, 12]); the methodology proposed in this paper offers a cleaner way to achieve the goal;
- the direct mapping of a HLFT into a SWN greatly alleviates the state explosion problem.

To prove some of the above assertions, we have carried on an example of a multiprocessor system. The computational saving obtained by using a SWN model instead of the corresponding unfolded GSPN model has been extensively examined. Furthermore, we proposed some structural reductions of the SWN model to further optimize the analysis step. Finally, we have introduced into the model some statistical dependencies due to the inclusion of an imperfect coverage.

Possible extensions to the present work are sketched in the following:

- failure-time distributions, either with mass at time 0 or at time ∞ can be included, as in [5].
- Sources of dependency (other than coverage) can be easily modeled, as for instance different repair policies, or dynamic dependencies like in [11] or adaptive components as in [12].
- Firing rates can be marking dependent.
- In the paper, SWN models are used for quantitative analysis, only: we are now working on the extension of the qualitative analysis based on T-invariants to HLFT SWN models.

References

[1] E.J. Henley, H. Kumamoto. *Reliability Engineering and Risk Assessment* Prentice Hall, 1981.

[2] N.G. Leveson. *Safeware: System Safety and Computers* Addison-Wesley, 1995.

[3] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. Wiley Series in Parallel Computing, 1995.

[4] G.S. Hura, J.W. Atwood. The Use of Petri Nets to Analyze Coherent Fault Trees. *IEEE Transactions on Reliability*, R-37:469–474, 1988.

[5] M. Malhotra and K. Trivedi. Dependability modeling using Petri nets. *IEEE Transactions on Reliability*, R-44:428–440, 1995.

[6] R. Sahner, K.S. Trivedi, and A. Puliafito. *Performance and Reliability Analysis of Computer Systems: An Example-based Approach Using the SHARPE Software Package*. Kluwer Academic Publisher, 1996.

[7] G. Peterka and T. Murata. Proof procedure and answer extraction in Petri net model of logic programs. *IEEE Transactions on Software Engineering*, 15(2):209–217, 1989.

[8] L. Portinale. Exploiting T-invariant analysis in diagnostic reasoning on a Petri net model. In *Proc. 14th Int. Conf. on Application and Theory of Petri Nets, LNCS 691*, pp. 339–356. Springer Verlag, Chicago, 1993.

[9] D. Zhang and T. Murata. Fixpoint semantics for Petri net model of definite clause logic programs. Technical Report UIC-EECS-87-2, University of Illinois at Chicago, 1987.

[10] S.A. Doyle, J. Bechta Dugan, A. Patterson-Hine. A combinatorial approach to modeling imperfect coverage. *IEEE Transactions on Reliability*, R-44:87–94, 1995.

[11] R. Manian, D.W. Coppit, K.J. Sullivan, J.B. Dugan. Bridging the gap between systems and dynamic fault tree models. In *Proc. of IEEE Annual Reliability and Maintainability Symposium*, pp. 105-111, 1999.

[12] G. Szabó, P. Gáspár. Practical treatment methods for adaptive components in the fault tree analysis. In *Proc. of IEEE Annual Reliability and Maintainability Symposium*, pp. 97-104, 1999.

[13] G. Chiola and G. Franceschinis. A structural colour simplification in Well-Formed coloured nets. In *Proc. 4th Intern. Workshop on Petri Nets and Performance Models*, pp. 144–153, Melbourne, Australia, December 1991. IEEE-CS Press.

[14] G.Chiola, G.Franceschinis, R.Gaeta, M.Ribaudo, "GreatSPN 1.7: GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets", *Performance Evaluation*, Vol. 24, n. 1,2, November 1995, pp. 47-68.

[15] M. Ajmone Marsan, S. Donatelli, G. Franceschinis, F. Neri, *Reductions in Generalized Stochastic Petri Nets and Stochastic Well-formed Nets: An Overview and an Example of Application* Network Performance Modeling and Simulation, J. Walrand, K. Bagchi and G. Zobrist (editors), 1997.

[16] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed coloured nets for symmetric modelling applications. *IEEE Transactions on Computers*, 42(11), November 1993.

[17] J.M. Couvreur. The general computation of flows for coloured nets. In *Proc.* 11^{th} *Intern. Conference on Application and Theory of Petri Nets*, Paris, France, June 1990.

[18] J.M. Couvreur, S. Haddad, and J.F. Peyre. Resolution parametree d'une famille de systemes d'equations lineaires a solutions positives. Technical Report 90–38, Université Paris 6, 4 Place Jussieu, 75252 Paris Cedex 05, France, September 1990. IBP Tech. Report (in French).

[19] S. Haddad and J.M. Couvreur. Towards a general and powerful computation of flows for parametrized coloured nets. In *Proc.* 9^{th} *Europ. Workshop on Application and Theory of Petri Nets*, Venezia, Italy, June 1988.