Università degli Studi di Torino

Dipartimento di Informatica
C.so Svizzera 185, 10149 Torino, Italy

DOTTORATO DI RICERCA IN INFORMATICA
Ciclo XVIII

Extended Fault Trees Analysis supported by Stochastic Petri Nets

Ph. D. thesis

by **Daniele Codetta-Raiteri**
codetta@di.unito.it

ADVISOR:
Prof. Andrea Bobbio

Ph. D. COORDINATOR:
Prof. Pietro Torasso

November 15^{th}, 2005

Contents

Chapter 1

Introduction

This work presents several extensions to the *Fault Tree* [90] formalism used to build models oriented to the *Dependability* [103] analysis of systems. In this way, we increment the modelling capacity of Fault Trees which turn from simple combinatorial models to an high level language to represent more complicated aspects of the behaviour and of the failure mode of systems. Together with the extensions to the Fault Tree formalism, this work proposes solution methods for extended Fault Trees in order to cope with the new modelling facilities. These methods are mainly based on the use of Stochastic Petri Nets.

Some of the formalisms described in this work are already present in the literature; for them we propose alternative solution methods with respect to the existing ones. Other formalisms are instead part of the original contribution of this work.

In this chapter, we present the state of art on Fault Trees and their extensions (section 1.2), and we describe what is the original contribution of this thesis (section 1.3). Moreover, the aim of this chapter is also introducing some notions about Dependability (section 1.1.1), justifying the use of models for the Dependability analysis (section 1.1.2), and presenting some measures to quantify the Dependability level of a system (section 1.1.3).

1.1 Concepts of Dependability

1.1.1 Definition of Dependability

We talk about safety critical systems when we deal with systems whose incorrect behaviour may cause undesirable consequences to the system itself, to the operators, to the population or to the environment. This definition fits categories of systems such as industrial production plants, electric power plants, and transportation systems. In these cases, *Dependability* is a crucial point in the design of the systems. Dependability is the property of a system to be dependable in time: we can say that the Dependability level of a system is as high as we are confident that the system will provide correctly its service during its life cycle. The incorrect

behaviour of a system may be caused by faults, failures or errors involving its components; Dependability requirements are part of the design specifications of safety critical systems.

Dependability does not concern only safety, risks and design specifications of the system, but it influences other aspects as well, such as the technical assistance and maintenance of the system, and the market competition. The technical assistance can be planned in terms of time, cost and logistic, according to the Dependability evaluation of the system.

For instance, the decision of the warranty period of a technological item is linked to the Dependability of the item; moreover, the repair or replacement of failed components by a maintenance crew, is an aspect to take in account when forecasting the cost of maintenance of the system; this cost is the sum of the cost of spare components, of the cost of the personnel dedicated to the system repair, and of the economic loss due to the production suspension during the repair time. In order to minimize the maintenance cost, two maintenance policies are possible: the proactive maintenance and the reactive maintenance; in the first case, the maintenance action tries to prevent the failure of the components or of the whole system; in the second case, the maintenance action is triggered by the failure of the system.

Besides these two policies, the Dependability of a system can be improved by means of fault forecasting and fault tolerance; a system providing a service is *fault tolerant* [56, 101, 102] when the system is characterized by the capacity of assuring the service although a failure has involved a part of the system. Fault tolerance is typically achieved by replicating the critical components in the system.

Finally, Dependability may influence customers' choices: advertisement messages stress the Dependability and the image of a brand may depend on the Dependability of its products or services.

Recently, we assisted at the wide diffusion of computing and information technologies in several industrial and economic areas; when computing or networking systems are adopted to support activities with associated relevant risks, the concept of Dependability becomes relevant also for this class of systems [91, 92, 100, 105].

1.1.2 Dependability evaluation

There are two main methods of evaluation of the Dependability: the *Measurement-based* method and the *Model-based* method. The first method requires the observation of the behaviour in the operational environment, of the physical objects composing the system. In this way, Dependability measurements are obtained and concern objects such as component prototypes or effective components of the system; in the second case, the component may be evaluated by means of accelerated life tests.

The measurement-based method is the most believable, but it may be unpractical or too expensive; in these cases, the model-based method is preferable and consists of the construction of a model representing the behaviour of the system in

terms of primitives defined in the formalism associated with the model. The model of the system must be a convenient abstraction of the system; this means that a model may not completely capture the behaviour of the system, but the level of accuracy of the model must be enough high to represent correctly the aspects of the system behaviour which are of interest for the Dependability evaluation of the system. The degree of accuracy of a model depends on the capacity of the associated formalism to extrapolate the system features.

With respect to the measurement-based method, the model-based method is less believable, but less expensive. Models can be the object of analysis or simulation, and can be mainly classified as combinatorial models and state space based models. The models in the first category represent the structure of the system in terms of logical connection of working (failed) components in order to obtain the system success (failure). State space based models instead, represent the behaviour of the system in terms of reachable states and possible state transitions.

Combinatorial models have an intuitive notation, they are easy to be designed and manipulated, and they can be efficiently analyzed by means of combinatorial methods. Despite of these advantages, combinatorial models suffer from a very limited modelling power, mainly due to the assumption of the statistical independence of the events. Examples of combinatorial models are *Reliability Block Diagrams* (RBD) [83, 108], *Event Trees* [33, 110] and *Fault Trees* (FT) [90].

When the accuracy of combinatorial models is not enough to capture the characteristics of the system to be modelled, we can resort to state space based models; the models is this category have a greater modelling power with respect to the combinatorial models, but the state space analysis may be computationally expensive. This depends on the number of states in the model; however, the state space size may grow exponentially with respect to the number of components in the system. When the analysis of state space based models become unpractical due to the high number of states, Dependability measures can be obtained from these models by means of simulation. In general, state space based models are addressed to expert model designer. Examples of models in this category are the *Markov Chains* [83, 100] and the *Stochastic Petri Nets* [1, 69, 89].

1.1.3 Measures of Dependability

The concept of Dependability is quite general; in order to evaluate the Dependability of an item, we need some measures to characterize numerically the Dependability. The mechanisms that lead to failure a technological object are very complex and depend on many factors, such as physical, technical, human and environmental factors. These factors may not obey to deterministic laws, so we can consider the time to failure of an item as a random variable. For this reason, the Dependability evaluation in quantitative terms, is based on the probabilistic approach, and consists of the computation of several numerical measures characterizing the Dependability.

One of these measures is called *Reliability* [55, 63, 67, 81, 93, 99] and is indi-

cated by $R(t)$: the Reliability of an item (component or system) at time t ($R(t)$), is the probability that the item performs the required function in the interval $(0, t)$ given the stress and environmental conditions in which it operates. The *Unreliability* of an item at time t, is the probability that the item is already failed at time t.

Repairable systems are characterized by the alternated up and down states, due to the alternated occurrences of failures and repairs of the systems. In the case of repairable systems [57], we talk about *Availability* instead of Reliability: the Availability $A(t)$ of an item at time t, is the probability that the item is correctly working at time t. The *Unavailability* of an item at time t is the probability that the item is not performing the required function at time t. The Unavailability at time t of an item is the complement of the Availability at the same time: $1 - A(t)$.

The probabilistic approach

Let X be the random variable representing the time to failure of a non repairable item. The *cumulative distribution function* (cdf) of X is indicated by $F(t)$ and provides the Unreliability of the item at time t:

$$F(t) = Pr\{X \leq t\} \tag{1.1}$$

The following properties hold for $F(t)$:

- $F(0) = 0$
- $lim_{t \to +\infty} F(t) = 1$
- $F(t)$ is non decreasing.

The Reliability of the item at time t is given by the *survivor function*:

$$R(t) = Pr\{X > t\} = 1 - F(t) \tag{1.2}$$

The following properties hold for $R(t)$:

- $R(0) = 1$
- $lim_{t \to +\infty} R(t) = 0$
- $R(t)$ is non increasing.

Given a derivable cdf $F(t)$, the density function $f(t)$ of X, is defined as

$$f(t) = \frac{dF(t)}{dt} \tag{1.3}$$

$$f(t)dt = Pr\{t \leq X < t + dt\}$$

The *Mean Time To Failure* (MTTF) of the item is given by

$$MTTF = E[X] = \int_0^{+\infty} tf(t)dt = \int_0^{+\infty} R(t)dt \tag{1.4}$$

$h(t)$ is the *hazard (failure)* rate of an item:

$$h(t) = \frac{f(t)}{R(t)} = \frac{f(t)}{1 - F(t)} \tag{1.5}$$

$h(t)\Delta t$ is the conditional probability that the item will fail in the interval $(t, t+\Delta t)$ given that it is functioning at time t.
$f(t)\Delta t$ is the unconditional probability that the unit will fail in the interval $(t, t + \Delta t)$.

Fig. 1.1 shows the typical bathtub shape of the $h(t)$ function curve. The life cycle of the item consists of the sequence of the following phases:

- *Decreasing Failure Rate* (DFR) phase - In this phase, the failure rate decreases with time. This phase is due to undetected defects of the item.

- *Constant Failure Rate* (CFR) phase - In this phase, the failure rate is *age independent*; this means that the failure rate remains constant in time. Moreover, the failure rate value is much lower than in the early-life period. In this phase, the failure is caused by random effects. The CFR phase is the useful life period of the item.

- *Increasing Failure Rate* (IFR) phase - In this phase, the failure rate increases with age. This phase is due to deterioration (wear-out) of the item.

The last phase does not concern all the classes of item; for instance, mechanical components can be object of deterioration, while the IFR phase is not present in the life cycle of several electronic components.

If the random variable X representing the time to failure of an item, is ruled by the *negative exponential distribution*, the failure rate is age independent along the complete life cycle of the item; in other words, the DFR phase and IFR phase are not present in the life cycle of the item. This distribution has only one parameter indicated by λ; the cdf, the Reliability function and the density function according to this distribution are reported in Tab. 1.1. Fig. 1.2 and Fig. 1.3 show the Reliability function curve and the density function curve, respectively, for $\lambda = 1$.

We suppose that an item has been operating (has not failed) until time t_1; the remaining (residual) lifetime of the item is given by $Y = X - t_1$. The negative exponential distribution is characterized by the *memoryless* property: the distribution of Y does not depend on t_1. This means that the distribution of the residual lifetime of the item, does not depend on how long the item has been operating. If the distribution of Y is indicated by $G_{t_1}(t)$, the memoryless property can be proved

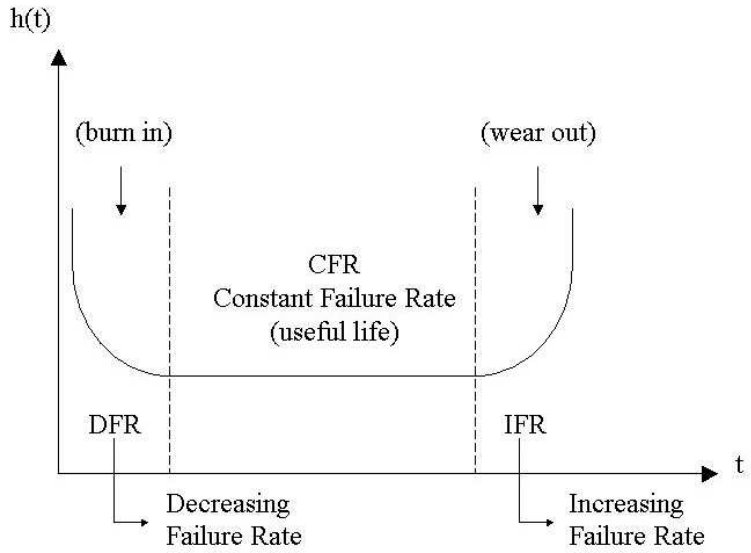

Figure 1.1: The bathtub shape of the hazard (failure) rate function ($h(t)$) curve.

cdf	$F(t) = 1 - e^{-\lambda t}$
Reliability	$R(t) = e^{-\lambda t}$
Density function	$f(t) = \lambda e^{-\lambda t}$
Failure rate	$h(t) = f(t)/R(t) = \lambda$
Mean Time To Failure	$MTTF = 1/\lambda$

Table 1.1: The negative exponential distribution.

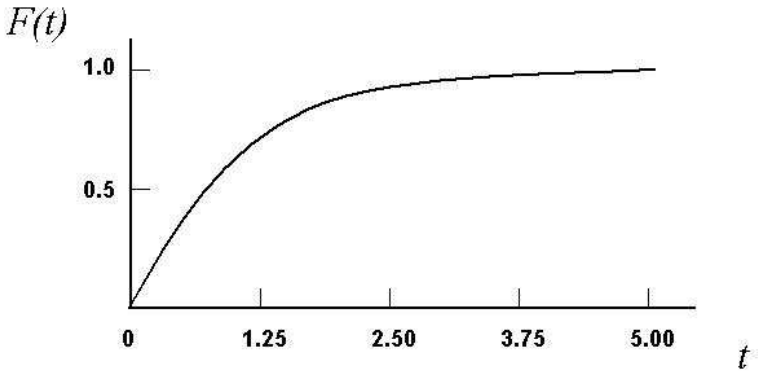

Figure 1.2: The Reliability function curve for $\lambda = 1$.

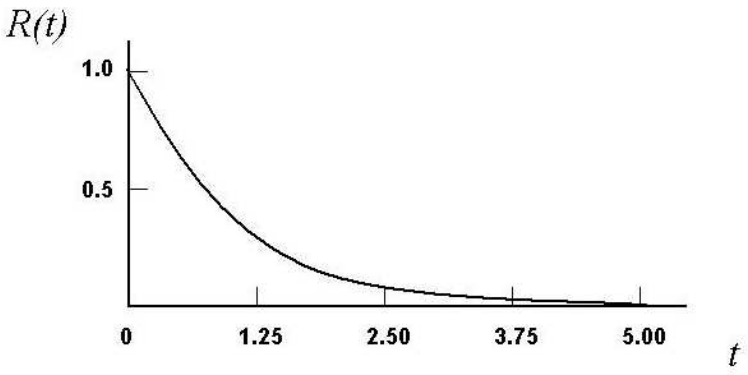

Figure 1.3: The density function curve for $\lambda = 1$.

in this way:

$$\begin{aligned}
G_{t_1}(t) &= Pr\{Y \leq t | X > t_1\} = \\
&= Pr\{X \leq t_1 + t | X > t_1\} = \\
&= \frac{Pr\{t_1 < X \leq t_1 + t\}}{Pr\{X > t_1\}} = \\
&= \frac{F(t_1 + t) - F(t_1)}{1 - F(t_1)} = \\
&= \frac{1 - e^{-\lambda(t_1+t)} - (1 - e^{-\lambda t_1})}{1 - (1 - e^{-\lambda t_1})} = \\
&= \frac{e^{-\lambda t_1} - e^{-\lambda(t_1+t)}}{e^{-\lambda t_1}} = \\
&= \frac{e^{-\lambda t_1} - e^{-\lambda t_1} \cdot e^{-\lambda t}}{e^{-\lambda t_1}} = \\
&= \frac{e^{-\lambda t_1}(1 - e^{-\lambda t})}{e^{-\lambda t_1}} = \\
&= 1 - e^{-\lambda t} = F(t) \qquad (1.6)
\end{aligned}$$

Thus $G_{t_1}(t)$ is independent of t_1 and is identical to the original exponential distribution of X ($F(t)$). The failure of the item is not due to its gradual deterioration, but it is some suddenly appearing failure.

1.2 State of the art on Fault Trees

This thesis presents several extensions to the *Fault Tree* (FT) formalism, with the aim of improving its modelling power and consequently to increase its capacity of capturing the behaviour of the system to be modelled. Due to the introduction of new modelling facilities, new methods for the analysis of extended FTs, become

necessary. In this section, we briefly present the characteristics and the solution techniques for several versions of the FT formalism, present in the literature.

1.2.1 Standard Fault Trees

The FT model was born as a combinatorial model oriented to the Dependability analysis of systems (section 1.1.1). A FT is a *direct acyclic graph* (DAG) representing how several combinations of *basic events* lead to the occurrence of a particular event called *top event*.

Besides being a combinatorial model (section 1.1.2), the FT is a stochastic model; in this sense, the basic events of the FT are stochastic events and they occur after a period of time which is a random variable. On a FT model, we can compute several measures; in particular, it is possible to compute the probability of the top event to be occurred at a certain time. If the basic events consist of the failure events of the system components, each basic event is ruled by some probability distribution, and the top event corresponds to the failure of the whole system, then the FT allows the computation of the system Unreliability (section 1.1.3) at a certain time, given by the probability of the top event to be occurred at that time.

In this thesis, we consider the FT as a model oriented to the Unreliability computation, and we assume that the basic events (component failures) are ruled by the negative exponential distribution (section 1.1.3) whose parameter λ is the failure rate of the component. However other probability distributions can be assigned to the basic events of a FT.

Besides the computation of the system Unreliability on a FT model (quantitative analysis), a FT can be the object of the qualitative analysis; this means computing the *Minimal Cut Sets* (MCS) (section 2.3.1) of the system. A MCS is a minimal set of components whose contemporary state of failure determines the failure of the whole system.

In the original version of the FT formalism, basic events are assumed to be independent and the combinations of basic events leading to the top event, can only be expressed by means of *Boolean gates* (or logic ports) representing the Boolean operators $\vee$ and $\wedge$. Standard FTs can be efficiently analyzed, but their modelling power is strongly limited by the assumptions mentioned above. A way to perform both the qualitative and quantitative analysis of FTs, consists of resorting to *Binary Decision Diagrams* (BDD) [16, 18, 79, 94].

Several extensions to the FT formalism have been proposed in the literature, with the purpose of increasing the modelling power of FTs. The introduction of new primitives in the formalism, determined the necessity of new analysis methods for the extended FTs.

We concentrate our attention on two extensions to the FT formalism: the *Parametric Fault Tree* (PFT) [11, 51] formalism and the Dynamic Fault Tree (DFT) [39, 40, 70, 71] formalism.

1.2.2 Parametric Fault Trees

One of the common ways to improve the dependability of a system, consists of replicating a unit or a subsystem providing a certain service; for instance, in a client-server computing system, several servers for the same service might be available in order to avoid that the failure of a single server unavails the service.

Modelling a system with replicated parts as a FT, would lead to the presence of several identical "subtrees"; if their size is too large, the model design would not be very practical; in order to give a more compact modeling of the system, the PFT has been proposed; using PFT, identical subtrees are folded to a single parametric subtree; in this way, only one representative of the several replicas is present, while the identity of each replica is maintained through the values that the parameters can assume; parameterization can be applied several times in the same PFT and at several depths.

Two solution methods were proposed for PFT models in [11]: the first one consists of unfolding the PFT model, in other words deriving from the PFT the corresponding FT. The second method consists of converting the PFT model in the equivalent High-level Stochastic Petri Net using the *Stochastic Well-formed Coloured Petri Net* (SWN) [25] formalism.

The unfolding technique reduces the PFT formalism to be only a formal notation to compactly represent the redundancies in the system, while the analysis is still performed on the unfolded FT, hence losing the possibility of exploiting in the analysis phase the regular structure of the model: several subtrees in the unfolded FT are similar and the analysis could be performed on only one representative subtree per equivalence class.

The use of SWN was motivated by the possibility of analyzing systems with some kind of dependency (not allowed in standard FT analysis): in this case however, the model state space must be generated, which could be very expensive from a computational point of view. The SWN analysis techniques allow to automatically exploit the system symmetries and group states into macro states (state aggregates), however the cost might still be high. In [11] it was proposed to use the SWN structural analysis technique based on the computation of minimal T-semiflows to obtain the MCSs. This technique however cannot be applied in general since the problem of computing a generating family of minimal, parametric T-semiflows of SWN models is still an open problem.

1.2.3 Dynamic Fault Trees

In the DFT formalism, several new gates, called dynamic gates, have been introduced in order to model dependencies among failure events or component states; dynamic gates introduce temporal dependency, functional dependency and the presence of spare components with a failure rate varying with the spare state: dormant or working. Due to the presence of dependencies, DFT models are state space based models, instead of combinatorial models.

The DFT formalism was introduced by J. B. Dugan et al. The dynamic gates were defined in [40]; the algorithm to derive from a DFT model, its equivalent state space in form of *Continuous Time Markov Chain* (CTMC) [83, 100], was proposed in [70].

The state space analysis may have very high computational costs, or even be unfeasible. For this reason, a modular approach to analyze DFTs, was proposed in [61, 71]; the modularization technique to analyze FTs (section 2.5.3) has been extended to deal with DFTs. Modules are classified as static or dynamic: static modules are independent subtree containing only Boolean gates; dynamic modules contain dynamic gates. Each module is analyzed with the proper technique: a static module is converted to a BDD; a dynamic module is converted to a CTMC. So, the state space analysis is limited to dynamic gates, while static modules are solved by means of less computational expensive BDDs (combinatorial solution).

First, the way to perform the quantitative analysis of DFTs, was studied. More recently some methods were proposed to perform the qualitative analysis of DFTs [98], to compute importance measures (section 2.3.3) on DFTs [76] (sensitivity analysis), and to derive diagnostic systems from DFTs [4, 5].

The qualitative analysis of DFTs returns the MCSs and the Minimal Cut Sequences of the system. As a MCS is a minimal set of basic events leading to the occurrence of the TE, a Minimal Cut Sequence is an ordered sequence of BEs leading to the occurrence of the top event. In other words, the top event is caused by a Minimal Cut Sequence if both all its basic events occur, and they occur in a specific order. Minimal Cut Sequences are justified by the presence in the DFT of dynamic gates requiring or forcing specific temporal orders among some events in the DFT model (see section 4.2).

A software tool called *Galileo* [42, 97] for the quantitative analysis of DFTs, was recently extended to perform also the sensitivity and qualitative analysis of DFTs.

Besides the development of the DFT formalism by Dugan et al., some attempts to combine FTs with models of other nature (mainly state space models) were proposed in the literature: in [41, 14] FTs are combined with CTMCs; in [19, 12, 20, 64] FTs are combined with Petri Nets. FTs with multistate components are proposed in [66, 108].

1.3 Original contribution and motivations

Contribution to Parametric Fault Trees. We first concentrated on PFTs trying to provide a way to analyze this kind of models avoiding the drawbacks characterizing the currently available techniques described in section 1.2.2. At the same time, our intent was exploiting the parametric form of the PFT models, not only in the model construction phase, but also in the analysis phase. As the parametric form allows to reduce the model size, our aim is reducing the number of steps necessary to perform the model analysis by exploiting the parametric form.

To this aim, we extended BDDs used to analyze in an efficient way FTs, in order to cope with PFTs maintaining the parametric form: we obtained *Parametric Binary Decision Diagrams* (pBDD) [10]. We succeeded in generating the pBDD equivalent to a PFT, and in performing both the qualitative and the quantitative analysis on the pBDD.

Using pBDDs to analyze PFTs, we avoid the unfolding of the model. Moreover, we extended the definition of module (independent subtree) in the case of PFTs. As in the case of FTs, the detection of PFT modules allows to decompose the PFT model to further reduce the complexity of its analysis.

Contribution to Dynamic Fault Trees. DFTs or their dynamic module need the state space analysis; this means generating and analyzing the corresponding CTMC. A DFT model may represent complex failure modes characterized by several kinds of dependency among the events, and modelled by combinations of several dynamic gates. Thus, the direct generation of the CTMC from a DFT is a complicated task and requires a complex algorithm [70].

At the same time, efficient ways to generate the CTMC from a *Generalized Stochastic Petri Net* (GSPN) [1] are available. GSPNs allows to represent the failure mode modelled by a DFT; generating the GSPN equivalent to a DFT is less complicated then generating the CTMC equivalent to the DFT, especially when the DFT represents complex failure modes with a lot of dependencies (dynamic gates).

So, we formalized the way to perform the conversion of a DFT model into a GSPN, in form of *model-to-model transformation* [36] based on *graph transformations* [50]. The conversion of a DFT into GSPN is straightforward even though several combinations of dynamic gates are present. After the conversion, the CTMC can be derived from the GSPN with the available techniques [1].

The conversion from DFT to GSPN can concern the whole DFT or its dynamic modules; the DFT analysis exploiting modules has been considered taking into account the possibility of mapping dynamic modules to GSPNs and performing their analysis in this form.

Contribution to Repairable Fault Trees. In this thesis, we present a new extension of the FT formalism, called *Repairable Fault Tree* (RFT) [32]. While the previous formalisms, such as FT, PFT and DFT, allowed to represent only the failure mode of the system, the RFT formalism allows to represent the presence of repair processes as well. Besides events and gates, RFT models contain a new primitive called *Repair Box* to represent the repair of a set of components.

As DFTs, RFT models require the state space analysis, since repair processes establish some kind of dependency among the events in the model. The RFT analysis is realized by means of modularization and conversion of modules into GSPNs.

Contribution to extended FT formalisms integration. Finally, we integrated the PFT, DFT and RFT formalism, in order to build models including all the mod-

elling facilities introduced in each formalism: the parametric form, dynamic gates, repair boxes. The resulting formalism is called *Dynamic Repairable Parametric Fault Tree* (DRPFT) [8]. The integration of such facilities required to review the semantic of dynamic gates in order to be combined with the parametric form and repair boxes.

The analysis of DRPFT is still performed exploiting modules; the state space analysis is performed by mapping modules to SWN. SWNs allow to maintain the parametric form, and to represent the dependencies due to dynamic gates and repair boxes. Moreover, a lumped CTMC can be derived from a SWN, reducing the state space size and consequently the computational costs of the state space analysis.

A set of graph transformation rules is provided in this thesis, to implement the model-to-model transformation from DRPFT to SWN.

1.4 Structure of the thesis

Chapter 2 describes standard FTs focusing on the qualitative and quantitative results computable on FT models through the BDD based FT analysis. The definition of FT module and the way to analyze FTs exploiting modules, is also described.

In chapter 3, the PFT formalism is defined; then, pBDDs are presented together with the way to generate the pBDD equivalent to a PFT model. The method to perform the PFT qualitative and quantitative analysis by means of pBDDs, is introduced. Moreover, the concept of module is redefined for PFTs.

The way to convert DFT models into GSPNs is proposed in chapter 4, where the DFT formalism and some concepts of model-to-model transformations and graph transformations, are also described. In the same chapter, we deal also with the DFT modularization. The way to detect and classify DFT modules is discussed.

The RFT formalism is presented in chapter 5; the repair box semantic is introduced together with some repair policies. The way to convert a RFT model into GSPN is described together with the way to perform the RFT analysis by modularization.

In chapter 6 the PFT, DFT and RFT formalism are integrated producing the DRPFT formalism. The DRPFT analysis supported by SWNs is discussed in this chapter, together with the way to detect and classify modules. The architecture of a software framework for the DRPFT analysis based on these concepts, is presented in this chapter. The graph transformation rules to convert a DRPFT model into SWN, are the content of appendix A.

The explanation of the concepts in this thesis, is supported by a running example consisting of a multiprocessor computing system [11, 69]. In each chapter, this system is modelled according to the formalism presented in the chapter evidencing particular aspects of the system behaviour that the facilities in the current formalism allow to represent. Then, the system model is analyzed according to the solution technique proposed in each chapter.

Chapter 2

Overview on Fault Trees

2.1 Introduction to Fault Trees

The *Fault Tree* (FT) [90] is a widespread stochastic model for the Reliability analysis of complex systems because it provides an intuitive representation of the system failure mode, it is easy to manipulate and it is currently supported by several software tools for its analysis. A FT models how combinations of failure events relative to the components of the system, can cause the failure of subsystems or of the whole system. An example of FT model is shown in Fig. 2.3.

The FT is a bipartite *direct acyclic graph* (DAG) whose nodes can belong to one of these two categories: events and gates; events concern the failure of components, subsystems or of the whole system, and they are in general graphically represented as rectangles; we can consider an event as a Boolean variable: it is initially $false$ and it becomes $true$ after the failure occurrence.

The events graphically represented as a rectangle with an attached circle are called *Basic Events* (BEs) and model the failure of the elementary components of the system; the occurrence time of such events is a random variable ruled by a probability distribution, typically the negative exponential distribution. In this case, the distribution parameter is the component failure rate λ equal to the inverse of the mean life time of the component. The BEs are statistically independent and are the terminal nodes of the FT.

The events represented simply by a rectangle are non-terminal nodes and represent the failure of subsystems; we call them *Internal Events* (IEs) and their occurrence is not ruled by a probability distribution as in the case of BEs, but they are the output of a gate node; *gates* are the other category of nodes that a FT can contain, and they are connected by means of arcs to several input events and to a unique output event; the effect of a gate is the propagation of the failure to its output event if a particular combination of its input events occurs. The occurrence of an IE is immediate, as soon as the particular combination of input events required by the gate is verified.

In the standard version of the FT model, three Boolean gates corresponding to

the AND ($\wedge$), OR ($\vee$) and "K out of N" ($K : N$) Boolean functions, are present. In particular, the $K : N$ function is applied to N Boolean variables; it returns $true$ if the value of at least K of the N Boolean variables is $true$; otherwise the $K : N$ function returns $false$. Actually, the $K : N$ Boolean function can be expressed by means of the functions AND, OR. For this reason and for sake of simplicity, we limit our attention on the AND gate and the OR gate, excluding the $K : N$ gate (also referred as voting gate).

Finally, we have a unique event, represented as a black rectangle, called *Top Event* (TE), modelling the failure of the whole system; the TE must be the output of a gate and can not be the input of any gate: we can consider it as the "root" of the FT.

A FT is a direct graph: arcs respect a logic circuit orientation: from the input events to the gate, and from the gate to the output event. A FT is also an acyclic graph, so the connection of events with gates (and vice-versa) by means of arcs, must not determine the presence of cyclic paths in the FT.

A FT model encodes a Boolean formula expressing the failure of the system (TE); the Boolean variables of such formula are the BEs of the FT and are combined through Boolean operators corresponding to the gates present in the FT.

2.2 FT formalism definition

A FT is a bipartite DAG whose nodes are either events ($\mathcal{E}$) or gates ($\mathcal{G}$), and are connected by means of arcs ($\mathcal{A}$); so, the FT formalism is given by the tuple
$\mathcal{FT} = (\mathcal{E}, \mathcal{G}, \mathcal{A}, \mathcal{BG}, \gamma, \lambda, \phi)$
where:

- $\mathcal{E} = \mathcal{BE} \cup \mathcal{IE} \cup \{TE\}$ is the set of the events in the FT; it is the union of the following sets:
  - $\mathcal{BE}$ is the set of the BEs;
  - $\mathcal{IE}$ is the set of the IEs;
  - {TE} is the set composed by the unique TE.
- $\mathcal{A} \subseteq (\mathcal{E} \times \mathcal{G}) \bigcup (\mathcal{G} \times \mathcal{E})$ is the set of the arcs according to the logic circuit orientation.
- $\mathcal{BG} = \{AND, OR\}$ is the set of Boolean gate types and is composed by the AND gate type and the OR gate type (for sake of simplicity, we omit the $K : N$ gate type).
- $\gamma : \mathcal{G} \rightarrow \mathcal{BG}$ is the function assigning to each gate its type.
- Given $g \in \mathcal{G}$,
  - $\bullet g = \{e \in \mathcal{E} : \exists (e, g) \in \mathcal{A}\}$ is the set of input events of g;

  - $g\bullet = \{e \in \mathcal{E} : \exists(g, e) \in \mathcal{A}\}$ is the output event of g.

  Given $e \in \mathcal{E}$,

  - $\bullet e = \{g \in \mathcal{G} : \exists(g, e) \in \mathcal{A}\}$ is the gate having e as output event;
  - $e\bullet = \{g \in \mathcal{G} : \exists(e, g) \in \mathcal{A}\}$ is the set of gates having e as one of their input events.

- The following conditions about the connection of events with gates, must hold:

  - $\forall g \in \mathcal{G} : \gamma(g), |\bullet g| \geq 2$
  - $\forall g \in \mathcal{G}, |g \bullet| = 1$
  - $\forall e \in \mathcal{BE}, |\bullet e| = 0$
  - $\forall e \in \mathcal{BE}, |e \bullet| > 0$
  - $\forall e \in \mathcal{IE}, |\bullet e| = 1$
  - $\forall e \in \mathcal{IE}, |e \bullet| > 0$
  - $|\bullet TE| = 1$
  - $|TE \bullet| = 0$

  In other words, a gate of any type has at least two input events. Any gate has one output event. A BE can not be the output of any gate, while the TE can not be the input of any gate. A BE or an IE must be the input of at least one gate.

- $\lambda : \mathcal{BE} \rightarrow \mathbb{R}^+$ is the function assigning to each BE a failure rate, assuming that BEs are ruled by the negative exponential distribution.

- $\phi : \mathcal{E} \rightarrow \mathbb{B} = \{true, false\}$ is the function returning the Boolean value of an event (Boolean variable). Given $y \in \{\mathcal{E} - \mathcal{BE}\}$, $\{x_1, \ldots, x_n\} \subset \{\mathcal{E} - TE\}$, $g \in \mathcal{G}$, $\bullet g = \{x_1, \ldots, x_n\}$, $g\bullet = \{y\}$,

  - if $\gamma(g) = AND$ then $\phi(y) = \bigwedge_{i=1}^{n} \phi(x_i)$
  - if $\gamma(g) = OR$ then $\phi(y) = \bigvee_{i=1}^{n} \phi(x_i)$

  Considering that the $true$ value of an event indicates the fact that the event has occurred, if a gate is of type AND (Fig. 2.1.a), then its output event occurs if all the input events have occurred; if a gate is of type OR (Fig. 2.1.b), then the output event occurs if at least one of the input events has occurred.

  FTs containing these three types of gate are referred as *coherent* FTs [90]; this means that in the Boolean formula equivalent to a FT model, the negation operator ($\neg$) is not present. The negation of events (Boolean variables) is

possible in FT models, if other two types of gate are used: NOT and XOR [90]. The NOT gate corresponds to the negation operator, and has only one input event and the Boolean value of its output event is the negation of the value of its input event. The XOR gate indicates the *exclusive OR* operator; if an IE is the output of a XOR gate, its value is *true* if exactly one of the several input events of the XOR gate is *true*. However, we limit our attention on coherent FTs.

- According to the circuit logic orientation of the FT arcs (section 2.2), we have a *path* between the node $u_1 \in \{\mathcal{E} \cup \mathcal{G}\}$ and the node $u_k \in \{\mathcal{E} \cup \mathcal{G}\}$ if there is a sequence of nodes $u_1, u_2, \ldots, u_{k-1}, u_k \in \{\mathcal{E} \cup \mathcal{G}\}$ such that $\forall i \in \{1, \ldots, k-1\}, \exists (u_i, u_{i+1} \in \mathcal{A})$.
  A path between the nodes u_1 and u_k is indicated by the expression $[u_1 \rightarrow u_k]$ and includes all the nodes and the arcs along that path:
  $[u_1 \rightarrow u_k] = [u_1, (u_1, u_2), u_2, \ldots u_{k-1}, (u_{k-1}, u_k), u_k]$

- Given the events $e, e' \in \mathcal{E}$, e is *reachable* from e' if $\exists [e' \rightarrow e]$.

- Given $e \in \mathcal{E}$, $\circ e = \{e' \in \mathcal{E} : \exists [e' \rightarrow e]\}$.
  In other words, $\circ e$ is the set of events such that e is reachable from them. In a FT, the following property must hold:
  $\forall e \in \mathcal{E}, e \in \circ TE$

- Given $e \in \mathcal{E}$, we indicate the subtree rooted in e with the expression $\widehat{e}$, where $\widehat{e}$ is composed by any $[b \rightarrow e] : b \in \mathcal{BE}$.

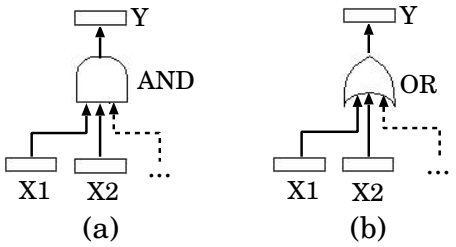

Figure 2.1: Boolean gates: (a) AND, (b) OR.

2.2.1 Running example

This section provides a case study of multiprocessor computing system (referred in the next sections as "Multiproc" system and inspired from [69]). First, a general description of the system is provided (section 2.2.1); then, the failure mode of the system (section 2.2.1) is modelled by means of a FT model (section 2.2.1).

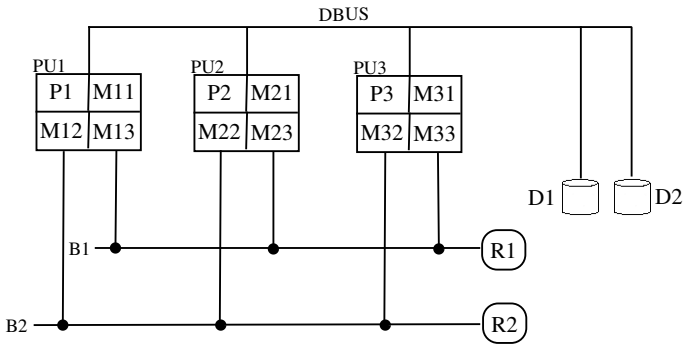

Figure 2.2: Scheme of the Multiproc system.

System description

The system consists of a multiprocessor computing system whose scheme is shown in Fig. 2.2; it is mainly composed by three processing units ($PU1$, $PU2$, $PU3$), two shared memories ($R1$, $R2$) and two hard disks ($D1$, $D2$) containing the software and the data respectively.

Each processing unit is composed by one processor and three internal memories; in the case of the device $PU1$, they are indicated by $P1$, $M11$, $M12$, $M13$, respectively.

The shared memory $R1$ is connected to the processing units by means of the bus $B1$, while the shared memory $R2$ can be accessed by the processing units by means of the bus $B2$. Each computing device can use $R1$ and $R2$ to perform computations when its internal memories are failed.

The connection of the processing units with the hard disks is established by the bus $DBUS$.

The failure mode of the system

We suppose that the correct functioning of at least one processing unit is required for the system to be working; so, the failure of all the processing units causes the whole system failure. The failure of a processing unit is due to the failure of its processor, or to the contemporary failure of all its internal memories together with the denied access to the shared memories. It is not possible the use of $R1$ when it is failed or when the bus $B1$ is failed; in the same way, $R2$ can not be accessed by the processing units when $R2$ is failed or when the bus $B2$ is failed.

The whole system failure is caused also by the impossibility to access any hard disk; this happens if the failure of at least one of them occurs, or if the bus $DBUS$ fails.

The probability of failure of the components of the system, obeys to the negative exponential distribution; Tab. 2.1 indicates the failure rates for each type of component.

Component	Failure rate (λ)
Processor	5.0E-7 h^{-1}
Disk	8.0E-7 h^{-1}
Memory	3.0E-8 h^{-1}
Bus	2.0E-9 h^{-1}

Table 2.1: The failure rate for each type of component.

FT model of the system

Fig. 2.3 shows the FT model for the Multiproc system; the system failure (TE) is the output of a gate of type OR, whose input events are DA (disk access) and CM (computing module); CM represents the impossibility to perform computations due to the failure of all the processing units, while DA represents the impossibility to access the hard disks.

The event CM is the output of a gate of type AND, whose input events are $PU1$, $PU2$ and $PU3$ representing the failure of the corresponding processing units. $PU1$ is the output of a gate of type OR having $P1$ and $MEM1$ as input events; $P1$ is a BE modelling the failure of the processor of $PU1$, while $MEM1$ represents the failure of the memories that $PU1$ can access.

The event $MEM1$ is the output of a gate of type AND having two inputs: $MM1$ and SM; $MM1$ represents the failure of all the internal memories of $PU1$, while SM models the failure of the shared memories. $MM1$ is the output of a gate of type AND, whose input events are $M11$, $M12$ and $M13$; such BEs represent the failure of each of the internal memories of $PU1$. The event SM is the output of a gate of type AND with input events $BR1$ and $BR2$; $BR1$ represents the impossibility to access the shared memory $R1$, so $BR1$ is the output of a gate of type OR having $R1$ and $B1$ as input events; such BEs represent the failure of the shared memory $R1$ and of the bus $B1$, respectively. Similarly, $BR2$ is the output of a gate of type OR having $R2$ and $B2$ as input events.

The subtrees having $MEM1$, $MEM2$ and $MEM3$ respectively as roots, share a common subtree rooted in the event SM, since the failure of all the processing units may depend on the state of the shared memories.

Besides the occurrence of the event CM, TE is caused also by the event DA; such event is the output of a gate of type OR, whose input events are $DBUS$ and MS. $DBUS$ models the failure of the disk bus, while MS is the failure of at least one disk; MS is the output of a gate of type OR, whose input events are $D1$ and $D2$.

Tab. 2.2 summarizes the names of the events associated with the failure of the components or subsystems.

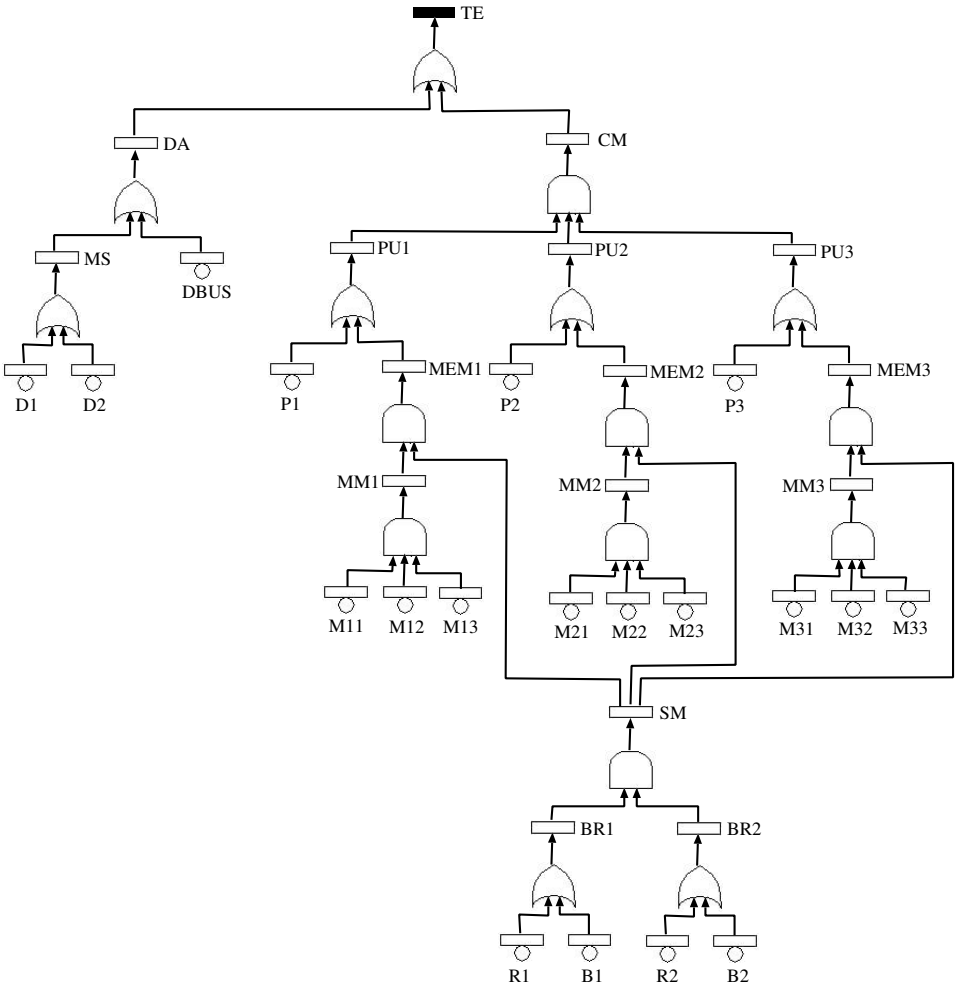

Figure 2.3: The FT model of the Multiproc system.

Event	Component / Subsystem
DA	Disk Access
$DBUS$	Disk Bus
MS	Mass Storage
$D1$	System Disk
$D2$	Data Disk
CM	Computing module
$PU1$	Processing Unit 1
$P1$	Processor of the Processing Unit 1
$MEM1$	Memory access of the Processing Unit 1
$MM1$	Internal Memory Module of the Processing Unit 1
$M11$	Internal Memory 1 of the Processing Unit 1
$M12$	Internal Memory 2 of the Processing Unit 1
$M13$	Internal Memory 3 of the Processing Unit 1
$PU2$	Processing Unit 2
$P2$	Processor of the Processing Unit 2
$MEM2$	Memory access of the Processing Unit 2
$MM2$	Internal Memory Module of the Processing Unit 2
$M21$	Internal Memory 1 of the Processing Unit 2
$M22$	Internal Memory 2 of the Processing Unit 2
$M23$	Internal Memory 3 of the Processing Unit 2
$PU3$	Processing Unit 3
$P3$	Processor of the Processing Unit 3
$MEM3$	Memory access of the Processing Unit 3
$MM3$	Internal Memory Module of the Processing Unit 3
$M31$	Internal Memory 1 of the Processing Unit 3
$M32$	Internal Memory 2 of the Processing Unit 3
$M33$	Internal Memory 3 of the Processing Unit 3
SM	Shared Memory access
$BR1$	Shared Memory module 1
$R1$	Shared Memory 1
$B1$	Memory Bus of the Shared Memory 1
$BR2$	Shared Memory module 2
$R2$	Shared Memory 2
$B2$	Memory Bus of the Shared Memory 2

Table 2.2: Correspondence between the event names and the components or subsystems.

2.3 Fault Tree Analysis

The *Fault Tree Analysis* (FTA) [95, 96] provides a set of techniques enabling to derive both qualitative results and quantitative measures from a FT model.

2.3.1 Qualitative analysis

The qualitative analysis supplies information about functional and logic properties of the system failure mode. One of the possible results of the qualitative analysis of a FT, is the detection of the *Minimal Cut Sets* (MCS) [46] of the system. The MCSs of a FT correspond the minimal scenarios of BEs leading the system to a failure. In other words, a MCS indicates a minimal set of components whose contemporary state of failure determines the whole system failure (TE).

Given the Boolean formula $F(x_1, \ldots, x_n)$ encoded by a FT and expressed on the set of variables $\mathcal{X} = \{x_1, \ldots, x_n\}$ corresponding to the BEs of the FT, an assignment over $\mathcal{X}$ is any mapping from $\mathcal{X}$ to $\mathbb{B} = \{true, false\}$.

If ρ is an assignment satisfying F ($\rho \models F$), then ρ is a solution of F; in other words, ρ makes F $true$. Given $\rho \models F$, F is *monotone* if

$$\forall x_i : \rho[x_i] = 0, \forall \rho' : (\rho'[x_i] = 1) \wedge (\rho'[x_j] = \rho[x_j]) \wedge (i \neq j), \rho' \models F \quad (2.1)$$

A Boolean formula where the operators are only $\vee$, $\wedge$, is monotone. Any formula encoded by a coherent FT is monotone.

A *literal* of $\mathcal{X}$ is either a Boolean variable $x_i \in \mathcal{X}$ or its negation $\neg x_i$. A *cut set* C of F is a set literals $C = \{l_1 \ldots l_m\} \subseteq \mathcal{X}$ such that if the conjunction of its elements is $true$, then F is $true$; if we indicate such conjunction as $\pi = \bigwedge_{j=1}^{m} l_j$, then C is a cut set if

$$\forall \rho : \rho \models \pi, \rho \models F \quad (2.2)$$

In other words, any solution of π is a solution of F. This property can also be expressed as $\pi \models F$. A cut set C of F is minimal (MCS) if there is no cut set C' of F such that $C' \subset C$.

In the case of monotic functions (coherent FTs), any cut set is composed by literals which are not negated variables, so there is only one assignment satisfying the conjunction of the literals of the cut set; this assignment maps any Boolean variable in the cut set to the value $true$. So, the detection of the MCSs of a coherent FT, consists of finding the minimal sets of non negated variables such that if their conjunction is $true$, also the formula encoded by the FT is $true$. In other words, a MCS of a coherent FT indicates a minimal set of BEs whose occurrence is necessary to determine the TE.

Let $F(x_1, \ldots, x_n)$ be the Boolean function encoded by a FT. According to the *Decomposition Theorem for Prime Implicants (or MCSs)* [45, 80], the set of the MCSs of F expressed as the conjunction of its elements, can be obtained as the union of three sets:

$$MCS[F(x_1, \ldots, x_n)]] = MCS_{10} \cup MCS_1 \cup MCS_0 \quad (2.3)$$

The definition of MCS_{10}, MCS_1 and MCS_0 follows:

$$\begin{aligned} MCS_{10} &= MCS[F(1, x_2, \ldots, x_n) \wedge F(0, x_2, \ldots, x_n)] \\ MCS_1 &= x_1 \wedge (MCS[F(1, x_2, \ldots, x_n)] - MCS_{10}) \\ MCS_0 &= \neq x_1 \wedge (MCS[F(0, x_2, \ldots, x_n] - MCS_1) \end{aligned}$$

If the FT is coherent, and consequently the encoded Boolean function in monotonic, the definition of MCS_{10}, MCS_1 and MCS_0 can be rewritten as follows:

$$\begin{aligned} MCS_{10} &= MCS[F(0, x_2, \ldots, x_n)] \\ MCS_1 &= x_1 \wedge (MCS[F(1, x_2, \ldots, x_n)] - MCS_{10}) \\ MCS_0 &= \emptyset \end{aligned}$$

So, eq. 2.3 in the case of coherent FTs, becomes:

$$MCS[F(x_1, \ldots, x_n)]] = MCS_{10} \cup MCS_1 \tag{2.4}$$

The recursive application of eq. 2.4 over all the variables $\{x_1, \ldots, x_n\}$ in the Boolean formula encoded by the coherent FT, provides the MCSs of the system.

The number of BEs (Boolean variables) in a MCS is called the *order* of the MCS. The order is a significant qualitative parameter since it highlights failure sets of events that might be more critical for the system. In fact, a MCS of order 1 means that the failure of a single basic component is sufficient to determine the TE, indicating no fault tolerance with respect to that component. In a MCS of order 2, two simultaneous failures of basic components are needed. For this reason, it is useful cataloging the MCS in increasing order, so that the list starts with those that are potentially most critical.

Another form of qualitative analysis of a FT, is the *Minimal Path Sets* (MPS) detection, which is dual to the MCSs detection and provides the minimal sets of components whose contemporary working state assures the working state of the whole system.

MCSs and MPSs detection allows the Reliability analyst to be concentrated on minimal sets of components instead of the whole system, in order to study their degree of participation to the system failure, or the way to improve the Reliability of the system.

2.3.2 Quantitative analysis

FT quantitative analysis presents the analyst with measures of system Unreliability. Several Reliability measures can be computed on the FT: the TE probability at time t corresponding to the system Unreliability at the same type, the occurrence probability of each MCS, and the system *Mean Time To Failure* (MTTF). The quantitative analysis of a FT provides also the component importance factors, which give quantitative information on the criticality of each component.

Quantitative analysis is performed by providing quantitative information about the basic component Unreliability expressed as a failure probability (that is the probability of the component being down). From this information the whole system Unreliability can be derived according to the FT structure. Often the failure probability of components is not directly expressed in the FT, instead a time to failure distribution is provided, from which a failure probability at time t can be derived. Typically the distribution is a negative exponential with parameter $\lambda(x)$, so that the probability that component x is down at time t can be computed through the following formula:

$$Pr\{x,t\} = 1 - e^{\lambda(x)t} \tag{2.5}$$

MCSs can also be ranked according to their probability to be occurred at a given time; given the MCS $M' = \{x_1, \ldots, x_m\}$ ($m \geq 1$), the probability of M' to be occurred at time t, is given by

$$Pr\{M',t\} = Pr\{\bigwedge_{i=1}^{m} x_i = true, t\} = \prod_{i=1}^{m} Pr\{x_i,t\} \tag{2.6}$$

If the probability to occur of the BEs $x_1, \ldots, x_m$ is ruled by a negative exponential distribution, the probability of M' to be occurred at time t is given by

$$Pr\{M',t\} = \prod_{i=1}^{m}(1 - e^{\lambda(x_i)t}) \tag{2.7}$$

where $\lambda(x_i)$ is the failure rate of the BE x_i.

The most common measure used to assess the system safety is its Unreliability in time. The system Unreliability, denoted by U , is a function of the basic components Unreliability: $U(u(t))$, where $u(t)$ is the vector of basic component failure probabilities. In FT such indicator corresponds to the TE probability to be occurred at time t ($Pr\{TE,t\}$). When the basic components are stochastically independent, such as in FTs, $Pr\{TE,t\}$ may be computed resorting to combinatorial formulas. In fact, for a given time instant t and fixed the basic components failure occurrence probabilities $u(t)$, the TE probability can be derived. This can be done exploiting the results of the MCSs detection and their quantitative analysis, by using the following inclusion-exclusion expansion, where $C_1, \ldots, C_n$ are the MCSs:

$$\begin{aligned} Pr\{TE,t\} &= Pr\{\bigcup_{i=1}^{n} C_i, t\} = \\ &= \sum_{i=1}^{n} Pr\{C_i,t\} - \sum_{\forall i \forall j} Pr\{C_i \cap C_j, t\} + \\ &\quad + \sum_{\forall i \forall j \forall k} Pr\{C_i \cap C_j \cap C_k, t\} + \\ &\quad + \ldots + (-1)^{n+1} \cdot Pr\{C_1 \cap \ldots \cap C_n, t\} \end{aligned} \tag{2.8}$$

For complex systems the above formula may be prohibitive to derive due to the huge amount of calculation to be performed; as a result, most FTA tools compute an approximation [88] based on the kinetic tree theory [104]. Recently, the computation of both qualitative results and quantitative measures has improved notably by the introduction of *Binary Decision Diagrams* (BDD) [16, 17, 18, 38, 79, 87, 111]. BDDs allow to encode the Boolean function characterizing the TE in a very compact way. The BDD representation had an enormous practical impact on the computational efficiency of the FT analysis algorithms allowing to derive the exact value of the indices of Reliability even for large system, without resorting to an approximate solution. In [15, 78, 94] innovative BDD-based algorithms showing improvement in qualitative and quantitative FTA of safety critical industrial systems are provided.

2.3.3 Importance measures

Among the risk-assessment and safety-analysis objectives, the classification of system's components according to their criticality is very important. So, in addition to the Reliability measure of a system or its own subcomponents, it is central to assess the role that a component takes on, with regard to the system Reliability. This analysis is significant to the Reliability engineer both during the design phase and successively in quantifying the risk-importance of the various system components.

Importance analysis allows to assess which component of the system is most critical to its Unreliability, so to find out the more cost effective solution to improve Reliability. Moreover, it permits to evaluate how much the results depend on the accuracy of the input parameters. To this purpose, several indices, commonly called the importance factors, have been proposed. Importance factors are time dependent measures and may be divided in two groups: measures calculated at one point in the time, such as those discussed later, and measures whose values are obtained averaging on a time period [75].

Among the various importance factors introduced in the context of importance analysis using FT, the most popular is due to Birnbaum [7] and is defined as the partial derivative of the system Unreliability with respect to the Unreliability of the addressed component. This measure is also known as *Marginal Impact Factor* (MIF). The importance of component i to the system Unreliability is by Birnbaum defined as:

$$MIF_i(t) = \frac{\partial U(u(t))}{\partial u_i(t)} \tag{2.9}$$

It is possible to show that for standard FTs, $MIF_i(t)$ is equal to:

$$MIF_i(t) = Pr\{TE|i,t\} - Pr\{TE|\bar{i},t\} \tag{2.10}$$

where $Pr\{TE|i,t\}$ and $Pr\{TE|\bar{i},t\}$ are the system Unreliability given that basic component i is failed and working (not failed), respectively.

Others measures of component importance have been proposed by the literature and they may be found in textbooks [63, 67]. Tab. 2.3 shows some of them.

Importance factor	Acronym	Definition
Marginal Importance Factor	$MIF_i(t)$	$Pr\{TE\|i,t\} - Pr\{TE\|\bar{i},t\}$
Critical Importance Factor	$CIF_i(t)$	$MIF_i(t) \cdot Pr\{i,t\}/Pr\{TE,t\}$
Diagnostic Impact Factor	$DIF_i(t)$	$Pr\{i\|TE,t\} = Pr\{TE \wedge i,t\}/Pr\{TE,t\}$
Risk Achievement Worth	$RAW_i(t)$	$Pr\{TE\|i,t\}/Pr\{TE,t\}$
Risk Reduction Worth	$RRW_i(t)$	$Pr\{TE,t\}/Pr\{TE\|\bar{i},t\}$

Table 2.3: Importance factors.

The MIF measure is useful to evaluate the criticality that an improvement in the component i Reliability may play in the system Reliability. The CIF index extends the MIF index to take into account such factor. The DIF is also known as *Vesley-Fussel Importance factor* [55] and it measures the fraction of the system Unreliability involving the situations in which component i has failed.

RAW_i for a given component i measures the increase in system failure probability, and calculated for different values of $u_i(t)$ it is a meter of the importance of maintaining the current level of Reliability for the component i. RRW represents the maximum decreasing of the risk it may be expected by increasing the Reliability of the component. This quantity may be useful to rank the components that are the best candidates for efforts leading to improving system Reliability.

Several works [43, 47, 48, 78, 94] presented efficient algorithms to derive exactly the importance factors above described using techniques based on BDD representations. Many FTA tools exploit these techniques. In the above discussion only single component importance factor measures have been introduced, but in principle it may be relevant to also compute importance factors for sets of components (e.g. for the MCS). In [13, 77] *Bayesian Network* (BN) are used to derive the posterior failure probability of a given subset of events, i.e. the probability that when the TE is observed, then that subset of events had occurred.

2.4 BDD based FT analysis

An efficient way to perform both the qualitative and quantitative analysis of a FT model, consists of generating from the FT model the corresponding BDD. From the BDD, the MCSs and the TE probability (system Unreliability) can be easily obtained by visiting the BDD nodes.

2.4.1 Introduction to BDDs

A FT model encodes a Boolean formula by means of BEs (corresponding to Boolean variables) and gates (corresponding to Boolean operators). A BDD can represent the same formula by means of the Shannon's decomposition: if F is a Boolean

function on variables $x_1, \ldots, x_n$, then equation 2.11 holds:

$$F = x_1 \wedge F_1 \vee \neg x_1 \wedge F_0 \tag{2.11}$$

In equation 2.11, F_1 is derived from F assuming that x_1 is $true$ ($F_1 = F|x_1 = true$), while F_0 is derived from F assuming that x_0 is $false$ ($F_0 = F|x_1 = false$).

Using recursively the Shannon's decomposition (equation 2.11), we can express F_1 as
$F_1 = x_2 \wedge F_{11} \vee \neg x_2 \wedge F_{10}$.
We can express F_0 as
$F_0 = x_2 \wedge F_{01} \vee \neg x_2 \wedge F_{00}$.
Shannon's decomposition can be recursively applied until each combination of Boolean values for the variables $x_1, \ldots x_n$ has been considered.

The equation 2.11 can be redefined using the *if-then-else* (ite) notation:

$$F = ite(x_1, F_1, F_0) \tag{2.12}$$

and can be given a graphical representation. Equation 2.12 must be interpreted as: if x_1 then F_1 else F_0. In other words, if x_1 is true, then F_1 holds; if x_1 is false, then F_0 holds. Also F_1 and F_0 can be expressed using the *ite* notation:
$F_1 = ite(x_2, F_{11}, F_{10})$
$F_0 = ite(x_2, F_{01}, F_{00})$

Let us consider the Boolean formula $F = a \vee b \wedge c$ expressed on the variables a, b, c. We can express G using the ite notation in this way:

$$\begin{aligned} G &= ite(a, F_1, F_0) = \\ &= ite(a, 1, F_0) = \\ &= ite(a, 1, ite(b, F_{01}, F_{00}) = \\ &= ite(a, 1, ite(b, ite(c, 1, 0), F_{00}) = \\ &= ite(a, 1, ite(b, ite(c, 1, 0), 0)) \end{aligned}$$

The final version of the Shannon's decomposition on a Boolean formula can be displayed by means of a BDD. A BDD is a DAG representing the Boolean formula in *ite* notation; for instance, the expression $ite(a, F_1, F_0)$ is represented in a BDD by the node corresponding to the variable a, with two outgoing edges called 1-edge and 0-edge. The 1-edge points to the subgraph relative to F_1, while the 0-edge points to the subgraph relative to F_0. Typically, the 1-edge is drawn on the left side of the node, while the 0-edge is drawn on the right side. The terminal nodes of a BDD correspond to the constants 1 and 0 which indicate the $true$ and the $false$ value for the whole Boolean formula, respectively, given the values assigned to the variables, by means of the 1-edges and 0-edges, along a path from the root node of the BDD, to a terminal node.

The BDD obtained for the boolean formula $F = a \vee b \wedge c$, is shown in Fig. 2.4; we can verify the equivalence of the BDD to the complete *ite* notation of F, and

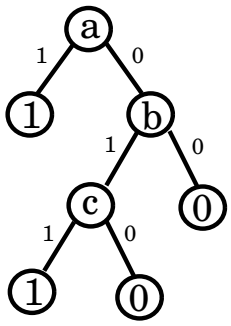

Figure 2.4: The BDD expressing the Shannon's decomposition for the Boolean formula $F = a \vee b \wedge c$.

compute the solution of the formula F for each combination of values assigned to the variables a, b, c. For instance, the path $a \rightarrow b \rightarrow c \rightarrow 1$ indicates that if $a = 0$, $b = 1$ and $c = 1$, then $F = 1$. Another possible path is $a \rightarrow b \rightarrow 0$ and indicates that if $a = 0$ and $b = 0$, then $F = 0$.

2.4.2 BDD construction

Given a FT model (or the equivalent Boolean formula), the construction of the corresponding BDD begins with the creation of a BDD for each BE (Boolean variable). Given the BE (Boolean variable) x, we obtain this BDD expressed in *ite* form: $ite(x, 1, 0)$.

Let us consider two subtrees[1] of the FT encoding the formula F and the formula G respectively; F in BDD form is expressed by $ite(a, F_1, F_0)$, while G in BDD form is expressed by $ite(b, G_1, G_0)$, where a and b correspond to BEs (Boolean variables). Suppose that the roots of these subtrees are the input events of the same gate g. If the type of g is OR, then the BDD resulting from the application of the $OR(\vee)$ Boolean operator to the BDDs corresponding to F and G is given by

$$ite(a, F_1, F_0) \vee ite(b, G_1, G_0) = ite(a, F_1 \vee ite(b, G_1, G_0), F_0 \vee ite(b, G_1, G_0)) \tag{2.13}$$

If the type of g is AND, then the BDD resulting from the application of the $AND(\wedge)$ Boolean operator to the BDDs corresponding to F and G is given by

$$ite(a, F_1, F_0) \wedge ite(b, G_1, G_0) = ite(a, F_1 \wedge ite(b, G_1, G_0), F_0 \wedge ite(b, G_1, G_0)) \tag{2.14}$$

If both BDDs have the same root node c ($F = ite(c, F_1, F_0), G = ite(c, G_1, G_0)$), then the application of the Boolean operators $OR(\vee)$, $AND(\wedge)$ is ruled by eq.

[1] Despite its name, a FT is not a tree graph, but it is a DAG; so, we force the use of the expression "subtree" to indicate a subgraph of the FT.

2.15 and eq. 2.16, respectively:

$$ite(c, F_1, F_0) \vee ite(c, G_1, G_0) = ite(c, F_1 \vee G_1, F_0 \vee G_0) \tag{2.15}$$

$$ite(c, F_1, F_0) \wedge ite(c, G_1, G_0) = ite(c, F_1 \wedge G_1, F_0 \wedge G_0) \tag{2.16}$$

The commutative property for both operators hold in semantic terms, but it does not hold in terms of graph topology. For instance, $ite(x, F_1, F_0) \vee ite(y, G_1, G_0)$ provides a BDD expressing the same Boolean formula encoded by the $ite(y, G_1, G_0)$ $\vee$ $ite(x, F_1, F_0)$, but such BDDs differ in terms of variables order and number of nodes and edges. The size of the final BDD representing a Boolean formula, heavily depends on the chosen variables order to follow while we build the BDD.

For this reason, a total order of the variables of the Boolean formula must be established before applying eq. 2.13, 2.14, 2.15, 2.16. In eq. 2.13 and in eq. 2.14, we assume that x precedes y in the order ($x \prec y$).

Several heuristics for the variables ordering have been proposed in the literature [15] with the aim of reducing the BDD size or the number of steps necessary to build the BDD. Some of them are described in the next section.

A BDD built according to a total order of the Boolean variables is called *Ordered BDD* (OBDD). In a OBDD, given a path from the root node to a terminal node 1 or 0, the variables are visited in ascending order. The size of a OBDD can be reduced also by applying the BDD simplification rules:

- *Isomorphic subgraphs merging*: in a BDD, isomorphic subgraphs have the same topology and involve the same set of variables. Isomorphic subgraphs can be merged without changing the graph semantic; this can be done in this way: we maintain only one isomorphic subgraph and we remove all the other ones; then, we redirect all the edges pointing on the removed subgraphs, toward the maintained one.

- *Useless nodes deletion*: if a node x has both its outgoing edges pointing to the same node y, then x can be removed: each edge pointing to x is redirected to y.

After the simplification of an OBDD, we obtain a *Reduced OBDD* (ROBDD). In a ROBDD each subgraph represents a distinct logic function.

2.4.3 Ordering variables

Finding the best ordering of the Boolean variables corresponding to the FT BEs, in order to reduce the size of the BDD corresponding to the FT as much as possible, is computationally intractable. The best known algorithms to this aim have a complexity in $O(3^n)$, where n is the number of Boolean variables. For this reason, heuristics with an acceptable complexity are used to sort the variables in order to reduce the size of the BDD encoding the same Boolean formula of a FT. Several heuristics with different degree of efficiency, have been proposed in the literature. The description of some heuristics follows.

- Heuristic H1: variables are ordered according to a depth-first left-most traversal of the FT. The complexity of H1 is $O(n)$, where n is the number of Boolean variables (BEs).

- Heuristic A3 [72]: the application of this heuristic follows three steps:

  - it assigns to each BE in the FT the weight 1; then, the weights are propagated bottom-up by assigning to each IE the sum of the weights of the BEs contained in the subtree rooted in such IE.
  - The input events of each gate are sorted with respect to the increasing order of their weights. This leads to restructure the FT.
  - The heuristic H1 is applied to the FT resulting from the gates input events ordering in the previous step, providing the Boolean variables order.

  The complexity of the heuristic A3 is $O(e \cdot k \cdot log\ k)$, where e is the number of gates, k is the maximum number of input events over all the gates of the FT, and $O(k \cdot log\ k)$ is the complexity necessary to sort k numbers.

- Heuristic H7 [54]: the application of this heuristic follows three steps:

  - it determines for every BE and IE in the FT, the number of fanouts; the number of fanouts for the event e is equal to $|e \bullet|$ (see section 2.2).
  - The input events of each gate are sorted with respect to the decreasing order of their number of fanouts.
  - The heuristic H1 is applied to the FT resulting from the gates input events ordering in the previous step, providing the Boolean variables order.

  The heuristic H7 has the same complexity of the heuristic A3.

In most cases, such heuristics are not deterministic, due to the presence of *ties*. We have a tie when we are sorting events according to a certain measure (for instance, the number of fanouts) and the same value for that measure holds for two or more events. For instance, if we sort the events according to the heuristic H7, two input events of the same gate may have the same number of fanouts.

A way to partially cope with the problem of ties, is applying another heuristic in case of tie. For example, if we use the heuristic H7 and the events e_1 and e_2 have the same number of fanouts, we can sort e_1 and e_2 according to the heuristic A3.

The degree of efficiency of a heuristic (or a combination of heuristics) is given by observing the size of the BDD obtained by sorting the FT BEs according to the heuristic. The degree of efficiency of a heuristic is not absolute, but it is case dependent. This means that, given a FT model, a certain heuristic H may sort the BEs (Boolean variables) in such a way to obtain a BDD whose size is lower than the size of the BDDs obtained by sorting the BEs according to any another

heuristic. This may not happen for another FT model, where the heuristic H may be less efficient than the other ones.

In [15] the heuristics proposed in this section, are tested on benchmarks in order to verify their efficiency on real life coherent FTs presenting a wide range of specificities, such as size and structure.

Sometimes, it might be convenient to choose the order of the BEs (variables) of a FT, in an ad-hoc way.

2.4.4 Qualitative analysis on the BDD

Once we have built the BDD corresponding to our FT, we can perform the qualitative analysis; this means detecting the MCSs, i.e. the minimal sets of BEs determining the TE (see section 2.3.1).

If $F(x_1, \ldots, x_n)$ is the formula encoded by the FT, and $x_1 \prec x_2 \prec \ldots \prec x_n$ is the variables order, the BDD corresponding to the FT will have this ite form: $ite(x_1, F_1, F_0)$. Eq. 2.3 can be rewritten in order to be applied to the BDD (ite) form of a Boolean function, in this way:

$$MCS[ite(x_1, F_1, F_0)] = x_1 \wedge (MC[F_1] - MC[F_0]) \cup MC[F_0] \tag{2.17}$$

Using recursively this formula on the BDD starting from the root node and visiting the whole BDD, we can obtain all the MCSs of the system. The complexity of this algorithm is linear in the size of the BDD. When we apply eq. 2.17 to a node x_k of the BDD having the form $ite(x_k, 1, 0)$, we obtain that $MCS[ite(x_k, 1, 0)] = x_k$.

There are other ways to determine the MCSs on the BDD. All the cut sets can be detected on the BDD by considering all the paths from the root node to the constant 1, and for each of these paths, a cut set is given by the variables whose value is set to 1 ($true$) along the path. The paths from the root node to the constant 1 which do not include any other path, provide the MCSs.

Another solution consists of generating from the BDD obtained from the FT, another BDD representing the disjunction of the conjunctions of the elements of the MCSs. This can be done by means of a specific algorithm [45, 78].

2.4.5 Quantitative analysis on the BDD

We can perform the quantitative analysis on the BDD; this means computing the probability that the TE has occurred (system Unreliability) versus time.

If $F(x_1, \ldots, x_n)$ is the formula encoded by the FT, and $ite(x_1, F_1, F_0)$ is its BDD representation, the probability of the TE at a given time t can be computed by means of eq. 2.18:

$$Pr\{ite(x_1, F_1, F_0), t\} = Pr\{x_1, t\} \cdot Pr\{F_1, t\} + (1 - Pr\{x_1, t\}) \cdot Pr\{F_0, t\} \tag{2.18}$$

The value of $Pr\{x_1, t\}$ depends on the probability distribution of the BE x_1, and on the time t; if the probability of x_1 is ruled by a negative exponential distribution with $\lambda(x_1)$ as failure rate, then $Pr\{x_1, t\} = 1 - e^{\lambda(x_1) \cdot t}$.

When we apply eq. 2.18 to a node x_k of the BDD having the form $ite(x_k, 1, 0)$, we obtain that $Pr\{ite(x_k, 1, 0), t\} = Pr\{x_k, t\}$.

The complexity of this algorithm is linear in the size of the BDD.

Besides the system Unreliability, the importance factors (section 2.3.3) can be easily computed on the BDD corresponding to the FT model [47, 48, 76].

2.4.6 Running example

BDD construction

In this section, we build the BDD corresponding to the Boolean formula encoded by the FT model in Fig. 2.3. As first step, we need to order the BEs of the TE (variables of the Boolean formula). We choose this order:
$DBUS \prec D1 \prec D2 \prec R1 \prec B1 \prec R2 \prec B2 \prec$
$\prec P1 \prec M11 \prec M12 \prec M13 \prec P2 \prec M21 \prec M22 \prec M23 \prec$
$\prec P3 \prec M31 \prec M32 \prec M33$

The second step to build a BDD corresponding to a FT, is the generation of the BDD for each BE. For instance, in the case of the BE $DBUS$, we obtain $ite(DBUS, 1, 0)$. Using eq. 2.13, 2.14, 2.15, 2.16, and respecting the variables order, we can compose together the BDDs relative to the BEs according the Boolean operators corresponding to the gates, in order to build the BDDs corresponding to the subtrees of the FT.

For instance, the BDD equivalent to the Boolean formula encoded by the subtree $\widehat{DA}$ is shown in Fig. 2.5, while Fig. 2.6 shows the BDD corresponding to $\widehat{SM}$.

The IE $MEM1$ is the output of a gate of type AND having $MM1$ and SM as input events. Since, $R1 \prec MM1$, the BDD corresponding to $\widehat{MEM1}$ is given by the composition of the BDD in Fig. 2.6 ($\widehat{SM}$) whose root is $R1$, and the BDD in Fig. 2.7 ($\widehat{MM1}$) whose root is $M11$, by means of eq. 2.14; the resulting BDD is shown in Fig. 2.8. The BDD relative to $\widehat{PU1}$, is shown in Fig. 2.9.

The BDDs concerning $\widehat{PU2}$ and $\widehat{PU3}$ are similar to the BDD in Fig. 2.9, and they all have the same root: $R1$. For this reason, their composition, in order to obtain the BDD corresponding to $\widehat{CM}$ (Fig. 2.10), is performed using eq. 2.16.

Composing the BDD in Fig. 2.5 ($\widehat{DA}$) and the BDD in Fig. 2.10 ($\widehat{CM}$), using eq. 2.13, we obtain the BDD corresponding to the Boolean formula encoded by the whole FT ($\widehat{TE}$); such BDD is shown in Fig. 2.11.

Analysis of the example

Given the failure rates in Tab. 2.1, we have obtained on the BDD, the probability of the TE (Unreliability) for a time varying from $1000h$ to $10000h$, by recursively applying equation 2.18. Such values are reported in Tab. 2.4.

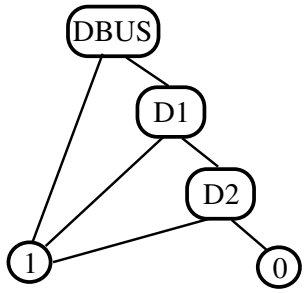

Figure 2.5: The BDD corresponding to $\widehat{DA}$ in the FT in Fig. 2.3.

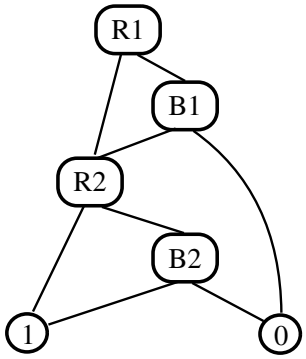

Figure 2.6: The BDD corresponding to $\widehat{SM}$ in the FT in Fig. 2.3.

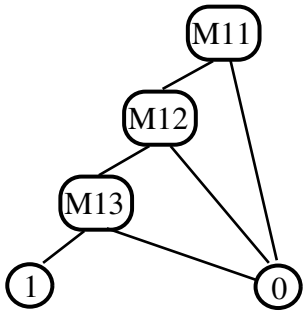

Figure 2.7: The BDD corresponding to $\widehat{MM1}$ in the FT in Fig. 2.3.

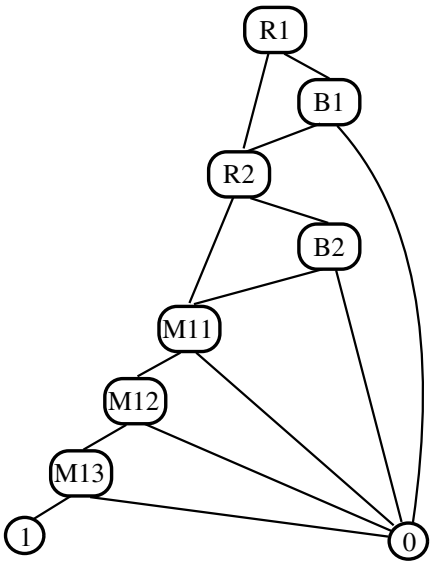

Figure 2.8: The BDD corresponding to $\widehat{MEM1}$ in the FT in Fig. 2.3.

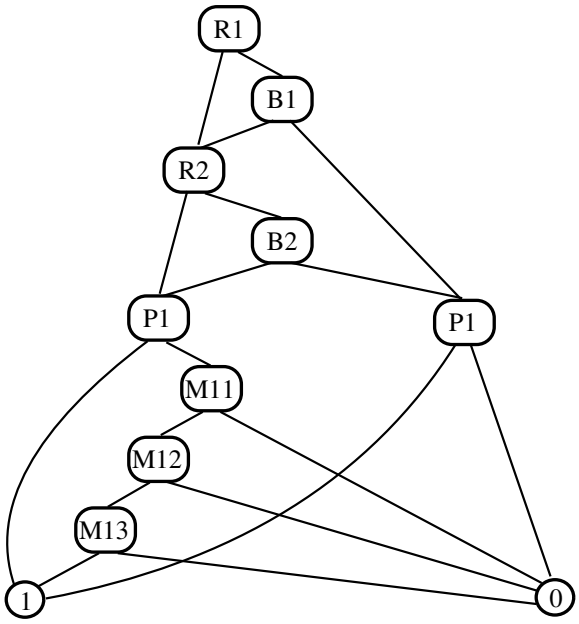

Figure 2.9: The BDD corresponding to $\widehat{PU1}$ in the FT in Fig. 2.3.

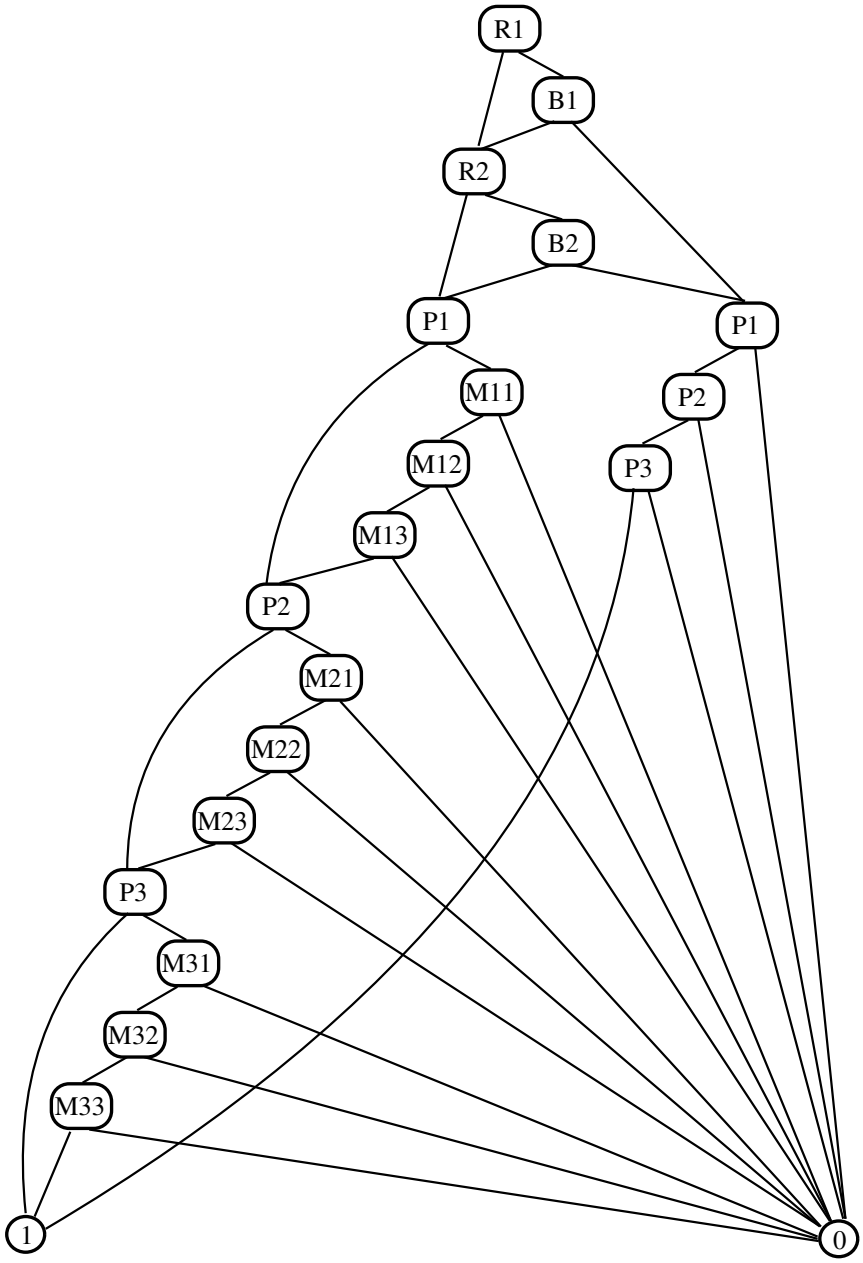

Figure 2.10: The BDD corresponding to $\widehat{CM}$ in the FT in Fig. 2.3.

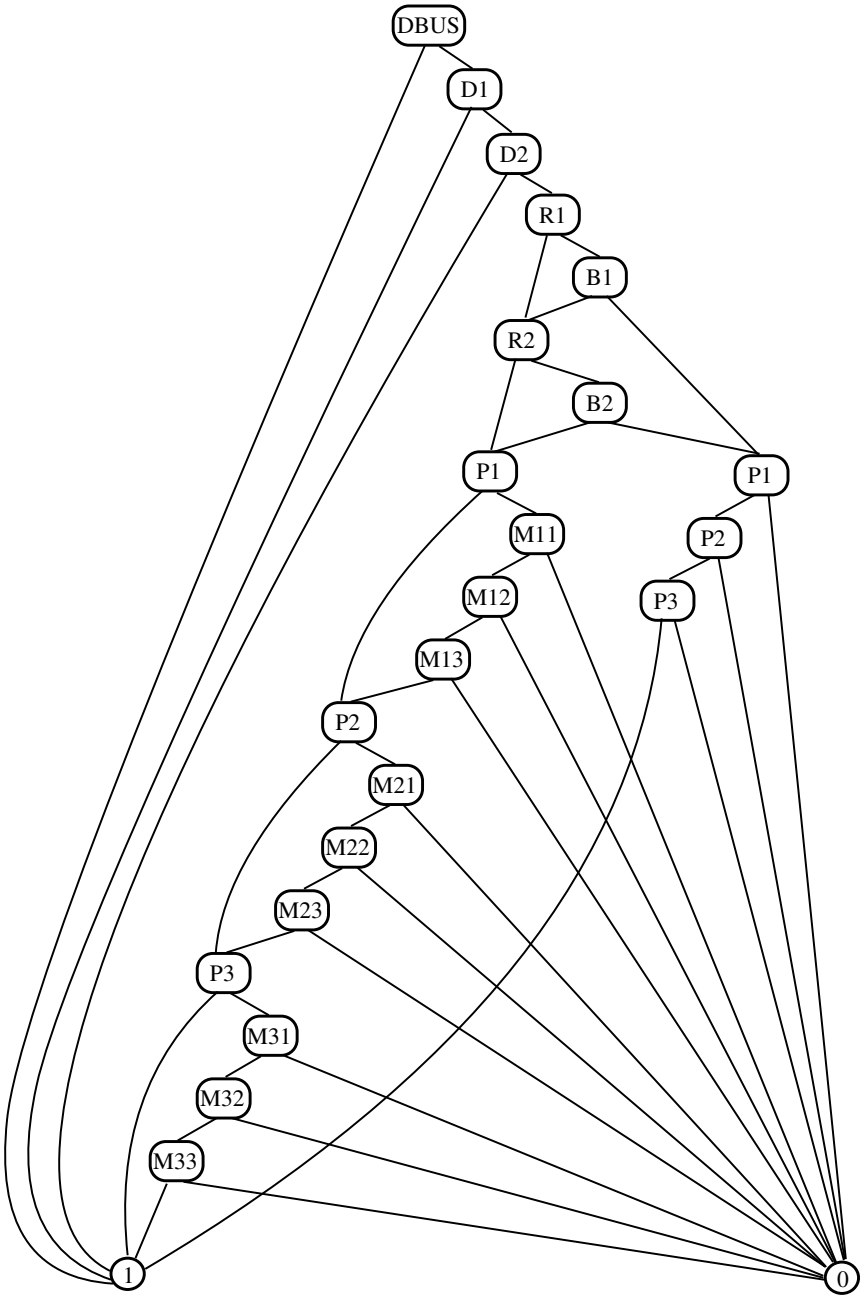

Figure 2.11: The BDD obtained from the FT model in Fig. 2.3.

time t	$Pr\{TE, t\}$
1000 h	1.600717E-03
2000 h	3.198873E-03
3000 h	4.794473E-03
4000 h	6.387520E-03
5000 h	7.978020E-03
6000 h	9.565979E-03
7000 h	1.115139E-02
8000 h	1.273428E-02
9000 h	1.431464E-02
10000 h	1.589248E-02

Table 2.4: Unreliability values for the Multiproc system.

Using eq. 2.17, we can derive the MCSs from the BDD in Fig. 2.11, corresponding to the FT model in Fig. 2.3. The obtained MCSs, sorted by their order (number of BEs inside a MCS), follow:

$1 : [D1]$
$2 : [D2]$
$3 : [DBUS]$
$4 : [P1P2P3]$
$5 : [B1B2M11M12M13P2P3]$
$6 : [B1R2M11M12M13P2P3]$
$7 : [R1B2M11M12M13P2P3]$
$8 : [R1R2M11M12M13P2P3]$
$9 : [B1B2P1M21M22M23P3]$
$10 : [B1B2P1P2M31M32M33]$
$11 : [B1R2P1M21M22M23P3]$
$12 : [B1R2P1P2M31M32M33]$
$13 : [R1B2P1M21M22M23P3]$
$14 : [R1B2P1P2M31M32M33]$
$15 : [R1R2P1M21M22M23P3]$
$16 : [R1R2P1P2M31M32M33]$
$17 : [B1B2M11M12M13M21M22M23P3]$
$18 : [B1R2M11M12M13M21M22M23P3]$
$19 : [R1B2M11M12M13M21M22M23P3]$
$20 : [R1R2M11M12M13M21M22M23P3]$
$21 : [B1B2M11M12M13P2M31M32M33]$
$22 : [B1R2M11M12M13P2M31M32M33]$
$23 : [R1B2M11M12M13P2M31M32M33]$
$24 : [R1R2M11M12M13P2M31M32M33]$
$25 : [B1B2P1M21M22M23M31M32M33]$
$26 : [B1R2P1M21M22M23M31M32M33]$

$27 : [R1B2P1M21M22M23M31M32M33]$
$28 : [R1R2P1M21M22M23M31M32M33]$
$29 : [B1B2M11M12M13M21M22M23M31M32M33]$
$30 : [B1R2M11M12M13M21M22M23M31M32M33]$
$31 : [R1B2M11M12M13M21M22M23M31M32M33]$
$32 : [R1R2M11M12M13M21M22M23M31M32M33]$

2.5 Module based FT analysis

Despite the efficiency of the use of BDDs, we may have to deal with FTs with a huge amount of events and gates. Moreover, when we are analyzing large systems, the qualitative analysis may return an high number of MCSs, and some of them may involve a lot of components. So, interpreting the MCSs relevance may be unpractical for the Reliability engineer.

A solution to these problems is solving FT models using the method of decomposition and aggregation. This means dividing the system (FT) into smaller subsystems (subtrees) to be analyzed in isolation, and synthesizing the results into an higher level FT to produce the whole system solution.

Using this approach, the computational cost is reduced by dealing with subtrees instead of the whole FT model; at the same time, the MCSs returned by the analysis of subtrees, concern subsystems instead of the whole system, so they can be interpreted by the Reliability engineer in an easier way.

2.5.1 Definition of module

The decomposition of the FT must be performed in such a way to obtain the correct final results when synthesizing the results obtained on the subtrees analyzed in isolation. To achieve this purpose, the FT must be decomposed in *modules* [2, 44]: a module is a subtree which is independent from any other subtree. The definition of module is provided in [44]:

Definition 1 *A **module** is a subtree whose terminal events (BEs) do no occur elsewhere in the FT.*

Formally, the event m is the root of a module iff for any other event e, either $e \in \circ m$, or $\circ m \cap \circ e = \emptyset$ [44]. The module whose root is m is the subtree $\widehat{m}$.

According to the definition of module, the whole FT ($\widehat{TE}$) and the BEs are modules. However, we do not classify the BEs as modules, since the detachment and the analysis in isolation of a BE does not allow any benefit in the FT analysis. A module may contain inner modules; a module rooted in a IE or in the TE, is *minimum* if it does not contain any other module (excluding the BEs).

2.5.2 Modules detection algorithm

In [44], an algorithm to detect the modules of a FT model, is presented. This algorithm consists of two depth-first left-most visit of the FT, starting from the TE and assuming the opposite orientation of the arcs with respect to the circuit logic orientation. Performing this kind of visit, the TE and each IE is visited at least twice: when descending from the (first) gate the event is input of, and when going back from the gate the event is output of. A BE is visited at least once, when descending from the gate the BE is input of.

Three variables are associated with every $e \in \mathcal{E}$: t_1, t_2, t_l; they indicate at which step of the visit, the event e has been visited for the first time, the second time and the last time, respectively. We associate other two variables with every $e' \in \mathcal{IE}$: min_{t_1} and max_{t_l}; they must be set to the minimum value of the variables t_1 in the graph underlying e', and to the maximum values of the variables t_l in the graph underlying e', respectively.

Before beginning the first visit, a counter is set to 0. During the first depth-first left-most visit of the FT, each time an event is visited, such counter is incremented by 1. The first time we visit the event e, we set the variables t_1 and t_l associated with e, to the current value of the counter. When we visit e for the second time, we set the variables t_2 and t_l to the current value of the counter. Every time we visit again e, we set the variable t_l to the current value of the counter. When we visit an event e for the second or successive time, we immediately go back to the gate such that e is input of, without visiting again the underlying subgraph.

In the second depth-first left-most visit of the FT, we compute for each IE the value of the variables min_{t_1} and max_{t_l}. Then, we can detect the modules in this way: an event $e' \in \mathcal{IE}$ is the root of a module if the following condition involving its variables t_1, t_2, min_{t_1} and max_{t_l}, holds:

$$(min_{t_1} > t_1) \wedge (max_{t_l} < t_2)$$

2.5.3 Modularization

Due to its independence, a module which is rooted in a IE, can be detached from the FT and analyzed in isolation, without affecting the correct analysis of the rest of the FT.

If we assume that the minimal modules are analyzed in isolation, then the FT analysis by *Modularization*, i. e. by exploiting modules, follows these steps:

1. Modules detection

2. Modules classification: for each module, we verify if it is a minimum module.

3. If the current FT contains the minimum module $\widehat{TE}$, then go to the step 8, else go to the step 4.

4. Decomposition: each minimum module is detached from the FT.
5. Minimum modules analysis: the detached minimum modules are analyzed.
6. Aggregation: each detached minimum module is replaced in the FT by a BE.
7. Go to step 1.
8. Analysis of the *reduced* FT.

In the aggregation step, the qualitative or quantitative analysis results obtained on the module are assigned to the BE replacing the module:

- if we are performing the quantitative analysis at time t, such BE has not an associated probability distribution, but a constant probability equal to the probability of occurrence of the module computed at time t;
- if we are performing the qualitative analysis, we assign to such BE the MCSs of the module it replaces.

The *reduced* FT mentioned in the step 4, is the version of the FT obtained by replacing the minimum modules, such that it contains a unique minimum module coincident with the FT. The reduced FT does not require a further modularization, so it can be entirely analyzed; the analysis of the reduced FT provides the results for the whole system.

During the analysis by modularization of a FT, it may happen that a minimum module contains a low number of events such that its analysis is not enough computationally expensive to justify the cost of the decomposition, analysis and aggregation step of such module. To avoid this inconvenience, we can add a further condition to be verified in order to detect if an event $e' \in \mathcal{IE}$ is the root of a module (section 2.5.1): $|\circ e'| \geq k$, where k indicates the minimum number of events that a subtree must contain in order to be a module.

2.5.4 Running example

Instead of analyzing entirely the FT model of the Multiproc system (Fig. 2.3), we can perform its analysis by modularization (section 2.5). In this section, we limit our attention to the module detection (section 2.5.2).

Fig. 2.12 shows all the modules present in the FT model of the Multiproc system; they are:
$\widehat{TE}$, $\widehat{DA}$, $\widehat{MS}$, $\widehat{CM}$, $\widehat{MM1}$, $\widehat{MM2}$, $\widehat{MM3}$, $\widehat{SM}$, $\widehat{BR1}$, $\widehat{BR2}$.
Each module is graphically indicated by a dashed line around it. The subtrees $\widehat{PU1}$, $\widehat{PU2}$, $\widehat{PU3}$, $\widehat{MEM1}$, $\widehat{MEM2}$, $\widehat{MEM3}$ are not classified as modules because they share the common subtree $\widehat{SM}$. The minimal module in the FT are:
$\widehat{MS}$, $\widehat{MM1}$, $\widehat{MM2}$, $\widehat{MM3}$, $\widehat{BR1}$, $\widehat{BR2}$.
All of them contain no inner modules.

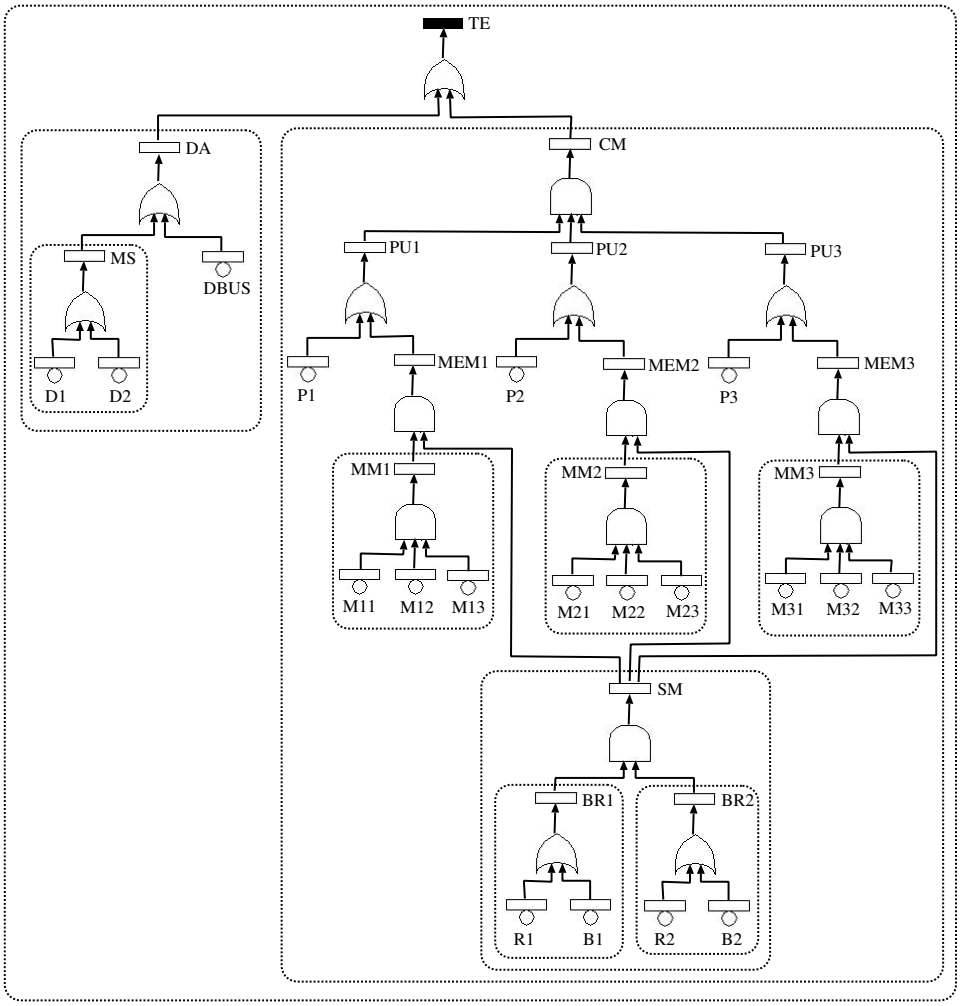

Figure 2.12: The modules in the FT model of the Multiproc system.

Chapter 3

pBDD based PFT Analysis

3.1 Introduction to Parametric Fault Trees

One of the ways to improve the Reliability of a system, consists of replicating critical components or subsystems; sometimes, due to the redundancy of the system, the FT model contains several similar subtrees concerning the replicated parts in the system; moreover, the failure rates of the BEs inside such subtrees, may be the same. So, the construction of such FT becomes quite unpractical for the Reliability engineer, since he has to draw several identical (large) subtrees. Besides this drawback, the analysis of the FT of a redundant system, leads to repeat some steps, due to the presence of symmetric subtrees, and their analysis returns identical results.

For this reasons, a particular extension of the FT formalism called *Parametric Fault Tree* (PFT) [11, 51] was proposed with the purpose of providing the compact modelling of the redundant parts of the system. An example of PFT model is shown in Fig. 3.1.

Using the PFT formalism, the subtrees with the same structure and the same failure rates (replicated subtrees) can be folded in a single parametric subtree, while their identity is maintained through a parameter: each replicated subtree folded in a parametric subtree is identified by a specific value of the parameter. Such value ranges over the *type* of the parameter; for instance, the type of the parameter i is the set $C_i = \{1, \ldots, n\}$, with $n \geq 1$.

The model design is simplified using the PFT formalism, since the model designer can fold subtrees with the same structure in a single parametric subtree, avoiding to draw (large) identical subtrees, and consequently reducing the number of elements in the model. Such reduction is proportional to the level of redundancy in the system.

The PFT formalism extends the FT formalism; besides the parameters and their types, the PFT formalism introduces other new primitives, with respect to the FT formalism. In the construction of the PFT model, each time replicated subsystems are encountered, a parametric subtree must be drawn in the model. The root of a parametric subtree is a *Replicator Event* (RE): a RE graphically appears as a dotted

rectangle, and a parameter is declared together with its type, inside the RE. Such parameter is associated also with the events in the parametric subtree if an instance of them is present in each of the replicated subtrees represented in compact form by the parametric subtree. The cardinality of the type of the parameter declared in the RE, indicates the number of replicated subtrees folded in the parametric subtree, while each element of the type identifies a distinct replicated subtree.

Parametric subtrees may be nested, so a parametric subtree may contain inner parametric subtrees. In this case, the RE which is the root of an inner subtree, may contain several parameters: the parameters declared in the REs which are the roots of the outer subtrees, and the parameter declared in the RE which is the root of the inner parametric subtree. By convention, the set of the parameters associated with an event, is ordered by the increasing depth of the REs where the parameters are declared in. So, if the set of parameters of the RE e are $\{i_1, \ldots, i_m\}$, i_m is the parameter declared in e.

In general, if a RE has several parameters, the relative parametric subtree folds as many replicated subtrees as the possible combinations of values of the parameters.

Another new primitive introduced in the PFT formalism is the *Basic Replicator Event* (BRE); a BRE allows to fold several BEs with the same failure rate and connected to the same gate. To this aim, a parameter is declared in a BRE which may have other parameters if it is nested in one or more parametric subtrees.

3.2 PFT formalism definition

The PFT formalism can be defined by the tuple
$\mathcal{PFT} = (\mathcal{E}, \mathcal{G}, \mathcal{A}, \mathcal{P}, \mathcal{T}, \mathcal{BG}, \gamma, \tau, \theta, \delta, \omega, \lambda, \phi)$
where

- $\mathcal{E} = \mathcal{BE} \cup \mathcal{IE} \cup \{TE\} \cup \mathcal{RE} \cup \mathcal{BRE}$ is the set of the events in the PFT; it is the union of the following sets:
  - $\mathcal{BE}$ is the set of the BEs;
  - $\mathcal{IE}$ is the set of the IEs;
  - {TE} is the set composed by the unique TE;
  - $\mathcal{RE}$ is the set of the REs;
  - $\mathcal{BRE}$ is the set of the BREs.
- $\mathcal{A} \subseteq (\mathcal{E} \times \mathcal{G}) \bigcup (\mathcal{G} \times \mathcal{E})$ is the set of the arcs according to the logic circuit orientation.
- $\mathcal{BG} = \{AND, OR\}$ is the set of Boolean gate types.
- $\gamma : \mathcal{G} \to \mathcal{BG}$ is the function assigning to each gate its type.

- $\mathcal{P}$ is the set of parameters.
- $\mathcal{T}$ is the set of types.
- $\tau : \mathcal{P} \rightarrow \mathcal{T}$ is the function assigning to each parameter the corresponding type.
- $\theta : \{\mathcal{E} - \{TE\}\} \rightarrow \bigotimes\{\mathcal{P} \cup \epsilon\}$ is the function returning the set of parameters associated with an event.
- Given $e \in \mathcal{E}$, $\circ e = \{e' \in \mathcal{E} : \exists[e' \rightarrow e]\}$.
  In a PFT, the following property must hold:
  $\forall e \in \mathcal{E}, e \in \circ TE$
- Given $e \in \mathcal{E}$, $\widehat{e}$ is composed by any $[b \rightarrow e] : b \in \mathcal{BE} \cup \mathcal{BRE}$. $\widehat{e}$ indicates the subtree rooted in e.
- $\delta : \{\mathcal{RE} \cup \mathcal{BRE}\} \rightarrow \mathcal{P}$ is the function returning the parameter declared in a RE or in a BRE. The following properties must hold:
  - $\forall e_1, e_2 \in \{\mathcal{RE} \cup \mathcal{BRE}\} : e_1 \neq e_2, \delta(e_1) \cap \delta(e_2) = \emptyset$
  - $\forall e \in \mathcal{RE}, \forall p \in \delta(e), \exists e' : p \in \theta(e') \wedge e' \in \circ e$
  - Given $e \in \mathcal{RE}$, $\delta(e) = \{p\}$, $e' \in \circ e$, $p \in \theta(e')$, we have that $\forall e'' \in [e' \rightarrow e], p \in \theta(e'')$

  Given the event $e \in \mathcal{E}$ such that $\theta(e) \neq \emptyset$ and $\theta(e) = \{p_1, \ldots p_m\}$, with $m \geq 1$, we indicate such event in the PFT model as $e(p_1, \ldots, p_m)$. If $e \in \{\mathcal{RE} \cup \mathcal{BRE}\}$, then $\delta(e) = \{p_m\}$.
- $\omega : \{\mathcal{E} - \{TE\}\} \rightarrow \mathbb{N}$ is the function returning the multiplicity of an event:
  - $\forall e \in \{\mathcal{E} \cup \mathcal{IE} \cup \{TE\}\}, \omega(e) = 1$
  - $\forall e \in \{\mathcal{RE} \cup \mathcal{BRE}\}, \omega(e) = |\tau(\delta(e))|$
- Given $g \in \mathcal{G}$,
  $\bullet g = \{e \in \mathcal{E} : \exists(e, g) \in \mathcal{A}\}$ is the set of input events of g;
  $g\bullet = \{e \in \mathcal{E} : \exists(g, e) \in \mathcal{A}\}$ is the output event of g.
  Given $e \in \mathcal{E}$,
  $\bullet e = \{g \in \mathcal{G} : \exists(g, e) \in \mathcal{A}\}$ is the gate having e as output event;
  $e\bullet = \{g \in \mathcal{G} : \exists(e, g) \in \mathcal{A}\}$ is the set of gates having e as one of their input events.
- The following conditions about the connection of events with gates, must hold:
  - $\forall g \in \mathcal{G}, \sum_{\forall e \in \bullet g} \omega(e) > 1$
  - $\forall g \in \mathcal{G}, |g \bullet| = 1$

  - $\forall e \in \{\mathcal{BE} \cup \mathcal{BRE}\}, |\bullet e| = 0$
  - $\forall e \in \{\mathcal{E} - \{TE\}\}, |e \bullet| > 0$
  - $\forall e \in \{\mathcal{IE} \cup \mathcal{RE}\}, |\bullet e| = 1$
  - $|\bullet TE| = 1$
  - $|TE \bullet| = 0$

- $\lambda : \{\mathcal{BE} \cup \mathcal{BRE}\} \rightarrow \mathbb{R}^+$ is the function assigning a failure rate to a BE or to a BRE, if we assume its negative exponential distribution.

- $\phi : \mathcal{E} \rightarrow \mathbb{B} = \{true, false\}$ is the function returning the Boolean value of an event (Boolean variable). Given $y \in \{\{TE\} \cup \mathcal{IE} \cup \mathcal{RE}\}$, $g \in \mathcal{G}$, $g\bullet = \{y\}$,

  - if $\gamma(g) = AND$ then

$$\phi(y) = \bigwedge_{\forall e : e \in \bullet g \wedge e \in \{\mathcal{IE} \cup \mathcal{BE}\}} \phi(e) \wedge \bigwedge_{\forall e : e \in \bullet g \wedge e \in \{\mathcal{RE} \cup \mathcal{BRE}\}} \bigwedge_{\forall i \in \tau(\delta(e))} \phi(e(i))$$

  - if $\gamma(g) = OR$ then

$$\phi(y) = \bigvee_{\forall e : e \in \bullet g \wedge e \in \{\mathcal{IE} \cup \mathcal{BE}\}} \phi(e) \vee \bigvee_{\forall e : e \in \bullet g \wedge e \in \{\mathcal{RE} \cup \mathcal{BRE}\}} \bigvee_{\forall i \in \tau(\delta(e))} \phi(e(i))$$

3.2.1 Running example

In this section, we refer to the Multiproc system and to its FT model described in section 2.2.1. Several redundancies and symmetries can be observed in the Multiproc system: we have three processing units composed by the same type and the same number of components: one processor and three internal memories. We have two identical shared memories, and each of them has its own bus in order to be connected to all the processing units. Finally, we have two hard disks.

Such symmetries and redundancies in the system lead to the presence in the FT model, of several subtrees with the same structure; for instance, subtrees $PU1$, $PU2$, $PU3$ are isomorphic and their BEs concern the same types of components (processors and memories). Moreover, the basic events referring to the same class of components (for instance, the processors) have the same failure rate. The subtree rooted in the event SM contemporary belongs to the subtrees $\widehat{PU1}$, $\widehat{PU2}$, $\widehat{PU3}$. Also $\widehat{SM}$ contains symmetries.

The symmetries in the system are reflected first in its FT model (Fig. 2.3), and consequently in the corresponding BDD (Fig. 2.11) where we can note that several subgraphs have the same structure and are composed by variables referring to components of the same type. For instance, the node $R1$ is connected through its outgoing 0-edge to $B1$, while $R2$ is connected through its outgoing 0-edge, to $B2$. The subgraph composed by $P1$, $M11$, $M12$, $M13$ has the same structure of

the subgraphs composed by $P2$, $M21$, $M22$, $M23$, and by $P3$, $M31$, $M32$, $M33$, respectively. Moreover, we have three cascading nodes concerning the processors: $P1$, $P2$, $P3$.

PFT model of the system

Fig. 3.1 shows the PFT model of the Multiproc system; such model is semantically equivalent to the FT model in Fig. 2.3, but the number of nodes is clearly reduced.

In the PFT model, the whole system failure is still represented by TE which is the output of a gate of type OR whose input events are CM and DA. With respect to the FT model, the subtrees $\widehat{PU1}$, $\widehat{PU2}$ and $\widehat{PU3}$ have been folded in the PFT model, in the parametric subtree $\widehat{PU(i)}$ ($PU(i) \in \mathcal{RE}$); the type of the parameter i is $C1 = \{1, 2, 3\}$. The event CM is the output of a gate of type OR having $PU(i)$ as its unique input event.

The parameter i is associated also with the events $P(i)$, $MEM(i)$ and $MM(i)$, in order to express that an instance of such events is present inside each instance of the parametric subtree $\widehat{PU(i)}$. The BEs modelling the failure of the internal memories of a processing unit, have been folded in the BRE $M(i, k)$, with k of type $C4 = \{1, 2, 3\}$. This expresses that each instance of processing unit (identified by i) contains a set of internal memories whose cardinality is $|C4|$.

In the PFT model, the event SM has no parameters since it belongs to all the instances of the parametric subtree $\widehat{PU(i)}$. In the FT, the subtrees $\widehat{BR1}$ and $\widehat{BR2}$ have the same structure, so in the PFT model, they have been folded in the parametric subtree $\widehat{BR(j)}$ ($BR(j) \in \mathcal{RE}$) with $\tau(j) = C2 = \{1, 2\}$. $BR(j)$ is the output of a gate of type OR having $R(j)$ and $B(j)$ as input events. This means that in each instance of the parametric subtree $\widehat{BR(j)}$, there is a distinct instance of R and of B.

The BEs $D1$ and $D2$ modelling the failure of the hard disks in the FT model, have been folded in the PFT model, in the BRE $D(h)$, with $\tau(h) = C3 = \{1, 2\}$.

Tab. 3.1 indicates the correspondence between the events in the PFT and the components or subsystems in the Multiproc system.

3.3 PFT analysis

While the quantitative analysis on a PFT model still returns the probability of the TE (Unreliability of the system) at a given time, the qualitative analysis of a PFT returns *Parametric Minimal Cut Sets* (pMCS) [10, 11, 28] instead of the ordinary ones. An ordinary MCS (section 2.3.1) is a minimal set of basic components whose contemporary failed state leads to the whole system failure. In a system with redundancies, we may obtain several MCSs with the same order and composed by components of the same class; a pMCS groups such ordinary MCSs in an equivalence class evidencing only the failure pattern regardless the identity of the replicated components inside the pMCS. This allows a reduction of the failure patterns

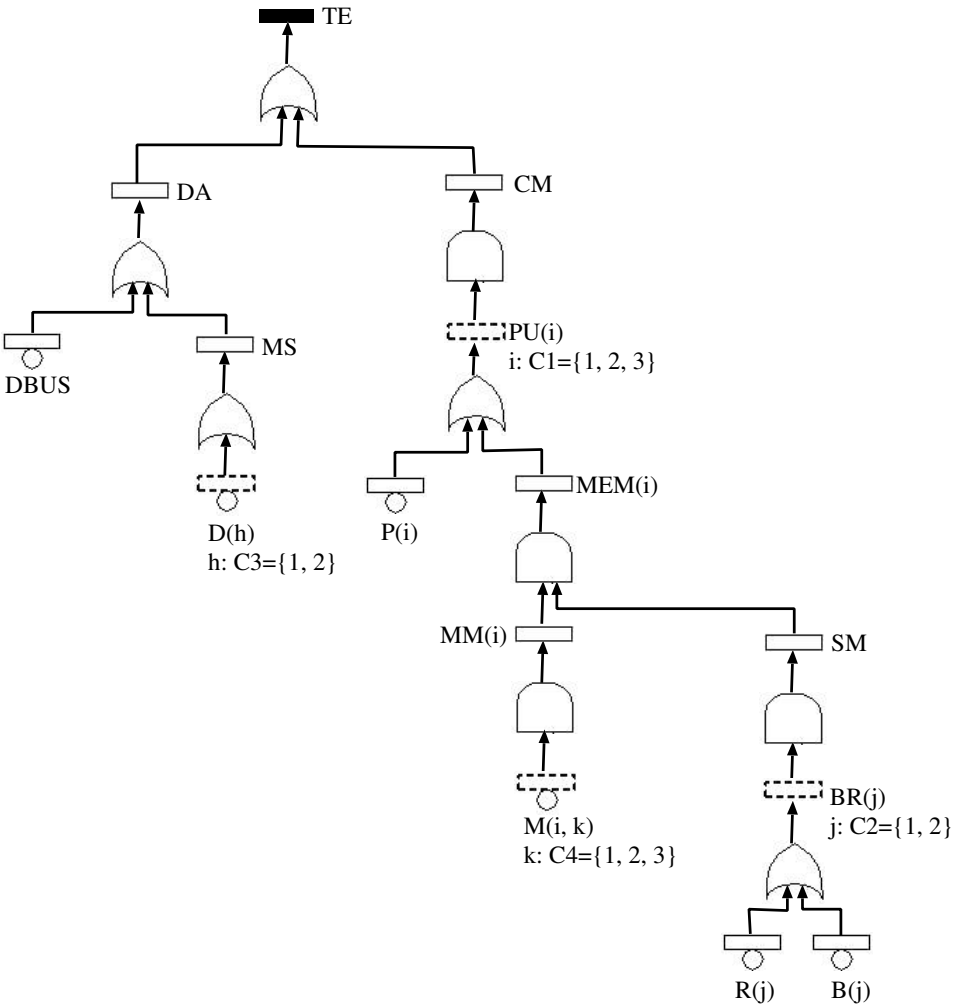

Figure 3.1: The PFT model for the Multiproc system.

Event	Component / Subsystem
DA	Disk Access
$DBUS$	Disk Bus
MS	Mass Storage
$D(h)$	Hard Disks
CM	Computing module
$PU(i)$	Processing Unit i
$P(i)$	Processor of the Processing Unit i
$MEM(i)$	Memory access of the Processing Unit i
$MM(i)$	Internal Memory Module of the Processing Unit i
$M(i,k)$	Internal Memories of the Processing Unit i
SM	Shared Memory access
$BR(j)$	Shared Memory module j
$R(j)$	Shared Memory j
$B(j)$	Memory Bus of the Shared Memory j

Table 3.1: Correspondence between the events and the components or subsystems.

to be examined. Interpreting the PFT as a Boolean formula, a pMCS is a set of solutions of the formula.

As the use of the PFT formalism avoids drawing replicated subtrees, the approach to perform the analysis of a PFT model has to avoid their repeated analysis by performing the analysis of only one replica.

An efficient way to perform the analysis of a FT, is based on the use of BDDs, as shown in chapter 2. In this chapter instead, we adapt the approach based on BDDs, in order to cope with parametric subtrees: BDDs are extended in order to deal with parameters obtaining *Parametric Binary Decision Diagrams* (pBDD) [10, 28].

3.3.1 Related work on PFT analysis

In [21] a method for computing the MCSs of FT models with *event classes* has been proposed: basic nodes can represent subsets of similar basic events. The same event class e can appear as input of different gates with different multiplicities (the notation used is $e[n_i]$, meaning that at least n_i events in class e have occurred, where n_i must of course be less then or equal to the event class cardinality). This could be expressed by the PFT formalism by using a BRE and one or more gates of type $K : N$; on the contrary the full expressiveness of the PFT formalism is not covered by the extension proposed in [21]. It would be interesting to investigate whether the analysis method proposed in [21] could be extended to work on PFT.

3.4 Parametric BDDs

As the use of the PFT formalism allows to compact a FT with symmetric subtrees, a BDD with symmetric subgraphs can be folded in a pBDD. An example of pBDD is shown in Fig. 3.11.

pBDDs are an extension of BDDs, studied in order to cope with parameterization. While in a BDD the nodes are Boolean variables, in a pBDD, a node can be a Boolean variable, a parametric Boolean variable, or a *Parametric Box* (pBox) [10, 28]. A Boolean variable has no parameters, while a parametric Boolean variable of a pBDD has a set of parameters associated with.

A pBox instead, is a new primitive introduced in pBDDs, and is graphically represented as a dashed rectangle including a parametric graph. A pBox has two attributes: a *replication parameter* and a *replication operator*.

The aim of a pBox is representing in a compact way, the BDD obtained by the composition according to a Boolean operator (AND ($\wedge$), OR ($\vee$)), of symmetric BDDs. Such BDDs are represented in a compact way inside the pBox, by a unique parametric graph composed by parametric variables and possibly by inner pBoxes, while their identity is maintained by the *type* of the replication parameter. The replication operator of the pBox is the operator used to compose together the symmetric BDDs. As in the PFT formalism (section 3.2) the type of the replication parameter of a pBox, is the set that the parameter can range over.

A pBox has one *input interface* and two output interfaces: the *1-output interface* and the *0-output interface*. pBox interfaces are graphically indicated as small black circles. The incoming edges[1] of a pBox must point the input interface of the pBox. A pBox can be pointed by one or several incoming edges.

Inside a pBox, all the 1-edges which are not pointing to an internal variable, or to an inner pBox, must be directed to the 1-output interface. All the 0-edges which are not pointing to an internal variable, or to an inner pBox, must be directed to the 0-output interface.

The aim of the 1-output interface and the 0-output interface, is twofold: they represent the constant 1 and the constant 0 respectively, of the symmetric BDDs folded in the parametric graph of the pBox; output interfaces are also the point of connection of the pBox with the underlying nodes.

The outgoing edges of a pBox must be one 1-edge and one 0-edge. The outgoing 1-edge of the pBox is drawn between the 1-output interface of the pBox and an external node (variable or pBox). The outgoing 0-edge of the pBox is drawn between the 0-output interface of the pBox and an external node (variable or pBox). In this way, any node which is external to the pBox can not be connected with nodes inside the pBox, but only with its interfaces.

All the parametric variables inside a pBox whose replication parameter is p, must have p inside their parameter sets.

pBoxes can be nested; in that case, the condition about parameters still holds:

[1] pBDD edges have the tree structure orientation.

a parametric variable inside an inner pBox must have all the parameters associated with the pBoxes containing such parametric variable.

3.4.1 pBDD formal definition

A pBDD is a DAG and can be defined by the tuple
$\mathcal{PBDD} = (\mathcal{N}, \mathcal{D}, r, \mathcal{P}, \mathcal{T}, \mathcal{L}, \tau, \delta, \gamma, \theta, \eta, \psi)$
where

- $\mathcal{N} = \mathcal{V} \cup \mathcal{PB} \cup \mathcal{PV} \cup \mathcal{C}$ is the set of nodes; it is the union of these sets:
  - $\mathcal{V}$ is the set of Boolean variables.
  - $\mathcal{PB}$ is a set of pBoxes.
  - $\mathcal{PV}$ is the set of Parametric Boolean variables.
  - $\mathcal{C} = \{1, 0\}$ is the set composed by the Boolean constants 1 and 0 corresponding to the $true$ and $false$ Boolean value, respectively.
- $r \in \{\mathcal{V} \cup \mathcal{PB}\}$ is the root node.
- $\mathcal{D} \subseteq (\mathcal{N} \times \mathcal{N})$ is the set of edges, according to the the tree structure orientation.
- $\mathcal{L} : \mathcal{D} \rightarrow \{''0'','' 1''\}$ is the function assigning a label to an edge. The label can be either "0" or "1".
- $\mathcal{P}$ is a set of parameters.
- $\mathcal{T}$ is a set of types.
- $\tau : \mathcal{P} \rightarrow \mathcal{T}$ is the function returning the type of a parameter.
- $\delta : \mathcal{PB} \rightarrow \mathcal{P}$ is the function returning the replication parameter of a pBox.
- $\gamma : \mathcal{PB} \rightarrow \{AND, OR\}$ is the function returning the replication operator of a pBox.
- Any $b \in \mathcal{PB}$ can be defined by the tuple (I_i, PG), where
  - I_i is the input interface of b, where the edges pointing b are concentrated.
  - PG is the parametric subgraph inside b. PG can be defined by the tuple $(\mathcal{N}', \mathcal{D}', r')$, where
    - $\mathcal{N}' = \mathcal{PB}' \cup \mathcal{PV}' \cup \mathcal{I}$ is the set of nodes in PG; it is the union of these sets:
      - $\mathcal{PV}' \subseteq \mathcal{PV}$ is the set of Boolean parametric variables in PG.
      - $\mathcal{PB}' \subseteq \mathcal{PB}$ is the set of pBoxes in PG.

        - $\mathcal{I} = \{\underline{1}, \underline{0}\}$, where $\underline{1}$ is the 1-output interface of b, and $\underline{0}$ is the 0-output interface of b.
    - $r' \in \mathcal{PV}' \cup \mathcal{PB}'$ is the root node of PG.
    - $\mathcal{D}' \subseteq (\mathcal{N}' \times \mathcal{N}')$ is the set of edges in PG.

- $\forall b, b' \in \mathcal{PB} : b \neq b', (b \cap b' = \emptyset) \vee (b \subset b') \vee (b \supset b')$
- Given $x \in \mathcal{N}, \eta(x) = \{b \in \mathcal{PB} : x \in b.\mathcal{N}'\}$
- $\theta : \mathcal{PV} \to \bigotimes \mathcal{P}$ is the function returning the set of parameters of a Boolean parametric variable.
  - $\forall x \in \mathcal{PV}, \theta(x) = \bigcup_{\forall b \in PB : b \in \eta(x)} \delta(b)$
- Given $x \in \mathcal{N}$, $\psi(x) = |\eta(x)|$ is the function returning the nesting level of a node; this means the number of pBoxes containing that node.
  $\forall x \in \mathcal{V}, \psi(x) = 0$
- $\forall (x, y) \in \mathcal{D}, (\psi(x) = \psi(y)) \wedge (\eta(x) = \eta(y))$
- Given $x \in \mathcal{N}$,
  - $\bullet x = \{x' \in \mathcal{N} : \exists (x', x) \in \mathcal{D}\}$ is the set of parent nodes of x.
  - $x\bullet = \{x' \in \mathcal{N} : \exists (x, x') \in \mathcal{D}\}$ is the set of descending nodes of x.
  - $x\bullet_1 = \{x' \in \mathcal{N} : \exists (x, x') \in \mathcal{D} \wedge \mathcal{L}(x, x') ='' 1''\}$
  - $x\bullet_0 = \{x' \in \mathcal{N} : \exists (x, x') \in \mathcal{D} \wedge \mathcal{L}(x, x') ='' 0''\}$
  - The following conditions must hold:
    - $\forall y \in \mathcal{N} - \{\{r\} \cup \bigcup_{\forall b \in \mathcal{PB}} b.PG.r'\}, |\bullet y| \geq 1$
    - $|\bullet r| = 0$
    - $\forall b \in \mathcal{PB}, |\bullet b.PG.r'| = 0$
    - $\forall y \in \mathcal{C}, |\bullet y| \geq 1$
    - $\forall b \in \mathcal{PB}, \forall y \in b.PG.\mathcal{I}, |\bullet y| \geq 1$
    - $\forall y \in \mathcal{N} - \{\mathcal{C} \cup \bigcup_{\forall b \in \mathcal{PB}} b.PG.\mathcal{I}\}, |y \bullet| = 2$
    - $\forall y \in \mathcal{C}, |y \bullet| = 0$
    - $\forall b \in \mathcal{PB} \forall y \in b.PG.\mathcal{I}, |y \bullet| = 0$
    - $\forall y \in \mathcal{N} - \{\mathcal{C} \cup \bigcup_{\forall b \in \mathcal{PB}} b.PG.\mathcal{I}\}, |y \bullet_1| = 1\}$
    - $\forall y \in \mathcal{N} - \{\mathcal{C} \cup \bigcup_{\forall b \in \mathcal{PB}} b.PG.\mathcal{I}\}, |y \bullet_0| = 1\}$
    - $\forall y \in \mathcal{N} - \{\mathcal{C} \cup \bigcup_{\forall b \in \mathcal{PB}} b.PG.\mathcal{I}\}, y\bullet = y \bullet_1 \cup y\bullet_0$
    - $\forall y \in \{\mathcal{C} \cup \bigcup_{\forall b \in \mathcal{PB}} b.PG.\mathcal{I}\}, |y \bullet_1| = 0$
    - $\forall y \in \{\mathcal{C} \cup \bigcup_{\forall b \in \mathcal{PB}} b.PG.\mathcal{I}\}, |y \bullet_0| = 0$
- According to the tree structure orientation of the pBDD edges ($\mathcal{D}$), we have a *path* between $x_1 \in \mathcal{N}$ and $x_k \in \mathcal{N}$ if the following conditions hold:

- $\eta(x_1) = \eta(x_k)$
- $\exists x_1, x_2, \ldots, x_k \in \mathcal{N} : \forall i \in \{2, \ldots, k-1\}, \eta(x_i) = \eta(x_1) = \eta(x_k) \wedge \forall i \in \{1, \ldots, k-1\}, \exists (k_i, k_{i+1}) \in \mathcal{D}$

A path between the nodes x_1 and x_k is indicated by the expression $[x_1 \rightarrow x_k]$ and includes all the nodes and the edges along that path:

$$[x_1 \rightarrow x_k] = [x_1, (x_1, x_2), x_2, (x_2, x_3), \ldots, x_{k-1}, (x_{k-1}, x_k), x_k]$$

- Given $x, x' \in \mathcal{N} : \eta(x) = \eta(x')$, x' is *reachable* from x if a path between x and x' exists.
- Given $x \in \mathcal{N}$, $x\circ = \{x' \in \mathcal{N} : \exists [x \rightarrow x'])\}$.
- Given $x \in \mathcal{N} : \eta(x) = \emptyset$, we indicate the subgraph rooted in x with the expression $\widehat{x}$. $\widehat{x}$ is composed by any $[x \rightarrow x'] : x' \in \mathcal{C}$

3.4.2 Parametric *ite* notation

The ite notation (section 2.4.1) is used to express the structure of a BDD. In order to deal with pBDDs, the ite notation needs to be extended in order to indicate the presence of pBoxes and their interfaces.

The structure of a pBDD can be expressed by using the *Parametric if-then-else* notation ($pite$). Let us consider the pBox b with $\delta(b) = k$, $\tau(k) = K$, $\gamma(b) = AND$, $b.PG = pite(x(k), F_1(k), F_0(k))$, $x \in b.\mathcal{PV}' \subseteq \mathcal{PV}$, $\theta(x) = k$. Then, b is represented in $pite$ notation in this way:

$$[pite(x(k), F_1(k), F_0(k)]_{k:K,AND}$$

If the replication operator of b is instead OR ($\gamma(b) = OR$), the $pite$ representation of b becomes,

$$[pite(x(k), F_1(k), F_0(k)]_{k:K,OR}$$

For example, the $pite$ notation of the pBox PB_j in the pBDD in Fig. 3.11, is $PB_j = [pite(R(j), \underline{1}, pite(B(j), \underline{1}, \underline{0}))]_{j:C2,AND}$ where $\underline{1}$ indicates the 1-output interface, while $\underline{0}$ indicates the 0-output interface.

In general, a pBox is indicated in $pite$ notation by expressing its internal parametric graph inside a couple of square brackets, while the replication parameter of the pBox, together with its replication operator, is expressed just after the close square bracket.

A pBox may contain inner pBoxes; in this case, the expression between square brackets, of the parametric graph of the outer pBox, will contain the name of the inner pBox(es), while their $pite$ notation is expressed in isolation. For example, in the pBDD in Fig. 3.11, the pBox named PB'_i contains the inner pBox named PB_k. The $pite$ notation of PB_k is

$$PB_k = [pite(M(i, k), \underline{1}, \underline{0})]_{k:C4,AND}$$

So, the $pite$ notation of PB'_i is

$$PB'_i = [pite(P(i), \underline{1}, pite(PB_k, \underline{1}, \underline{0}))]_{i:C1,AND}$$

The expressions $\underline{1}$ and $\underline{0}$ in the $pite$ notation of a certain pBox, always refer to the 1-output interface and the 0-output interface of such pBox, respectively.

In order to provide the $pite$ notation relative to a whole pBDD, we have first to express in $pite$ notation, any pBox inside the pBDD. In the case of the pBDD in Fig. 3.11, we have that:
$PB_h = [pite(D(h), \underline{1}, \underline{0})]_{h:C3,OR}$
$PB_j = [pite(R(j), \underline{1}, pite(B(j), \underline{1}, \underline{0}))]_{j:C2,AND}$
$PB_k = [pite(M(i,k), \underline{1}, \underline{0})]_{k:C4,AND}$
$PB'_i = [pite(P(i), \underline{1}, pite(PB_k, \underline{1}, \underline{0}))]_{i:C1,AND}$
$PB''_i = [P(i), \underline{1}, \underline{0}]$

The $pite$ notation for the whole pBDD is

$$pite(DBUS, 1, pite(PB_h, 1, ite(PB_j, ite(PB'_i, 1, 0), ite(PB''_i, 1, 0))))$$

where the expressions 1 and 0 are the terminal nodes (constants) 1 and 0 corresponding to the Boolean values $true$ and $false$, respectively.

3.4.3 pBDD unfolding

In order to clarify the semantic of a pBDD, in this section, we show how it is possible to unfold a pBDD; this means deriving from a pBDD the semantically equivalent BDD.

We suppose to have the pBox PB_k expressed in $pite$ notation as

$$[pite(x(k), F_1(k), F_0(k))]_{k:K,AND}$$

with $\tau(k) = K = \{1, \ldots, n\}$.

Moreover, we suppose that PB_k has no inner pBoxes, and that we have an order for the Boolean variables inside such graph. Then, the graph represented in compact form by PB_k is given by the unfolding of the pBox:

$$\bigwedge_{k=1}^{n} pite(x(k), F_1(k), F_0(k)) \tag{3.1}$$

If instead, $\gamma(PB_k) = OR$, PB_k can be expressed using the $pite$ notation as

$$[pite(x(k), F_1(k), F_0(k))]_{k:K,OR}$$

while the graph represented in compact form by PB_k is given by the unfolding of the pBox:

$$\bigvee_{k=1}^{n} pite(x(k), F_1(k), F_0(k)) \tag{3.2}$$

The operator $\wedge$ in eq. 3.1 is applied according to eq. 2.14 and eq. 2.16; the operator $\vee$ in eq. 3.2 is applied according to eq. 2.13 and eq. 2.15.

In other words, the unfolding of a pBox consists of composing according to a Boolean relation (AND, OR), as many instances of the parametric graph contained in the pBox, as the number of possible values of the replication parameter of the pBox, given by the cardinality of the type of the parameter. For each possible value of the replication parameter, we have an instance of the parametric graph. An instance of the parametric graph corresponds to a graph with the same structure of the parametric graph, where we have instances of the parametric variables according to the value of the replication parameter concerning the parametric graph instance.

An instance of a parametric variable is a variable identified by the name of the parametric variable and a possible value of the parameter. For instance, given the parametric variable $x(k)$, $x1$ is an instance of $x(k)$ given that $k = 1$.

Moreover, in an instance of the parametric graph, the 1-output interface of the pBox is replaced by the constant 1, while the 0-output interface becomes the constant 0. For example, an instance of the pBox

$$PB_j = [pite(R(j), \underline{1}, pite(B(j), \underline{1}, \underline{0}))]_{j:C2,AND}$$

in the pBDD in Fig. 3.11, is

$$pite(R1, 1, pite(B1, 1, 0))$$

given $j = 1$.

If in the order of the parametric variables of the pBDD we have that $x(k) \prec y(k)$, then in the graph resulting from the the unfolding of the pBox PB_k, we have that

- $\forall i \in K, x(k = i) \prec y(k = i)$
- $\forall i, j \in K : i < j, x(i) \prec x(j)$

After the composition of the instances of the parametric graph of the pBox, the resulting graph must be connected with the rest of the pBDD:

- the edge(s) pointing the input interface of the pBox before the unfolding, are redirected toward the root node of the graph resulting from the unfolding of the pBox.
- All the edges pointing to the constant 1 in the graph resulting from the pBox unfolding, are redirected toward the node pointed by the 1-edge of the pBox before the unfolding. All the edges pointing to the constant 0 in the graph resulting from the pBox unfolding, are redirected toward the node pointed by the 0-edge of the pBox before the unfolding.

If we have to unfold a pBox having inner pBoxes, first, any inner pBox must be unfolded, then the outer pBox can be unfolded. If we perform the unfolding of the pBDD in Fig. 3.11, we obtain the BDD in Fig. 2.11.

3.4.4 Related work on BDDs in compact form

In [62] a parametric version of BDD called Linearly Inductive BDD (LIBDD) has been proposed, which resembles the pBDD introduced in this paper. The application area is rather different but the basic idea (which allows to exploit symmetries) is similar. The LIBDD have in general a more complex structure than the pBDD, and contain cycles (because are used to represent linearly inductive Boolean functions) but the *unfolding* semantics behind the two parametric BDD extensions is very similar. The pBDD extension is powerful enough for our purposes and it is amenable to a simpler analysis.

3.5 PFT analysis by means of pBDDs

Given a PFT model, we can derive from it the equivalent pBDD. The pBDD shown in Fig. 3.11 is equivalent to the PFT model in Fig. 3.1.

Observing the pBDD in Fig. 3.11, we can note that the Boolean variables of the pBDD correspond to a BEs having no parameters in the corresponding PFT. A parametric Boolean variable of the pBDD corresponds to a BE whose parameter set is not empty, or to a BRE of the PFT. The parameter set of a parametric variable in the pBDD, is the same parameter set of the BE or BRE that the parametric variable corresponds to.

The replication parameters of the pBoxes inside the pBDD, are the parameters declared in the (B)REs inside the PFT. Such parameters maintain their type when mapped from the PFT to the pBDD.

In this section, we show how it is possible to derive a pBDD from a PFT model. The first step consists of ordering the BEs and the BREs of the PFT. Then, the pBDD relative to the whole PFT, can be built by composing the pBDDs relative to subtrees of the PFT.

After the creation of the complete pBDD, we can perform on it both the quantitative and qualitative analysis.

3.5.1 Restrictions on PFT models

The pBDD based methods for the analysis of PFTs, proposed in this chapter, can be applied if some restrictions concerning the connection of events with gates, and the use of parameters, are respected in the PFT model.

The restrictions on the ways to connect events with gates, are:

- the set of types of gate is limited to $\{AND, OR\}$.
- $\forall e \in \{\mathcal{RE} \cup \mathcal{BRE}\}, |e\bullet| = 1$
- $\forall g \in \mathcal{G}, \forall e \in \{\mathcal{RE} \cup \mathcal{BRE}\} : e \in \bullet g, \bullet g = \{e\}$

This means that a (B)RE can be the input of only one gate, and a gate having a (B)RE as input event, must not have other input events.

The restrictions on the use of the parameters, are:
Given $y \in \mathcal{RE}$, the set U is defined in this way:

$$U = \{e \in \mathcal{E} : e \in \circ y \wedge \nexists y' \in \mathcal{RE} : e \in \circ y' \wedge y' \neq e \wedge y' \neq y \wedge y' \in \circ y\}$$

The set U is the set of the events such that y is reachable from them and there are no other REs along the paths from the element of U and y.

- $\forall x \in U \cap \{\mathcal{BE} \cup \mathcal{IE}\}$, the set of parameters of x can be one of the following sets:
  - $\theta(x) = \theta(y)$
  - $\theta(x) = \theta(y''), y'' \in \mathcal{RE} \wedge x \in \circ y'' \wedge y'' \neq y$
  - $\theta(x) = \emptyset$

  This means that the parameter set of a non replicator event x included in U, must be the same of y, or the same of another RE y'' such that y'' is reachable from x (and from y), or the empty set.

- $\forall x \in U \wedge x \in \{\mathcal{RE} \cup \mathcal{BRE}\}$, the set of parameters of x can be one of the following sets:
  - $\theta(x) = \theta(y) \cup \delta(x)$
  - $\theta(x) = \theta(y'') \cup \delta(x), y'' \in \mathcal{RE} \wedge x \in \circ y'' \wedge y'' \neq y \wedge y'' \neq x$
  - $\theta(x) = \delta(x)$

  This means that the parameter set of a (B)RE x included in U, must be the same of y with the addition of a new parameter ($\delta(x)$), or the same of another RE y'' with the addition of a new parameter ($\delta(x)$), such that y'' is reachable from x (and from y), or the set composed only by $\delta(x)$.

3.5.2 Ordering the PFT events

As we need to sort the BEs of a FT before generating the corresponding BDD, an order for the BEs and the BREs of a PFT, must be set and must be respected during the construction of the pBDD.

The events order influences the final dimensions of the pBDD. Several heuristics were proposed in the literature [15, 72, 54] in order to reduce the size of the BDD derived from a FT. For instance, the heuristic named H7 (section 2.4.3) sorts the BEs by the decreasing number of their fanouts, i.e. the number of gates that a single BE is input of.

This ordering policy may be useful in the construction of the pBDD corresponding to a PFT model, but in a PFT, an event may be the input of several gates, even though it has only one fanout. This is due to the presence of parameters. For instance, in Fig. 3.1, the event SM has one fanout, but actually it is the input of

several gates having $MEM(i)$ as output event, with i ranging over its type $C1$. In other words, $\widehat{SM}$ belongs to several replicated subtrees represented in compact form by one parametric subtree, in this case, the parametric subtree $\widehat{PU(i)}$. So, we need a way to determine, if a subtree is *shared*, i. e. it belongs to several replicated subtrees.

Definition 2 *Given $x \in \mathcal{E} - \{TE\}$, and $y \in \mathcal{RE}$, $\widehat{x}$ is **shared** by any replicated subtree represented in compact form by $\widehat{y}$, if $\delta(y) \notin \theta(x) \wedge x \in \circ y$. Briefly, we say that $\widehat{x}$ is shared by $\widehat{y}$.*

Our approach to sort the BEs together with the BREs of a PFT, is based on this idea: given $\widehat{x}$ shared by $\widehat{y}$, any event $e' \in \widehat{x}$ must appear in the order before any event $e : e \in \widehat{y} \wedge e \notin \widehat{x}$.

To this aim, instead of directly sorting the B(R)Es of the PFT, we group them in event classes and we sort the event classes. An *event class* is a set grouping the B(R)Es with the same set of parameters. We indicate an event class in this way:

$$EC\{p_1, \ldots, p_m\} = \{e \in \mathcal{BE} \cup \mathcal{BRE} : \theta(e) = \{p_1, \ldots, p_m\}\}$$

If the restrictions on the use of parameters in the PFT (defined in section 3.5.1), are respected, the following properties hold:

- $\forall e \in \mathcal{BE} : \theta(e) \neq \emptyset, \exists e' \in \mathcal{RE} : \theta(e) = \theta(e')$
- $\forall e \in \mathcal{BRE}, \nexists e' \in \mathcal{E} : \theta(e) = \theta(e')$

In other words, if the restrictions are respected, for each RE in the PFT we have an event class collecting the BEs having the same parameter set of such RE. We have also an event class for each BRE containing only such BRE. The BEs having no parameters can be grouped in an event class concerning the empty set of parameters: $EC\{\} = \{e \in \mathcal{BE} : \theta(e) = \emptyset\}$. So, each B(R)E can be assigned to a specific event class.

The event class $EC\{p_1, \ldots, p_m\}$ groups the B(R)Es having $\{p_1, \ldots, p_m\}$ as parameter set; such events belong to the subtree $\widehat{y} : y \in \mathcal{RE} \cup \mathcal{BRE} \wedge \theta(y) = \{p_1, \ldots, p_m\}$. We discover if a subtree is shared by several replicated subtrees, by observing the set of parameters of its root event; such set must be the same as the set of parameters of a RE or a BRE corresponding to an event class.

Since there is a correspondence between a (B)RE and an event class, ordering the (B)REs means ordering the event classes. The (B)REs of the PFT can be sorted by visiting the PFT. We perform a depth-first left-most visit of the PFT.

Initially, the order list is empty. When we visit the first (B)RE, such (B)RE becomes the first and unique element of the current order. When we visit the successive (B)RE x, we verify by means of def. 2 if $\widehat{x}$ is shared by some $\widehat{y} : y \in \mathcal{RE}$. If $\widehat{x}$ is shared, we look in the order list, starting from the initial element, for the first element y such that $\widehat{x}$ is shared by $\widehat{y}$, and we place x just before y in the order list. If $\widehat{x}$ is not shared by any subtree, x is appended to the order list.

When the visit of the PFT is complete, we have sorted all the (B)REs of the PFT. The order of the event classes is given by $EC\{\}$ followed by the event classes corresponding to the ordered (B)REs.

Inside each event class, it is possible to perform a further ordering based on some heuristic; for instance, we can sort the event in an event class according to the number of their fanouts.

At this point, we obtain the general ordering of the B(R)Es of the PFT, by the concatenation of the ordered events inside the ordered event classes. If $EC_1 \prec EC_2 \prec \ldots \prec EC_n$ is the ordered list of the event classes, and $e_{i1} \prec e_{i2} \prec \ldots$ is the ordered list of events inside the event class EC_i, the general order of the B(R)Es of the PFT is given by $e_{11} \prec e_{12} \prec \ldots \prec e_{21} \prec e_{22} \prec \ldots \prec e_{n1} \prec e_{n2} \prec \ldots$

3.5.3 The construction of the pBDD

The rules to build a BDD from a FT model, are still valid for the generation of the pBDD corresponding to a PFT, but in the pBDD construction, we have also to consider the presence of parameters and we have to create pBoxes.

Once the B(R)Es of the PFT have been ordered (section 3.5.2), we can create and compose the subgraphs of the pBDD using the same rules used to obtain a BDD from a FT:

- Given $x \in \mathcal{BE} : \theta(x) = \emptyset$ in the PFT, the pBDD corresponding to x is $pite(x, 1, 0)$ with $x \in \mathcal{V}$.
- Given $x \in \mathcal{BE} \cup \mathcal{BRE} : \theta(x) = \{p_1, \ldots, p_m\}, m \geq 1$ (we can indicate such event also as $x(p_1, \ldots, p_m)$) the pBDD corresponding to $x(p_1, \ldots, p_m)$ is $pite(x(p_1, \ldots, p_m), 1, 0)$ with $x(p_1, \ldots, p_m) \in \mathcal{PV}$.
- Given $y \in \mathcal{IE} \cup \mathcal{RE}$ and $g \in \mathcal{G}$ such that $y \in g\bullet$ and $\bullet g = \{x_1, \ldots, x_n\}, (n \geq 2)$ the pBDD equivalent to $\widehat{y}$ is given by the composition of the pBDDs equivalent to $\widehat{x_1}$, $\widehat{x_2}$, $\ldots$, $\widehat{x_n}$. If $\gamma(g) = OR$, such composition can be performed by means of eq. 2.13 and eq. 2.15. If $\gamma(g) = AND$, such composition can be performed by means of eq. 2.14 and eq. 2.16. In both cases, the order of the B(R)Es of the PFT must be respected. Moreover, eq. 2.13, 2.14, 2.15 2.16 refer to the composition of BDDs, so the terms a, b, c in such equations, refer to Boolean variables. If we use these equations to compose pBDDs, the terms a, b, c are still the root nodes of the pBDDs to be composed, but a, b, c can be Boolean variables, parametric Boolean variables or pBoxes.
- Given $y \in \mathcal{IE} \cup \mathcal{RE}$, $g \in \mathcal{G}$ and $x \in \mathcal{RE} \cup \mathcal{BRE}$ such that $y \in g\bullet$, $\bullet g = \{x\}$, $\delta(x) = p_x$ the pBDD D' equivalent to $\widehat{y}$ is given by the pBDD D equivalent to $\widehat{x}$ with the addition of some pBoxes:
  1. – if $\forall u \in D, p_x \in \theta(u)$ then we create one pBox $B = (I_i, PG)$ where I_i is the input interface of B, and $PG = D$ is the internal

parametric graph of B.

- if $\exists u \in D : p_x \notin \theta(u)$ then for each subgraph $\widehat{z} \in D : p_x \in \theta(z) \wedge \forall z' \in \bullet z, p_x \notin \theta(z')$, we create a pBox $B = (I_i, PG)$ where I_i is the input interface of B, and $PG = \widehat{z}$ is the internal parametric graph of B.

In both cases, $\delta(B) = p_x$ (p_x maintains its type $\tau(p_x)$), $\gamma(B) = \gamma(g)$. Due to the ordering policy of the PFT B(R)Es described in section 3.5.2, the following property holds:
$\forall v \in D.\mathcal{PV} : p_x \in \theta(v), \forall v' \in v\circ, p_x \in \theta(v')$
This happens because, every B(R)E u in $\widehat{x}$ such that $p_x \notin \theta(u)$, precedes in the order any B(R)E u' in $\widehat{x}$ such that $p_x \in \theta(u')$.

Then, we have to redirect the edges of PG pointing to the constant 1 and 0:

- Each $(u, 1) \in B.PG.\mathcal{D}' : u \in B.PG.\mathcal{PV}' \wedge 1 \in B.\mathcal{C}$ becomes $(u, \underline{1}) \in B.PG.\mathcal{D}' : u \in B.PG.\mathcal{PV}' \wedge \underline{1} \in B.PG.\mathcal{I}$
- Each $(u, 0) \in B.PG.\mathcal{D}' : u \in B.PG.\mathcal{PV}' \wedge 0 \in B.\mathcal{C}$ becomes $(u, \underline{0}) \in B.PG.\mathcal{D}' : u \in B.PG.\mathcal{PV}' \wedge \underline{0} \in B.PG.\mathcal{I}$

B must be connected to the constant 1 and 0:

- $(B, 1) \in D.\mathcal{D}$ is created.
- $(B, 0) \in D.\mathcal{D}$ is created.

The edges pointing to z before the creation of the pBox B, must point now to the input interface of B:

- each $(u, z) \in D.\mathcal{D} : u \in D.\mathcal{N} \wedge u \in \bullet z$ becomes $(u, B) \in D.\mathcal{D}$.

In this way, we have created D' from D.

2. Given the pBox B in the pBDD D', $B.PG$ may contain *isomorphic* subgraphs and useless nodes (see section 2.4.2). In $B.PG$, isomorphic subgraphs can be merged, while useless nodes can be deleted (section 2.4.2.
3. In the pBDD D' *isomorphic* pBoxes may be present; two pBoxes are isomorphic if their internal parametric graphs are isomorphic. Isomorphic pBoxes can be merged: we maintain only one of the isomorphic pBoxes, and we remove all the other ones. All the edges pointing the removed pBoxes, are redirected toward the input interface of the maintained pBox. After the creation of the pBox, and the possible simplification of its internal parametric graph PG, PG is not modified any more. Moreover, we deal with the pBox as it was a parametric Boolean variable, even though it contains a graph.
4. In the pBDD D' *useless* nodes may be present: $u \in \mathcal{V} \cup \mathcal{PV} \cup \mathcal{PB}$ is a useless node if $u\bullet_1 = u\bullet_0$. A useless node u' can be deleted from the pBDD D' in this way: each $(u, u') \in D'.\mathcal{D}$ becomes $(u, u\bullet_1)$.

The pBDD relative to the whole PFT is the pBDD corresponding to $\widehat{TE}$.

3.5.4 Quantitative analysis on the pBDD

The quantitative analysis of a pBDD returns the probability of the TE (Unreliability of the system) at a given time t. The quantitative analysis of the pBDD requires first the quantitative analysis in isolation of each of the pBoxes. A pBox can be analyzed only when the analysis of all its internal pBoxes has been completed.

When the analysis of a pBox has been completed, the pBox is replaced in the pBDD by a Boolean variable with the same name and the same probability of the pBox.

Analysis of a pBox

The quantitative analysis in isolation of the pBox B such that $\delta(B) = k$ and $\tau(k) = K$, is performed in this way: first, we compute the probability of its internal parametric graph $B.PG = pite(x(k), F_1(k), F_0(k))$ by means of eq. 3.3:

$$\begin{aligned} Pr\{B.PG\} &= Pr\{pite(x(k), F_1(k), F_0(k))\} = \\ &= Pr\{x\} \cdot Pr\{F_1(k)\} + (1 - Pr\{x(k)\}) \cdot Pr\{F_0(k)\} \end{aligned} \quad (3.3)$$

In eq. 3.3, if x replaces an inner pBox B' of B such that its probability has been previously computed, then $Pr\{x\} = Pr\{B'\}$. If instead x refers to a B(R)E of the PFT corresponding to the pBDD, $Pr\{x\} = 1 - e^{\lambda(x) \cdot t}$ where $\lambda(x)$ is the failure rate of the B(R)E x according to a negative exponential distribution.

The probability $Pr\{B.PG\}$ concerns the parametric graph of the pBox B, but it is not the probability of B. In other words, $Pr\{B.PG\}$ is the probability of one instance of such parametric graph. In order to obtain the probability of the pBox B, i. e. the probability of the graph resulting from the unfolding of B (section 3.4.3), we need to apply to $Pr\{B.PG\}$ an operator depending on the replication operator associated with B ($\gamma(B)$):

- if $\gamma(B) = AND$

$$Pr\{B\} = Pr\{B.PG\}^{|\tau(\delta(B))|} \quad (3.4)$$

- if $\gamma(B) = OR$

$$Pr\{B\} = R(Pr\{B.PG\}, |\tau(\delta B)|) \quad (3.5)$$

  where R is the recursive function defined in this way:

$$\begin{aligned} R(p, 1) &= p \\ R(p, i > 1) &= p + (1 - p)R(p, i - 1) \end{aligned} \quad (3.6)$$

All the instances of $B.PG$ for $\delta(B)$ ranging over $\tau(\delta(B))$ have the same probability $Pr\{B.PG\}$. Moreover, any node of an instance of $B.PG$ does not belong to any other instance of $B.PG$.

So, eq. 3.4 can be explained in this way: if the instances of $B.PG$ must be composed according to the AND relation, the probability of the resulting graph

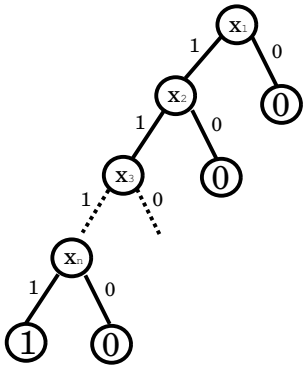

Figure 3.2: BDD with cascading nodes along the 1-edge.

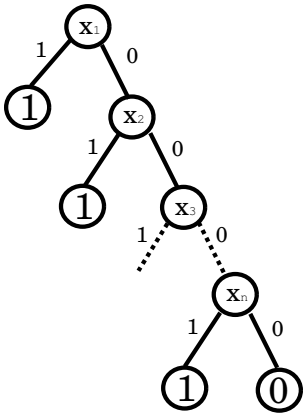

Figure 3.3: BDD with cascading nodes along the 0-edge.

is equivalent to the probability of the BDD shown in Fig. 3.2 and composed by $|\tau(\delta(B))|$ cascading nodes along the 1-edge. Each of these nodes has probability $Pr\{B.PG\}$. The probability of such BDD is given by eq. 3.4.

The meaning of eq. 3.4 is the following: if the instances of $B.PG$ must be composed according to the OR relation, the probability of the resulting graph is equivalent to the probability of the BDD shown in Fig. 3.3 and composed by $|\tau(\delta(B))|$ cascading nodes along the 0-edge. Each of these nodes has probability $Pr\{B.PG\}$. The probability of such BDD is given by eq. 3.5.

Analysis of the complete pBDD

When all the pBoxes in the pBDD have been analyzed and replaced by Boolean variables, we obtain a pBDD containing no pBoxes; the analysis of such pBDD can be performed by means of eq. 3.3 returning the final result, i. e. the probability

of the TE at the given time t.

3.5.5 Qualitative analysis on the pBDD

The qualitative analysis of a pBDD returns the pMCSs (section 3.3) of the system. The qualitative analysis of the pBDD requires first the qualitative analysis in isolation of each of the pBoxes. A pBox can be analyzed when the analysis of all its internal pBoxes has been completed.

When the analysis of a pBox has been completed, the pBox is replaced in the pBDD by a Boolean variable with the same name and the same set of pMCSs of the pBox.

Analysis of a pBox

The qualitative analysis in isolation of the pBox B such that $\delta(B) = k$ and $\tau(k) = K$, is performed in this way: first, we derive the pMCSs of its internal parametric graph $B.PG = pite(x(k), F_1(k), F_0(k))$ by means of eq. 3.7:

$$\begin{aligned} pMCS[B.PG] &= pMCS[pite(x(k), F_1(k), F_0(k))] = \\ &= x \wedge (pMCS[F_1(k)] - pMCS[F_0(k)]) \cup \\ &\quad \cup pMCS[F_0(k)] \end{aligned} \tag{3.7}$$

In eq. 3.7, if x replaces an inner pBox B' of B such that its pMCSs have been previously derived, then $pMCS[x] = pMCS[B']$. If instead x refers to a B(R)E of the PFT corresponding to the pBDD, $pMCS[x] = \{x\}$.

The set $pMCS[B.PG]$ concerns the parametric graph of the pBox B, but it is not the set of pMCSs of B. In other words, $pMCS[B.PG]$ is the set of pMCSs of one instance of such parametric graph. In order to obtain the set of pMCSs of the pBox B, i. e. the pMCSs of the graph resulting from the unfolding of B (section 3.4.3), we need to apply to $pMCS[B.PG]$ an operator depending on the replication operator associated with B ($\gamma(B)$):

- if $\gamma(B) = AND$

$$\begin{aligned} pMCS[B] &= \{pMCS[B.PG]\}^L \stackrel{def.}{=} \\ &= \bigotimes_{\forall k \in \tau(\delta(B))} pMCS[B.PG(k)] \end{aligned} \tag{3.8}$$

  where $\bigotimes$ indicates the Cartesian product.

- if $\gamma(B) = OR$

$$\begin{aligned} pMCS[B] &= \{pMCS[B.PG]\}_L \stackrel{def.}{=} \\ &= \bigcup_{\forall k \in \tau(\delta(B))} pMCS[B.PG(k)] \end{aligned} \tag{3.9}$$

In eq. 3.8 and in eq. 3.9, $pMCS[B.PG(k)]$ is the set of pMCSs of an instance of $B.PG$ according to the value assigned to k.

Analysis of the complete pBDD

When all the pBoxes in the pBDD have been analyzed and replaced by Boolean variables, we obtain a pBDD containing no pBoxes; the analysis of such pBDD can be performed by means of eq. 3.7 returning the final result, i. e. the pMCSs of the system.

3.5.6 Running example

Ordering of the events

Before the construction of the pBDD corresponding to the PFT in Fig. 3.1, we have to sort its B(R)Es. According to the ordering policy proposed in section 3.5.2, we create an event class corresponding to the parameter set of each (B)RE present in the PFT, with the addition of an event class for the BEs having no parameters. The (B)REs in the PFT in Fig. 3.1 are: $PU(i), M(i,k), BR(j), D(h)$; so, we have these event classes: $EC\{i\}, EC\{i,k\}, EC\{j\}, EC\{h\}, EC\{\}$. Then, we assign each B(R)E to the relative event class:
$EC\{i\} = \{P(i)\}$
$EC\{i,k\} = \{M(i,k)\}$
$EC\{j\} = \{R(j), B(j)\}$
$EC\{h\}) = \{D(h)\}$
$EC\{\} = \{DBUS\}$

Now, we have to sort the event classes; this means ordering the (B)REs in the PFT by means of a depth-first left-most visit of the PFT:

1. The first visited (B)RE is $D(h)$, so the order list of the (B)RE is set to $D(h)$.
2. The second visited (B)RE is $PU(i)$; $\widehat{PU(i)}$ is not shared by $\widehat{D(h)}$, so we append $PU(i)$ in the order list:
$$D(h) \prec PU(i)$$
3. The third visited (B)RE is $M(i,k)$; $\widehat{M(i,k)}$ is shared neither by $\widehat{D(h)}$, nor by $\widehat{PU(i)}$, so we append $M(i,k)$ in the order list:
$$D(h) \prec PU(i) \prec M(i,k)$$
4. The fourth and last visited (B)RE is $BR(j)$; $\widehat{BR(j)}$ is shared by $\widehat{PU(i)}$, so we place $BR(j)$ before $PU(i)$ in the order list:
$$D(h) \prec BR(j) \prec PU(i) \prec M(i,k)$$

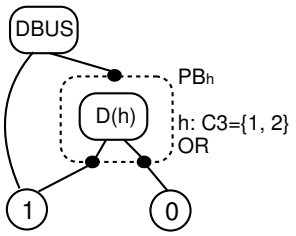

Figure 3.4: The pBDD corresponding to $\widehat{DA}$ in the PFT in Fig. 3.1.

The order of the event classes is given by $EC\{\}$ followed by the event classes corresponding to the ordered (B)REs:

$$EC\{\} \prec EC\{h\} \prec EC\{j\} \prec EC\{i\} \prec EC\{i,k\}$$

In this example, every event class, with the exception of $EC\{j\}$, contains only one B(R)E, so only the elements of $EC\{j\}$ need to be ordered: $R(j) \prec B(j)$.

The general order of the B(R)Es in the PFT, is given by the concatenation of the B(R)Es inside the ordered event classes:

$$DBUS \prec D(h) \prec R(j) \prec B(j) \prec P(i) \prec M(i,k)$$

Building the pBDD

Once the B(R)Es of the PFT have been ordered, we can build the pBDD according to the rules described in section 3.5.3. In this section, we describe some of the steps necessary to build the pBDD corresponding to the PFT model in Fig. 3.1.

We begin with the construction of the pBDD relative to the subtree $\widehat{DA}$. The pBDD relative to the BE $DBUS$ is $pite(DBUS, 1, 0)$; the pBDD relative to the BRE $D(h)$ is $pite(D(h), 1, 0)$. The IE named MS is the output of a gate of type OR having one input event: the BRE $D(h)$. So, we have to create a pBox: the pBDD corresponding to the subtree $\widehat{MS}$ is $pite(PB_h, 1, 0)$ where $PB_h = [D(h), \underline{1}, \underline{0}]_{h:C3,OR}$.

The pBDD relative to BE $DBUS$ is $pite(DBUS, 1, 0)$; the composition of such pBDD with the pBDD relative to the subtree $\widehat{MS}$ by means of eq. 2.13, provides the pBDD corresponding to the subtree $\widehat{DA}$ and shown in Fig. 3.4.

Let us consider now the construction of the pBDD for the subtree $\widehat{CM}$. Fig. 3.5 shows the pBDD for the subtree $\widehat{BR(j)}$. The RE named $BR(j)$ is the input event of a gate of type OR whose output event is SM. The pBDD for the subtree $\widehat{SM}$ is shown in Fig. 3.6.

The pBDD relative to the subtree $\widehat{MM(i)}$ is shown in Fig. 3.7. Since the event $MEM(i)$ is the output of a gate of type AND having $MM(i)$ and SM as input events, in order to generate the pBDD relative to the subtree $\widehat{MEM(i)}$, we need to

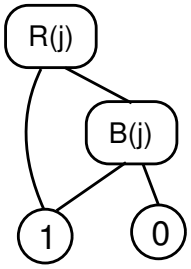

Figure 3.5: The pBDD corresponding to $\widehat{BR(j)}$ in the PFT in Fig. 3.1.

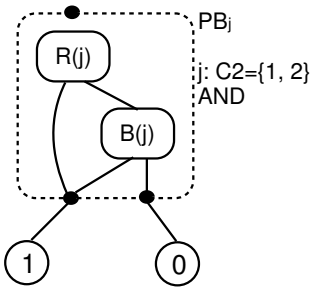

Figure 3.6: The pBDD corresponding to $\widehat{SM}$ in the PFT in Fig. 3.1.

compose the pBDD in Fig. 3.6 with the pBDD in Fig. 3.7, by means of eq. 2.14. The resulting pBDD is shown in Fig. 3.8.

The pBDD for the BE named $P(i)$ is $pite(P(i), 1, 0)$. The event $PU(i)$ is the output of a gate of type OR having $P(i)$ and $MEM(i)$ as input events. So, in order to obtain the pBDD for the subtree $\widehat{PU(i)}$, we have to compose the pBDD in Fig. 3.8 with the pBDD relative to $P(i)$. The result is shown in Fig. 3.9.

The event CM is the output of a gate of type AND having as unique input event the RE named $PU(i)$. To obtain the pBDD for the subtree $\widehat{CM}$, we need to create some pBoxes around several subgraphs of the pBDD in Fig. 3.9. The resulting pBDD is shown in Fig. 3.10.

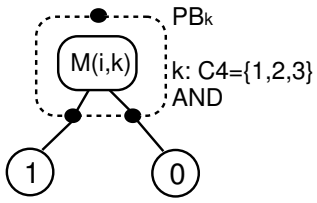

Figure 3.7: The pBDD corresponding to $\widehat{MM(i)}$ in the PFT in Fig. 3.1.

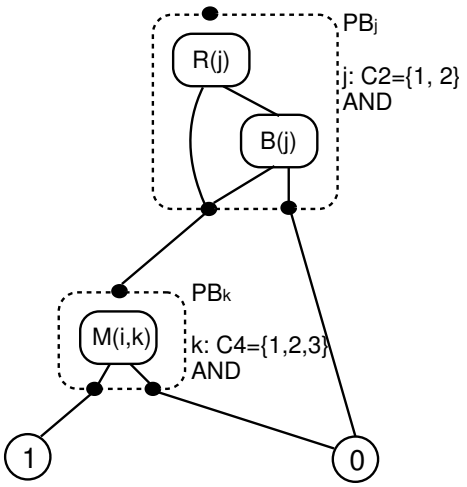

Figure 3.8: The pBDD corresponding to $\widehat{MEM(i)}$ in the PFT in Fig. 3.1.

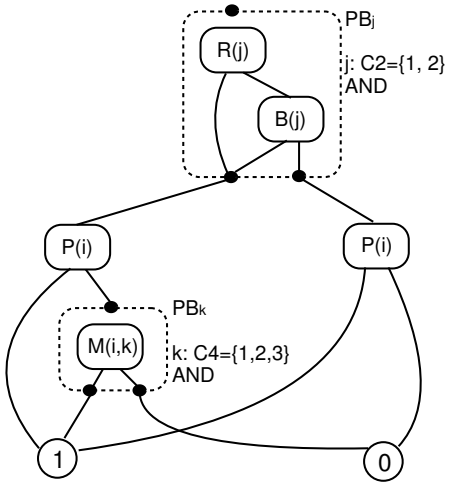

Figure 3.9: The pBDD corresponding to $\widehat{PU(i)}$ in the PFT in Fig. 3.1.

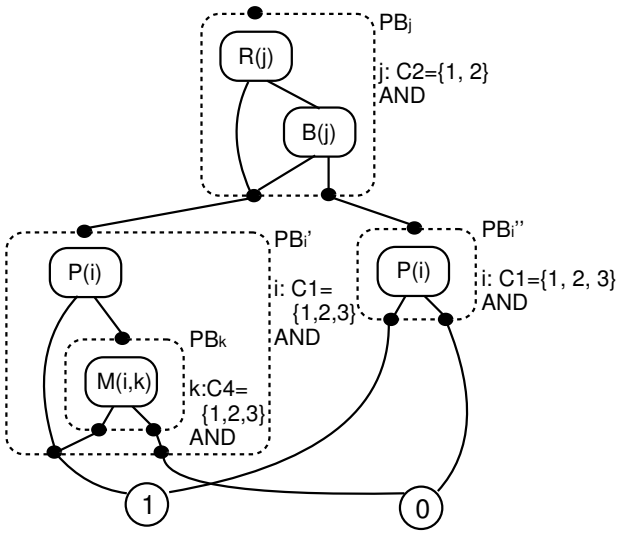

Figure 3.10: The pBDD corresponding to $\widehat{CM}$ in the PFT in Fig. 3.1.

Finally, the pBDD for the whole PFT ($\widehat{TE}$) is obtained by composing the pBDD in Fig. 3.4 with the pBDD in Fig. 3.10, by means of eq. 2.13. The resulting final pBDD is shown in Fig. 3.11.

Quantitative analysis

In this section, we perform the quantitative analysis on the pBDD in Fig. 3.11 for a mission time equal to $10000h$. The failure rates of the BEs and BREs in the PFT in Fig. 3.1 are indicated in Tab. 2.1. Such events are ruled by the negative exponential distribution. These failure rates are used to compute the probabilities of the Boolean variables and parametric Boolean variables in the pBDD in Fig. 3.11.

First, we have to analyze the pBoxes in isolation. We begin with the pBox PB_h with $\delta(PB_h) = h$, $\tau(h) = C3 = \{1, 2\}$, $\gamma(PB_h) = OR$. We have that

$$PB_h.PG = pite(D(h), \underline{1}, \underline{0})$$

We use eq. 3.3 to compute the probability of the parametric graph of PB_h:

$$\begin{aligned} Pr\{PB_h.PG\} &= Pr\{D(h)\} \cdot 1 + (1 - Pr\{D(h)\}) \cdot 0 = Pr\{D(h)\} = \\ &= 1 - e^{(-8.0 \cdot 10^{-7}) \cdot 10000} = 0.007968 \end{aligned}$$

Since $\gamma(PB_h) = OR$ the probability of the pBox PB_h is given by eq. 3.5:

$$\begin{aligned} Pr\{PB_h\} &= R(Pr\{PB_h.PG\}, |C3|) = R(0.007968, 2) = \\ &= 0.007968 + (1 - 0.007968) \cdot R(0.007968, 1) = \\ &= 0.007968 + 0.992032 \cdot 0.007968 = 0.0158727 \end{aligned}$$

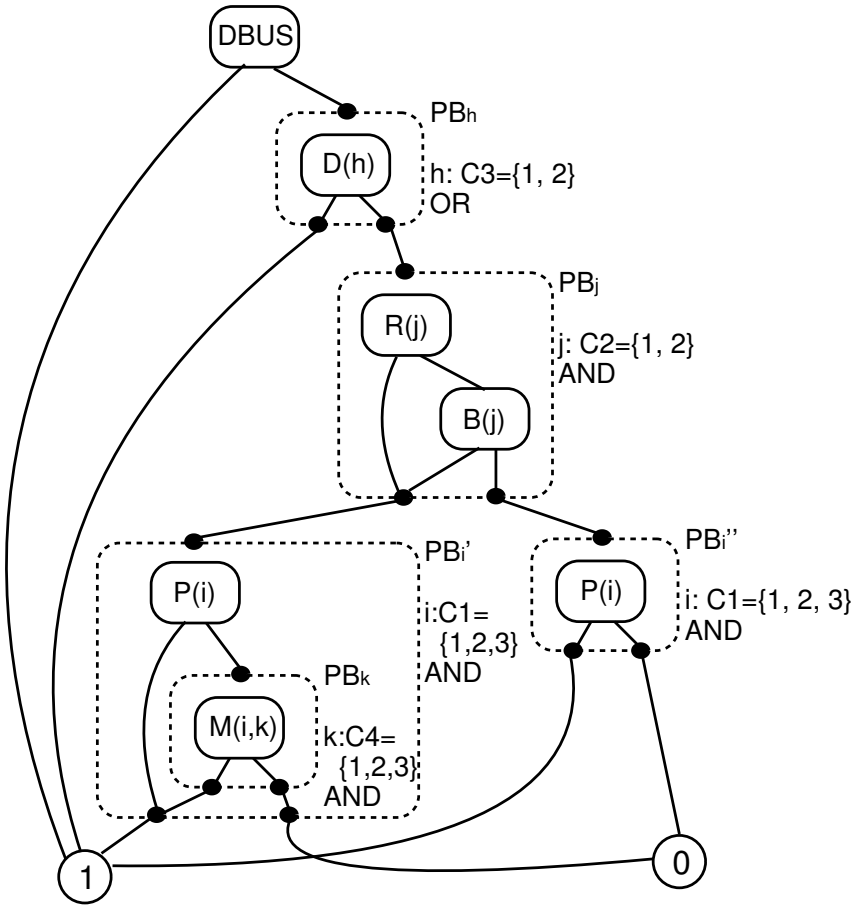

Figure 3.11: The pBDD corresponding to the PFT model in Fig. 3.1.

In the pBDD, we replace the pBox PB_h with a Boolean variable with the same name and the same probability.

We consider now the pBox PB_j with $\delta(PB_j) = j$, $\tau(j) = C2 = \{1, 2\}$ and $\gamma(PB_j) = AND$; the parametric graph of PB_j is

$$PB_j.PG = pite(R(j), \underline{1}, pite(B(j), \underline{1}, \underline{0}))$$

We use eq. 3.3 to compute the probability of $PB_j.PG$:

$$\begin{aligned} Pr\{PB_j.PG\} &= Pr\{R(j)\} \cdot 1 + (1 - Pr\{R(j)\}) \cdot (Pr\{B(j)\} \cdot 1 + \\ &\quad (1 - Pr\{B(j)\}) \cdot 0) = \\ &= 0.000319 \end{aligned}$$

Since $\gamma(PB_j) = AND$, the probability of the pBox PB_j is computed by means of eq. 3.4:

$$Pr\{PB_j\} = Pr\{PB_j\}^{|C2|} = 0.000319^2 = 1.02392 \cdot 10^{-7}$$

In the pBDD, we replace the pBox PB_h with a Boolean variable with the same name and the same probability.

We need to analyze the pBox PB_k before the analysis of $PB_i^{'}$, since PB_k is contained inside $PB_i^{'}$. We have that $\delta(PB_k) = k$, $\tau(k) = C4 = \{1, 2, 3\}$ and $\gamma(PB_k) = AND$. The parametric graph of PB_k is

$$B_k.PG = pite(M(i,k), \underline{1}, \underline{0})$$

. By means of eq. 3.3, we compute the probability of $B_k.PG$:

$$Pr\{PB_k.PG\} = Pr\{M(i,k)\} \cdot 1 + (1 - Pr\{M(i,k)\}) \cdot 0 = 0.000299$$

Since $\gamma(PB_k) = AND$, the probability of PB_k is computed by means of eq. 2.14:

$$Pr\{PB_k\} = Pr\{PB_k.PG\}^{|C4|} = 0.000299^3 = 2.69879 \cdot 10^{-11}$$

Inside the pBox $PB_i^{'}$, we replace the pBox PB_h with a Boolean variable with the same name and the same probability.

Now, we can analyze the pBox $PB_i^{'}$, since after this replacement, it does not contain inner pBoxes. $PB_i^{'}$ has $\delta(PB_i^{'}) = i$, $\tau(i) = C1 = \{1, 2, 3\}$ and $\gamma(PB_i^{'}) = AND$. The parametric graph of $PB_i^{'}$ is now

$$PB_i^{'}.PG = pite(P(i), \underline{1}, pite(PB_k, \underline{1}, \underline{0}))$$

The probability of the parametric graph of $PB_i^{'}$ is computed using eq. 3.3:

$$\begin{aligned} Pr\{PB_i^{'}.PG\} &= Pr\{P(i)\} \cdot 1 + (1 - Pr\{P(i)\}) \cdot (Pr\{PB_k\} \cdot 1 \\ &\quad + (1 - Pr\{PB_k\}) \cdot 0) = \\ &= 1 - e^{5 \cdot 10^{-7} \cdot 10000} \cdot 1 + (1 - (1 - e^{5 \cdot 10^{-7} \cdot 10000})) \cdot \\ &\quad \cdot (2.69879 \cdot 10^{-11} \cdot 1 + (1 - 2.69879 \cdot 10^{-11}) \cdot 0) \\ &= 0.004987 \end{aligned}$$

Since $\gamma(PB_i^{'}) = AND$, the probability of $PB_i^{'}$ is computed using eq. 3.4:

$$Pr\{PB_i^{'}\} = Pr\{PB_i^{'}.PG\}^{|C1|} = 0.004987^3 = 1.24066 \cdot 10^{-7}$$

In the pBDD, we replace the pBox $PB_i^{'}$ with a Boolean variable with the same name and the same probability.

The last pBox to be analyzed in isolation is $PB_i^{"}$. $PB_i^{"}$ has $\delta(PB_i^{"}) = i$, $\tau(i) = C1 = \{1, 2, 3\}$ and $\gamma(PB_i^{"}) = AND$. The parametric graph of $PB_i^{"}$ is

$$B_i^{"}.PG = pite(P(i), \underline{1}, \underline{0})$$

By means of eq. 3.3, we compute the probability of $B_i^{"}.PG$:

$$Pr\{PB_i^{"}.PG\} = Pr\{P(i)\} \cdot 1 + (1 - Pr\{P(i)\}) \cdot 0 = 0.004987$$

Since $\gamma(PB_i^{"}) = AND$, the probability of $PB_i^{"}$ is computed by means of eq. 3.4:

$$Pr\{PB_i^{"}\} = Pr\{PB_i^{"}.PG\}^{|C1|} = 0.004987^3 = 1.24066 \cdot 10^{-11}$$

Inside the pBDD, we replace the pBox $PB_i^{"}$ with a Boolean variable with the same name and the same probability.

Now, in the pBDD, we have no pBoxes; the current pBDD is shown in Fig. 3.12. Its expression in $pite$ notation is

$$G = (pite(DBUS, 1, pite(PB_h, 1, pite(PB_j, pite(PB_i^{'}, 1, 0), pite(PB_i^{''})))))$$

We can compute the probability of the TE at time $t = 10000h$, on the current pBDD, by means of eq. 3.3.

$$\begin{aligned} Pr\{\widehat{PB_j}\} &= Pr\{PB_j\} \cdot Pr\{PB_i^{'}\} + (1 - Pr\{PB_j\}) \cdot Pr\{PB_i^{"}\} = \\ &= 1.02392 \cdot 10^{-7} \cdot 1.24066 \cdot 10^{-7} + \\ &\quad +(1 - 1.02392 \cdot 10^{-7}) \cdot 1.24066 \cdot 10^{-7} = \\ &= 1.24066 \cdot 10^{-7} \end{aligned}$$

$$\begin{aligned} Pr\{\widehat{PB_h}\} &= Pr\{PB_h\} + (1 - Pr\{PB_h\}) \cdot Pr\{\widehat{PB_j}\} = \\ &= 0.0158727 + (1 - 0.0158727) \cdot 1.24066 \cdot 10^{-7} = \\ &= 0.0158728 \end{aligned}$$

$$\begin{aligned} Pr\{G\} &= Pr\{DBUS\} + (1 - Pr\{DBUS\}) \cdot Pr\{\widehat{PB_h}\} = \\ &= 0.000019998 + (1 - 0.000019998) \cdot 0.0158728 = \\ &= 0.0158925 \end{aligned}$$

The last probability value we have obtained, is the probability of the TE at time $t = 10000h$. It corresponds to the value computed at the same time, on the ordinary BDD (Fig. 2.11) corresponding to the FT model in Fig. 2.3 (see section 2.4.6).

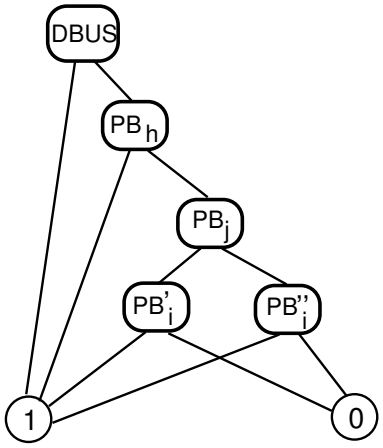

Figure 3.12: The reduced pBDD after the analysis of the pBoxes.

Qualitative analysis

In this section, we perform the qualitative analysis on the pBDD in Fig. 3.11.

First, we have to analyze the pBoxes in isolation. We begin with the pBox PB_h with $\delta(PB_h) = h$, $\tau(h) = C3 = \{1, 2\}$ and $\gamma(PB_h) = OR$. We have that

$$PB_h.PG = pite(D(h), \underline{1}, \underline{0})$$

We use eq. 3.7 to derive the pMCSs of the parametric graph of PB_h:

$$pMCS[PB_h.PG] = \{D(h)\}$$

Since $\gamma(PB_h) = OR$ the pMCSs of the pBox PB_h are given by eq. 3.9:

$$pMCS[PB_h] = \{D(h)\}_{C3}$$

In the pBDD, we replace the pBox PB_h with a Boolean variable with the same name and the same pMCSs.

We consider now the pBox PB_j with $\delta(PB_j) = j$, $\tau(j) = C2 = \{1, 2\}$ and $\gamma(PB_j) = AND$; the parametric graph of PB_j is

$$PB_j.PG = pite(R(j), \underline{1}, pite(B(j), \underline{1}, \underline{0}))$$

We use eq. 3.7 to derive the pMCSs of $PB_j.PG$:

$$pMCS[PB_j.PG] = \{R(j), B(j)\}$$

Since $\gamma(PB_j) = AND$, the pMCSs the pBox PB_j are derived by means of eq. 3.8:

$$pMCS[PB_j] = \{R(j), B(j)\}^{C2}$$

In the pBDD, we replace the pBox PB_h with a Boolean variable with the same name and the same pMCSs.

We need to analyze the pBox PB_k before the analysis of $PB_i^{'}$, since PB_k is contained inside $PB_i^{'}$. We have that $\delta(PB_k) = k$, $\tau(k) = C4 = \{1, 2, 3\}$ and $\gamma(PB_k) = AND$. The parametric graph of PB_k is

$$PB_k.PG = pite(M(i,k), \underline{1}, \underline{0})$$

By means of eq. 3.7, we obtain the pMCSs of $B_k.PG$:

$$pMCS[PB_k.PG] = \{M(i,k)\}$$

Since $\gamma(PB_k) = AND$, the pMCSs of PB_k are derived by means of eq. 3.8:

$$pMCS[PB_k] = \{M(i,k)\}^{C4}$$

Inside the pBox $PB_i^{'}$, we replace the pBox PB_h with a Boolean variable with the same name and the same pMCSs.

Now, we can analyze the pBox $PB_i^{'}$, since after this replacement, it does not contain inner pBoxes. $PB_i^{'}$ has $\delta(PB_i^{'}) = i$, $\tau(i) = C1 = \{1, 2, 3\}$ and $\gamma(PB_i^{'}) = AND$. The parametric graph of $PB_i^{'}$ is now

$$PB_i^{'}.PG = pite(P(i), \underline{1}, pite(PB_k, \underline{1}, \underline{0}))$$

The pMCSs of the parametric graph of $PB_i^{'}$ are obtained using eq. 3.7:

$$pMCS[PB_i^{'}.PG] = \{P(i), \{M(i,k)\}^{C4}\}$$

Since $\gamma(PB_i^{'}) = AND$, the pMCSs of $PB_i^{'}$ are derived using eq. 3.8:

$$pMCS[PB_i^{'}] = \{P(i), \{M(i,k)\}^{C4}\}^{C1}$$

In the pBDD, we replace the pBox $PB_i^{'}$ with a Boolean variable with the same name and the same pMCSs.

The last pBox to be analyzed in isolation is $PB_i^{"}$. $PB_i^{"}$ has $\delta(PB_i^{"}) = i$, $\tau(i) = C1 = \{1, 2, 3\}$ and $\gamma(PB_i^{"}) = AND$. The parametric graph of $PB_i^{"}$ is

$$B_i^{"}.PG = pite(P(i), \underline{1}, \underline{0})$$

By means of eq. 3.7, we obtain the pMCSs of $B_i^{"}.PG$:

$$pMCS[PB_i^{"}.PG] = \{P(i)\}$$

Since $\gamma(PB_i^{"}) = AND$, the pMCSs of $PB_i^{"}$ are derived by means of eq. 2.14:

$$pMCS[PB_i^{"}] = \{P(i)\}^{C1}$$

Inside the pBDD, we replace the pBox $PB_i^{"}$ with a Boolean variable with the same name and the same pMCSs.

Now, in the pBDD, we have no pBoxes; the current pBDD is shown in Fig. 3.12. Its expression in *pite* notation is

$$G = (pite(DBUS, 1, pite(PB_h, 1, pite(PB_j, pite(PB_i^{'}, 1, 0), pite(PB_i^{''})))))$$

We can compute the pMCSs of the whole system on the current pBDD, by means of eq. 3.7.

$$\begin{aligned} pMCS[\widehat{PB(j)}] &= pMCS[PB_j] \wedge (pMCS[PB_i^{'}] - \\ &\quad -pMCS[PB_i^{"}]) \cup pMCS[PB_i^{"}] = \\ &= \{\{R(j), B(j)\}^{C2} \wedge (\{P(i), \{M(i,k)\}^{C4}\}^{C1} - \\ &\quad -\{P(i)\}^{C1}), \{P(i)\}^{C1}\} \end{aligned}$$

$$\begin{aligned} pMCS[\widehat{PB_h}] &= pMCS[PB_h] \cup pMCS[\widehat{PB_j}] = \\ &= \{\{D(h)\}_{C3}, \{R(j), B(j)\}^{C2} \wedge (\{P(i), \{M(i,k)\}^{C4}\}^{C1} - \\ &\quad \{P(i)\}^{C1}), \{P(i)\}^{C1}\} \end{aligned}$$

$$\begin{aligned} pMCS[G] &= pMCS[DBUS] \cup pMCS[\widehat{PB_h}] = \\ &= \{DBUS, \{D(h)\}_{C3}, \{R(j), B(j)\}^{C2} \wedge \\ &\quad \wedge(\{P(i), \{M(i,k)\}^{C4}\}^{C1} - \{P(i)\}^{C1}), \{P(i)\}^{C1}\} \end{aligned}$$

The last set of pMCSs is relative to the whole system. Some of the pMCSs expressed in such form are not consistent with the definition of pMCS provided in section 3.3 because a pMCS must collect a set of ordinary MCSs with the same order and involving BEs concerning the same types of components.

In order to obtain the pMCSs in the correct form, we need to obtain a more explicit notation for the current pMCSs. This can be done by referring to the meaning of the operators defined in eq. 3.8 and in eq. 3.9:

- $DBUS$ is the ordinary MCS number 3 (section 2.4.6).
- $\{D(h)\}_{C3}$ becomes $\bigcup_{\forall h \in C3} D(h)$. Such pMCS collects the ordinary MCSs number 1 and 2 (section 2.4.6).
- $\{R(j), B(j)\}^{C2} \wedge (\{P(i), \{M(i,k)\}^{C4}\}^{C1} - \{P(i)^{C1}\} =$
  $= \{R(j), B(j)\}^{C2} \wedge \{\{P1P2P3, \cup_{\forall(a,b,c)}\{M(a,k)\}^{C4}P(b)P(c),$
  $\cup_{\forall(a,b,c)} \{M(a,k)\}^{C4}\{M(b,k)\}^{C4}P(c),$
  $\{M(1,k)\}^{C4}\{M(2,k)\}^{C4}\{M(3,k)\}^{C4}\} - \{P1P2P3\}\} =$
  $= \{R(j), B(j)\}^{C2} \wedge \{\cup_{\forall(a,b,c)}\{M(a,k)\}^{C4}P(b)P(c),$
  $\cup_{\forall(a,b,c)} \{M(a,k)\}^{C4}\{M(b,k)\}^{C4}P(c),$

$\{M(1,k)\}^{C4}\{M(2,k)\}^{C4}\{M(3,k)\}^{C4}\} =$
$= \{R(j),B(j)\}^{C2} \cup_{\forall(a,b,c)} \{M(a,k)\}^{C4} P(b)P(c),$
$\{R(j),B(j)\}^{C2} \cup_{\forall(a,b,c)} \{M(a,k)\}^{C4}\{M(b,k)\}^{C4} P(c),$
$\{R(j),B(j)\}^{C2}\{M(1,k)\}^{C4}\{M(2,k)\}^{C4}\{M(3,k)\}^{C4}$
where $a, b, c \in C_1$, and $a \neq b$, $a \neq c$, $b \neq c$.

- $\{R(j),B(j)\}^{C2} \cup_{\forall(a,b,c)} \{M(a,k)\}^{C4} P(b)P(c)$ becomes:

$$\bigcup_{\forall a,b,c \in C1: a\neq b, a\neq c, b\neq c, \forall d,e \in C2: d\neq e} R(d)B(e) \wedge \bigwedge_{k=1}^{3} M(a,k) \wedge P(b)P(c)$$

  Such pMCS is in correct form and collects the ordinary MCSs number 6, 7, 11, 12, 13, 14 (section 2.4.6).

$$\bigcup_{\forall a,b,c \in C1: a\neq b, a\neq c, b\neq c, \forall d,e \in C2: d\neq e} R(d)R(e) \wedge \bigwedge_{k=1}^{3} M(a,k) \wedge P(b)P(c)$$

  Such pMCS is in correct form and collects the ordinary MCSs number 8, 15, 16.

$$\bigcup_{\forall a,b,c \in C1: a\neq b, a\neq c, b\neq c, \forall d,e \in C2: d\neq e} B(d)B(e) \wedge \bigwedge_{k=1}^{3} M(a,k) \wedge P(b)P(c)$$

  Such pMCS is in correct form and collects the ordinary MCSs number 5, 9, 10.

- $\{R(j),B(j)\}^{C2} \cup_{\forall(a,b,c)} \{M(a,k)\}^{C4}\{M(b,k)\}^{C4} P(c)$ becomes:

$$\bigcup_{\forall a,b,c \in C1: a\neq b, a\neq c, b\neq c, \forall d,e \in C2: d\neq e} R(d)B(e) \wedge \bigwedge_{k=1}^{3} M(a,k) \wedge \bigwedge_{k=1}^{3} M(b,k) \wedge P(c)$$

  Such pMCS is in correct form and collects the ordinary MCSs number 18, 19, 22, 23, 26, 27.

$$\bigcup_{\forall a,b,c \in C1: a\neq b, a\neq c, b\neq c, \forall d,e \in C2: d\neq e} R(d)R(e) \wedge \bigwedge_{k=1}^{3} M(a,k) \wedge \bigwedge_{k=1}^{3} M(b,k) \wedge P(c)$$

  Such pMCS is in correct form and collects the ordinary MCSs number 20, 24, 28.

$$\bigcup_{\forall a,b,c \in C1: a\neq b, a\neq c, b\neq c, \forall d,e \in C2: d\neq e} B(d)B(e) \wedge \bigwedge_{k=1}^{3} M(a,k) \wedge \bigwedge_{k=1}^{3} M(b,k) \wedge P(c)$$

  Such pMCS is in correct form and collects the ordinary MCSs number 17, 21, 25.

- $\{R(j), B(j)\}^{C2}\{M(1,k)\}^{C4}\{M(2,k)\}^{C4}\{M(3,k)\}^{C4}$ becomes:
  - $\bigcup_{\forall d,e \in C2: d \neq e} R(d)B(e)M11M12M13M23M33M31M32M33$
    Such pMCS is in correct form and collects the ordinary MCSs number 30, 31.
  - $R1R2M11M12M13M23M33M31M32M33$
    Such pMCS is in correct form and is the ordinary MCS number 32.
  - $R1R2M11M12M13M23M33M31M32M33$
    Such pMCS is in correct form and is the ordinary MCS number 31.

- $\{P(i)\}^{C1}$ becomes $P1P2P3$ (ordinary MCS number 4).

3.5.7 Parametric form efficiency

In this section, we compare the FT model in Fig. 2.3 with the PFT model in Fig. 3.1; both models represent the failure mode of the Multiproc system described in section 2.2.1. Moreover, we compare the BDD in Fig. 2.11 with the pBDD in Fig. 3.11, derived from the FT and the PFT model of the Multiproc system, respectively. Such comparison shows the advantages of modelling redundant systems by means of PFT models (and performing their analysis by means of pBDDs), in terms of model size.

If we measure the model size as the number of events, the FT in Fig. 2.3 is composed by 35 events (the TE, 15 IEs, 19 BEs), while the PFT in Fig. 3.1 is composed by 15 events (the TE, 6 IEs, 2 REs, 4 BEs, 2 BREs); by means of the parametric form, the model size is reduced in the second case. If we measure the size of a (p)BDD as the number of nodes corresponding to (parametric) variables, the BDD derived from the FT of the Multiproc system, is composed by 22 nodes, while the pBDD derived from the PFT, is composed by 7 nodes. Thus, the size reduction of the PFT with respect to the FT, is reflected to the pBDD with respect to the BDD.

Such reduction becomes more evident if we increase the number of redundant components or subsystems; such increase produces a combinatorial growth in the number of events in the FT model and in the number of nodes in the corresponding BDD.

The redundant parts of the system are: the processing units, the internal memories of each processing unit, the shared memories with the relative memory buses, and the hard disks.

For the FT model, the number of events is given by the following formula:

$$1 + (3 + H) + (1 + (4 + K) \cdot I + (1 + 3 \cdot J))$$

This formula is the sum of the following addends:

- 1 takes in account the unique TE.

- $(3 + H)$ is the size of the subtree $\widehat{DA}$, where H is the number of disks.

- $(1 + (4 + K) \cdot I + (1 + 3 \cdot J))$ is the size of the subtree $\widehat{CM}$, where K is the number of internal memories, I is the number of processing units, and J is the number of shared memories (and of the corresponding memory buses). The size of the subtree $\widehat{SM}$ is given by $(1 + 3 \cdot J)$.

According to this formula and to the redundancy level, we can compute the size of the FT model of the Multiproc system.

Since in the PFT all the redundant parts are folded in (B)REs, the size of the PFT is given by the formula above assuming that $H = K = I = J = 1$. This means that the increase of the number of redundant parts in the system, does not determine a variation of the size of the model. In order to model in the PFT, an increase of the redundant parts, we have only to increase the cardinality of the types associated with the parameters.

Let us concentrate now on the BDD and the pBDD. The number of nodes of the BDD derived from the FT is given by this formula:

$$1 + H + 2 \cdot J + (K + 1) \cdot I + I$$

where H, J, I and K have the same meaning as in the previous formula. Observing the BDD in Fig. 2.11, the addends of this formula have this meaning:

- 1 takes into account the DBUS node at the root of the BDD.

- H takes into account the subgraph composed by the nodes $D1, D2$; in the pBDD, this subgraph is folded in the pBox PB_h.

- $2 \cdot J$ takes into account the subgraph composed by the nodes $R1, B1$, and the subgraph composed by $R2, B2$; these subgraphs are folded in the pBox PB_j of the pBDD.

- $(K + 1) \cdot I$ takes into account the subgraph composed by the nodes $P1$, $M11$, $M12$, $M13$, the subgraph composed by the nodes $P2$, $M21$, $M22$, $M23$, and the subgraph composed by the nodes $P3$, $M31$, $M32$, $M33$; in the pBDD, these subgraphs are folded in the pBox $PB_i^{'}$ containing the inner pBox PB_k.

- I takes into account the subgraph composed by the cascading nodes $P1$, $P2$, $P3$; this subgraph is folded in the pBox $PB_i^{"}$ of the pBDD.

Due to the folded representation of the system redundancies, the size of the pBDD can be computed by means of this formula assuming that $H = K = I = J = 1$. This means that the pBDD size is not dependent on the redundancy level. Moreover, since the increase of the number of redundant parts does not determine an increase of the PFT size, also the corresponding pBDD does not change its size.

# proc. units	FT size	BDD size	PFT size	pBDD size
3	35	22	15	7
4	42	27	15	7
5	49	32	15	7
6	56	37	15	7
7	63	42	15	7
8	70	47	15	7
9	77	52	15	7
10	84	57	15	7
20	154	107	15	7
30	224	157	15	7
40	294	207	15	7
50	364	257	15	7

Table 3.2: FT, BDD, PFT and pBDD size according to the number of processing units.

Tab. 3.2 shows how the size of the FT and of the BDD changes with respect to an increasing number of processing units, assuming that every processing unit is still composed by one processor and three internal memories, with the presence of two shared memories and two disks.

3.6 PFT modularization

In this section, we extend the algorithm for the detection of modules [44], described in section 2.5, in order to deal with PFT models too. In the case of PFTs, modules can be still detected by means of such algorithm, but we have to add a further condition in order to verify if an event is the root of a module.

The algorithm still consists of two depth-first left-most visit of the PFT: in the first visit, we set the values of the variables t_1, t_2, t_l for each event; in the second visit, we determine the values of the variables min_{t_1} and max_{t_l} for each IE and RE of the PFT (see section 2.5.2).

Then, we can detect the modules in this way: an event $e' \in \mathcal{IE} \cup \mathcal{RE}$ is the root of the module $\widehat{e'}$ if all the following conditions involving its variables t_1, t_2, min_{t_1} and max_{t_l}, hold:

- $min_{t_1} > t_1$
- $max_{t_l} < t_2$
- $\forall e \in \mathcal{E} : e \in \circ e', \theta(e) \supseteq \theta(e')$

The third condition is specific of PFTs: it guarantees that $\widehat{e'}$ does not contain any shared subtree.

3.6.1 Running example

Instead of analyzing entirely the PFT model of the Multiproc system (Fig. 3.1), we can perform its analysis by modularization (section 3.6). In this section, we limit our attention on the modules detection on the PFT model of the system, according to the PFT module definition provided in section 3.6. Fig. 3.13 shows all the modules present in the PFT model; they are:
$\widehat{TE}$, $\widehat{DA}$, $\widehat{MS}$, $\widehat{CM}$, $\widehat{MM(i)}$, $\widehat{SM}$, $\widehat{BR(j)}$.
Each module is graphically indicated by a dashed line around it. The subtrees $\widehat{PU(i)}$ and $\widehat{MEM(i)}$ are not classified as modules because the subtree $\widehat{SM}$ is shared by $\widehat{PU(i)}$ and $\widehat{MEM(i)}$. So, the third condition indicated in section 3.6, for a subtree to be a module, is not respected by $\widehat{PU(i)}$ and $\widehat{MEM(i)}$.

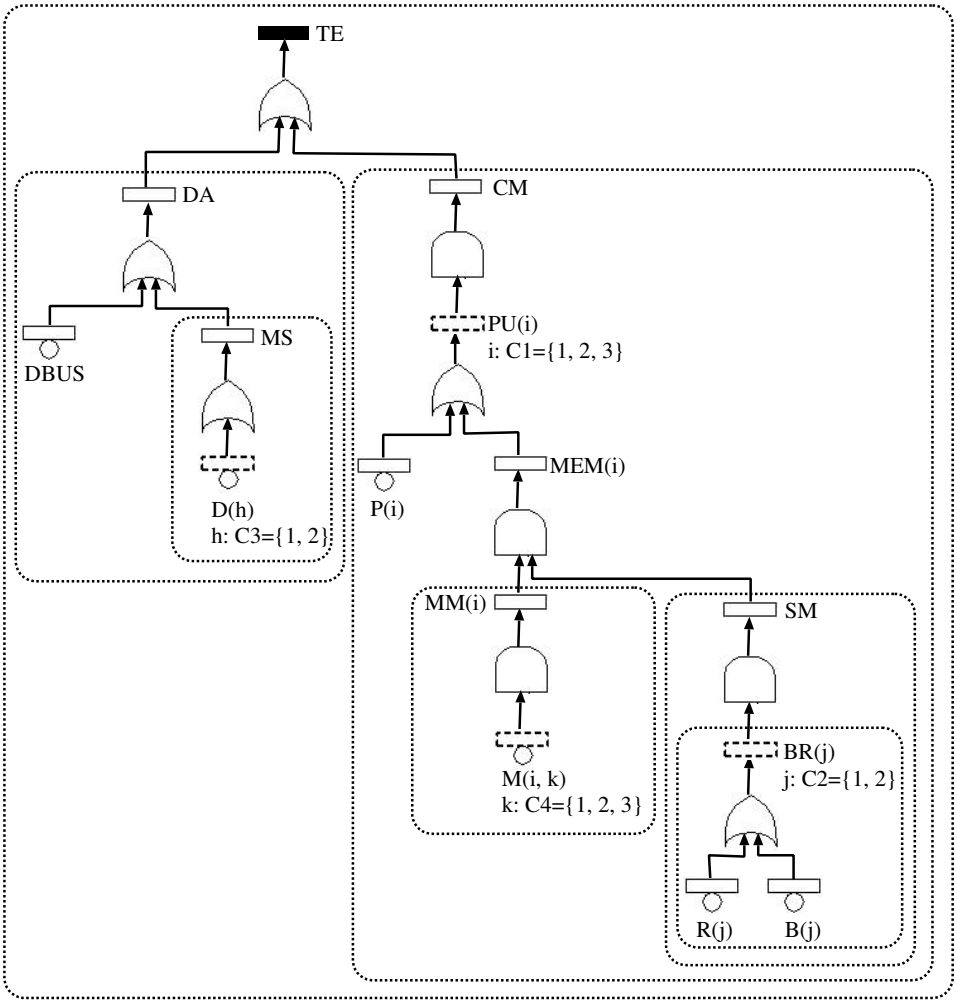

Figure 3.13: The modules in the PFT model of the Multiproc system.

Chapter 4

Petri Nets supporting DFT Analysis

4.1 Introduction to Dynamic Fault Trees

In the FT and PFT models, some assumptions hold: component failure events are assumed to be statistically independent, a component can be only in two states (working or failed) and the relations among the events are expressed by means of Boolean operators. All these assumptions allow to easily analyze both in qualitative and quantitative terms, the system modelled as a FT or a PFT, by resorting to efficient BDDs (section 2.4) or pBDDs (chapter 3).

At the same time, these assumptions are a relevant limit to the modelling power of FTs and PFTs, since in these models we can not represent dependencies involving the failure events or the state of the components.

One of the FT evolutions proposed in the literature in order to increase the modelling power of FTs, is called *Dynamic Fault Tree* (DFT) [39, 40, 70, 71, 110]. The DFT formalism in an extension of the FT formalism, where a new class of gates called *dynamic gates*, has been introduced. The dynamic gates model functional dependencies, temporal dependencies and the presence of spare components. A dependency arises in the failure process when the failure behaviour of a component depends on the state of the system.

Due to the presence of dependencies in the model, the solution techniques used for FTs, are not suitable to analyze the DFTs. While FTs are combinatorial models, DFTs need the state space solution; this means generating all the possible system states and stochastic transitions between states, according to the DFT model. In other words, we need to obtain the *Continuous Time Markov Chain* (CTMC) [83, 100] of the system, from the DFT model.

Generating the CTMC directly from a DFT, may not be so straightforward, while efficient techniques to generate the CTMC from a *Generalized Stochastic Petri Net* (GSPN) [1] are already available and are implemented in several tools, such as *GreatSPN* [26]. So, an alternative way to perform the state space solu-

tion of a DFT, consists of converting the DFT to the equivalent GSPN; then, the CTMC can be generated from the GSPN, and the Unreliability of the system can be computed on the CTMC.

In this chapter, we present a method to convert a DFT model in a GSPN [30]. The conversion of DFTs in GSPNs can be classified as a *model-to-model transformation* [36, 68], and can be expressed by means of *graph transformation* rules [3, 6, 49, 50].

4.2 Dynamic gates semantic

While Boolean gates indicate Boolean relations among the events, dynamic gates establish some kind of dependency among the events. Dynamic gates differ from Boolean gates also for other aspects:

- some dynamic gates have no input and output events, but they have trigger and dependent events;
- the input or dependent events of dynamic gates may be ordered: in this case, an order number is assigned to each of the arcs connecting the dynamic gate to its input or dependent events.

The arcs of a DFT are oriented: the arcs touching the Boolean gates have the same orientation adopted in FTs and PFTs; the orientation of the arcs touching the dynamic gates depends on the type of the gate.

The description of every dynamic gate follows:

- *Functional Dependency Gate* ($FDEP$) (Fig. 4.2.a): a gate g of type $FDEP$ is connected to one trigger event q by means of the arc (q, g), and to a set of n ($n \geq 1$) dependent events d_1, ..., d_n by means of the arcs (g, d_1), ..., (g, d_n). When q fails, d_1, ..., d_n are forced to fail. q, d_1, ..., d_n can be BEs or IEs.
- *Priority And* ($PAND$) (Fig. 4.2.b): a gate g of type $PAND$ is connected to the input events x_1, ..., x_n ($n \geq 2$) by means of the arcs (x_1, g), ..., (x_n, g), and to the output event y by means of the arc (g, y). If we indicate with $\phi(y)$ the Boolean value of y, $\phi(y) = true$ if both the following conditions hold:
  - $\bigwedge_{i=1}^{n} x_i = true$
  - $x_1, \ldots, x_n$ occurred in this order: $x_1 \prec x_2 \prec \ldots \prec x_n$.

  The order of $x_1, \ldots, x_n$ is given by an order number $(1, \ldots, n)$ assigned to each of the arcs (x_1, g), ..., (x_n, g). $x_1, \ldots, x_n$ can be BEs or IEs. y must be an IE.
- *Sequence Enforcing Gate* (SEQ) (Fig. 4.2.c): given a set of n ($n \geq 2$) events x_1, ..., x_n connected to a gate of type SEQ, x_1, ..., x_n are forced

to fail in a specific order: $x_1 \prec x_2 \prec \ldots \prec x_n$. We can classify $x_2, \ldots x_n$ as dependent events since each of them can occur only after the failure of its predecessor in the order; x_1 instead has no predecessor, so it is independent; in a sense, x_1 works as the trigger of this gate since its occurrence enables the occurrence of the successive events in the order. For this reason, in our notation, we connect x_1 to a gate g of type SEQ by means of the arc (x_1, g) having order number 1, while we connect g to x_i ($i \geq 2$) by means of the arcs (g, x_i) having order number i. We assume that the "trigger" event and the dependent events of a gate of type SEQ, can be only BEs.

- *Warm Spare Gate* (WSP) (Fig. 4.2.d): this gate models the presence of a main component m and a set of spare components $s_1, \ldots, s_m$ ($m \geq 1$) with the aim of replacing m in its function, if m fails. $s_1, \ldots, s_m$ are called "warm" spare components because they can be in three states instead of two: dormant (or stand-by), working, failed (Fig. 4.1).

  A spare is initially dormant and it turns to the working state if it has to replace the main component; at the same time, a spare may fail both in the dormant and in the working state; the spare failure rate changes depending on its current state: if the failure rate of the spare s_i is $\lambda(s_i)$ in the working state, $\alpha(s_i)\lambda(s_i)$ is its failure rate in the dormant state, with $0 < \alpha(s_i) < 1$; $\alpha(s_i)$ is the dormancy factor of the spare s_i, and its aim is to express the fact that spares have a reduced failure probability during the dormancy period. If the working spare s_i fails, m is replaced by the spare s_{i+1} instead of s_i.

  The input events of a gate g of type WSP are the BEs $m, s_1, \ldots, s_m$ modelling the failure of the main component and the failure of the spares. The input event m is connected to g by means of the arc (m, g), while the input event s_i is connected to g by means of the arc (s_i, g). The input event m is distinguished from the other ones by assigning an order number 0 to the arc (m, g). $s_1, \ldots, s_m$ must be ordered; the arc (s_i, g) has order number i ($1 \leq i \leq m$).

  If the IE y is the output event of g of type WSP, y is connected to g by means of the arc (g, y). If we indicate with $\phi(y)$ the Boolean value of y,

$$\phi(y) = m \wedge \bigwedge_{i=1}^{n} s_i$$

  We assume that the input events s_1, ..., s_m of a gate of type WSP, can not be the dependent events of a gate of type SEQ.

  Two alternative versions of this gate exist:

  - *Cold Spare* gate (CSP): $\forall s_i, \alpha(s_i) = 0$
    In this version of the gate, the spares can not fail during the dormancy period.

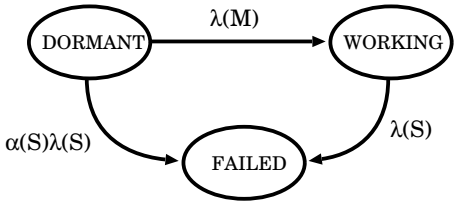

Figure 4.1: State space representation of the dependency of the spare S on the main component M.

– *Hot Spare* gate (HSP): $\forall s_i, \alpha(s_i) = 1$
In this version of the gate, the failure rate of the spares is the same both in the dormancy period and in the working period.

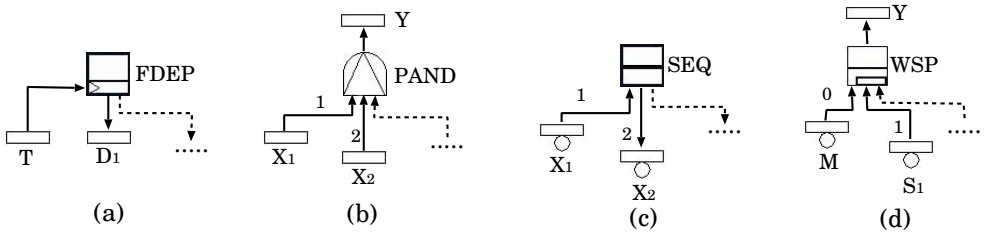

Figure 4.2: Dynamic gates: (a) FDEP, (b) PAND, (c) SEQ, (d) WSP.

An example of DFT model is shown in Fig. 4.4.

4.3 DFT formalism definition

The DFT formalism is given by the tuple
$\mathcal{DFT} = (\mathcal{E}, \mathcal{G}, \mathcal{A}, \mathcal{GT}, \gamma, \sigma, \alpha, \phi)$
where

- $\mathcal{E} = \mathcal{BE} \cup \mathcal{IE} \cup \{TE\}$ is the set of the events in the DFT; it is the union of the following sets:
  - $\mathcal{BE}$ is the set of the BEs;
  - $\mathcal{IE}$ is the set of the IEs;
  - {TE} is the set composed by the unique TE.
- $\mathcal{A} \subseteq (\mathcal{E} \times \mathcal{G}) \bigcup (\mathcal{G} \times \mathcal{E})$ is the set of the arcs.
- $\mathcal{GT} = \mathcal{BG} \cup \mathcal{DG}$ is the set of types of gate. It is the union of two sets:

  - $\mathcal{BG} = \{AND, OR\}$ is the set of Boolean gate types.
  - $\mathcal{DG} = \{PAND, FDEP, SEQ, WSP\}$ is the set of Dynamic gate types (described in section 4.2).

- $\gamma : \mathcal{G} \rightarrow \mathcal{GT}$ is the function assigning to each gate its type.
- $\sigma : \mathcal{A} \rightarrow \mathbb{N}$ is the function returning the order number of an arc.
- Given $g \in \mathcal{G} : \gamma(g) \in \mathcal{BG} \vee \gamma(g) = PAND \vee \gamma(g) = WSP$,
  - $\bullet g = \{e \in \mathcal{E} : \exists(e, g) \in \mathcal{A}\}$ is the set of input events of g ($|\bullet g| \geq 2$);
  - $g\bullet = \{e \in \mathcal{E} : \exists(g, e) \in \mathcal{A}\}$ is the output event of g ($|g\bullet| = 1$).
- Given $g \in \mathcal{G} : \gamma(g) = FDEP \vee \gamma(g) = SEQ$,
  - $\bullet g = \{e \in \mathcal{E} - \{TE\} : \exists(e, g) \in \mathcal{A}\}$ is the trigger event of g ($|\bullet g| = 1$);
  - $g\bullet = \{e \in \mathcal{E} - \{TE\} : \exists(g, e) \in \mathcal{A}\}$ is the set of dependent events of g ($|g\bullet| \geq 1$).
- Given $g \in \mathcal{G} : \gamma(g) = WSP$,
  - $\bullet_M g = \{e \in \mathcal{BE} : \exists(e, g) \in \mathcal{A} : \sigma(e, g) = 0\}$ is the input event of g relative to the main component ($|\bullet_M g| = 1$);
  - $\bullet_S g = \{e \in \mathcal{E} - \{TE\} : \exists(e, g) \in \mathcal{A} : \sigma(e, g) > 0\}$ is the set of input events of g relative to the spare components ($|\bullet_S g| \geq 1$).
  - $\bullet g = \bullet_M g \cup \bullet_S g$.
- Given $e \in \mathcal{E}$, $\bullet e = \{g \in \mathcal{G} : \exists(g, e) \in \mathcal{A}\} = \bullet_O e \cup \bullet_D e$ is the set of gates having e as output event or dependent event; it is the union of
  - $\bullet_O e = \{g \in \mathcal{G} : (\gamma(g) \in \mathcal{BG} \vee \gamma(g) = PAND \vee \gamma(g) = WSP) \wedge \exists(g, e) \in \mathcal{A}\}$ is the set of gates having e as output event.
  - $\bullet_D e = \{g \in \mathcal{G} : (\gamma(g) = FDEP \vee \gamma(g) = SEQ) \wedge \exists(g, e) \in \mathcal{A}\}$ is the set of gates having e as dependent event.

  $e\bullet = \{g \in \mathcal{G} : \exists(e, g) \in \mathcal{A}\} = e \bullet_I \cup e\bullet_T$ is the set of gates having e as input event or trigger event; it is the union of
  - $e\bullet_I = \{g \in \mathcal{G} : (\gamma(g) \in \mathcal{BG} \vee \gamma(g) = PAND \vee \gamma(g) = WSP) \wedge \exists(e, g) \in \mathcal{A}\}$ is the set of gates having e as input event.
  - $e\bullet_T = \{g \in \mathcal{G} : (\gamma(g) = FDEP \vee \gamma(g) = SEQ) \wedge \exists(e, g) \in \mathcal{A}\}$ is the set of gates having e as trigger event.
- The following conditions about the connection of events with gates, must hold:

  - $\forall e \in \mathcal{BE}, |\bullet_O e| = 0$
  - $\forall e \in \mathcal{BE}, |\bullet_D e| \geq 0$
  - $\forall e \in \mathcal{BE}, |e \bullet_I| + |e \bullet_T| \geq 1$
  - $\forall e \in \mathcal{IE}, |\bullet_O e| = 1$
  - $\forall e \in \mathcal{IE}, |\bullet_D e| \geq 0$
  - $\forall e \in \mathcal{IE}, |e \bullet_I| + |e \bullet_T| \geq 1$
  - $|\bullet_O TE| = 1$
  - $|\bullet_D TE| \geq 0$
  - $|TE \bullet| = 0$

- Given $g \in \mathcal{G} : \gamma(g) = SEQ$, we assume that
  - $\forall e \in \bullet g, e \in \mathcal{BE}$
  - $\forall e \in g\bullet, e \in \mathcal{BE}$
  - $\forall e \in g\bullet, \nexists g \in \mathcal{G} : \gamma(g) = WSP \wedge e \in \bullet_s g$
- $\lambda : \mathcal{BE} \rightarrow \mathbb{R}^+$ is the function assigning to each BE a failure rate.
- $\alpha : \mathcal{BE} \rightarrow (0, 1)$ is the function assigning to a BE connected to a gate of type WSP, a dormancy factor.
- $\phi : \mathcal{E} \rightarrow \mathbb{B} = \{true, false\}$ is the function returning the Boolean value of an event (Boolean variable).
- Given $e \in \mathcal{E}$, $\circ e = \{e' \in \mathcal{E} : \exists [e' \rightarrow e]\}$.
- Given $e \in \mathcal{E}$, $\widehat{e}$ is composed by any $[b \rightarrow e] : b \in \mathcal{BE} \cup \mathcal{BRE}$ ($\widehat{e}$ indicates the subtree rooted in e).

4.3.1 Running example

In this section, we extend the example of the Multiproc system introduced in section 2.2.1, by the addition of dependencies among some components in the system. This justifies the use of the DFT formalism to model the failure mode of the system. First, a description of the new version of the Multiproc system is provided, then its DFT model is presented.

System description

The system is mainly composed by three processing units ($PU1$, $PU2$, $PU3$), two spare memories ($R1$, $R2$), a primary ($D1$) and a backup disk ($D2$). The scheme of the system is shown in Fig. 4.3. Each processing unit is composed by one processor and one internal memory. In the case of $PU1$, they are indicated by $P1$ and $M1$ respectively.

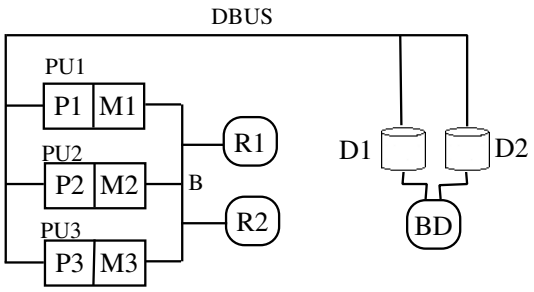

Figure 4.3: The scheme of the Multiproc system with dependencies.

Both spare memories are connected to the processing units by means of the memory bus B, and they can replace any internal memory if it fails. A spare memory can be in one of these three states: dormant, working, failed. A spare memory is in the dormant state (stand-by), while it is not replacing any internal memory; a spare memory is working while it is replacing an internal memory; from both the dormant and the working state, a spare memory can turn to the failed state. The failure rate of a spare memory changes according to its current state.

The processing units share the common primary hard disk $D1$, while $D2$ is the backup disk and contains the backup of the data contained in $D1$. In the system, there is a particular device named BD with the aim of performing the periodical update of $D2$. Initially, the processing units access $D1$ to store and retrieve their data, while $D2$ is only periodically accessed by BD for the update operations; so we assume that in this situation, $D2$ can not fail. If $D1$ fails, the processing units access $D2$ to read or write data instead of $D1$; from this moment, $D2$ can fail and its failure rate is equal to the failure rate of $D1$. Both $D1$ and $D2$ can be accessed by the processing units, through the disk bus $DBUS$.

The failure mode of the system

The correct functioning of at least one processing unit is required for the system to be working; so, the failure of all the processing units causes the whole system failure. A processing unit fails in two cases: if its processor fails, or if its internal memory fails and there are no available spare memories to replace it. A spare memory is available to replacement if it is not failed and it is not already replacing another internal memory. Moreover, the spare memories functionally depend on the memory bus B; so, if B fails the spare memories cannot be accessed by the processing units, and this has the same effect of the contemporary failure of the spare memories.

Another cause of failure of the system, is the compromised access to the hard disks; this happens in three cases: the failure of the disk bus $DBUS$, the failure of both $D1$ and $D2$, and the failure of BD. The failure of $DBUS$ prevents the access to both hard disks by the processing units; the failure of both disks prevents

Component	Failure rate (λ)	Dormancy factor (α)
Processor	5.0E-7 h^{-1}	
Disk	8.0E-7 h^{-1}	
Memory	3.0E-8 h^{-1}	0.1
Bus	2.0E-9 h^{-1}	
Backup Device	7.0E-8 h^{-1}	

Table 4.1: The failure rate for each type of component.

the storing and the retrieving of the data by the processing units; the failure of BD prevents the update of $D2$ while $D1$ is not yet failed.

The failure of BD is relevant only if it happens before the failure of $D1$. In this case, when $D1$ fails and is replaced by $D2$, this one is not updated, due to the previous failure of BD. If instead BD fails after the failure of $D1$, the update operation is no more necessary since the processing units access $D2$ instead of $D1$. So, in this case, the failure of BD has no negative effect on the correct functioning of the system.

The time to fail of the components is a random variable obeying to the negative exponential distribution; Tab. 4.1 indicates the failure rate (and the dormancy factor) for each type of component.

The DFT model of the system

The DFT model in Fig. 4.4 represents the failure mode of the system described in section 4.3.1. In this DFT, the TE is caused by the compromised access to the hard disks or by the failure of all the processing units, so TE is the output of a gate of type OR whose input events are DA and CM representing the TE causes respectively.

The IE named DA is the output of an OR gate whose input events are $DBUS$, UPD, MS representing the failure of the disk bus, the failed update of the backup disk, and the failure of both disks, respectively. The event UPD is the output of a gate of type $PAND$ whose ordered input events are BD and $D1$, representing the failure of the primary disk and the failure of the device dedicated to the update operations, respectively. So, UPD occurs when both BD and $D1$ have occurred, and if BD occurred before $D1$. The failure of the primary disk is represented by the BE $D1$, while the failure of the backup disk is represented by the BE $D2$. Since $D2$ can not fail before the failure of $D1$, $D1$ and $D2$ are connected to a gate of type SEQ by means of an arc with order number 1, and by means of an arc with order number 2, respectively. In this way, we indicate the order of failure ($D1 \prec D2$). The event MS is the output of a gate of type AND with $D1$ and $D2$ as input events.

The event CM indicates the failure of all the processing units, so it is the output of a gate of type AND having the events $PU1$, $PU2$, $PU3$ as input events. Each

of these input events represents the failure of one of the processing units. The event $PU1$ is the output of a gate of type OR whose input events are $P1$ and $MEM1$; $P1$ is a BE modelling the failure of the processor, while $MEM1$ represents the failure of the internal memory and the contemporary impossibility of the failed internal memory to be replaced by any spare memory. $MEM1$ is the output of a gate of type WSP having as input events $M1$, $R1$, $R2$. $M1$ is the BE relative to the internal memory and is connected to the gate by means of an arc whose order number is 0 to indicate that $M1$ is the main component of the gate. $R1$ and $R2$ are the BEs relative to the spare memories and are connected to the WSP gate by means of an arc with order number 1, and by means of an arc with order number 2, respectively; in this way, we indicate that $R1$ and $R2$ are spare components, and $R1$ must be used before $R2$ to replace $M1$, if $R1$ is available, i. e. $R1$ is not already replacing another main component, and is not already failed.

The failure of $PU2$ and $PU3$ is modelled in the DFT in a similar way. The BEs $R1$ and $R2$ are connected also to other two gates of type WSP, since $R1$ and $R2$ can replace also the internal memories of the other processing units. The functional dependency of the spare memories on the memory bus is modelled by a gate of type $FDEP$ having the BE B as trigger event, $R1$ and $R2$ as dependent events.

The failure rates (and the dormancy factors) of the BEs relative to the component of the system, are indicated in Tab. 4.1. The events in the DFT model and the corresponding components and subsystems, are summarized in Tab. 4.2.

4.4 Introduction to GSPN

GSPNs are an extension of the Petri Nets, characterized by the presence of two types of transitions: *immediate transitions* and *timed transitions*. Immediate transitions fire as soon as are enabled, while the firing of timed transitions is delayed of a period of time whose duration is a random variable ruled by a negative exponential distribution whose parameter is the *firing rate* assigned to the timed transition. Immediate transitions are graphically represented by black rectangles, while timed transitions are represented by white rectangles.

The primitives of the GSPN formalism are: places, transitions and arcs.

As in ordinary Petri Nets, the places of a GSPN contain a discrete number of tokens; the marking of a certain place is the number of tokens inside that place. A place is graphically represented as a circle. In a GSPN, the current state of the system is modelled by the current net marking, i. e. the number of tokens in each place of the net. Transitions are used to model the system state transitions; a transition is enabled when a certain net marking holds, and when the transition fires, some tokens are moved from a place to another changing the net marking, so the system state.

In GSPNs we have also two types of arcs: oriented arcs and inhibitor arcs. Oriented arcs are used to connect places to transitions and vice-versa, with the aim

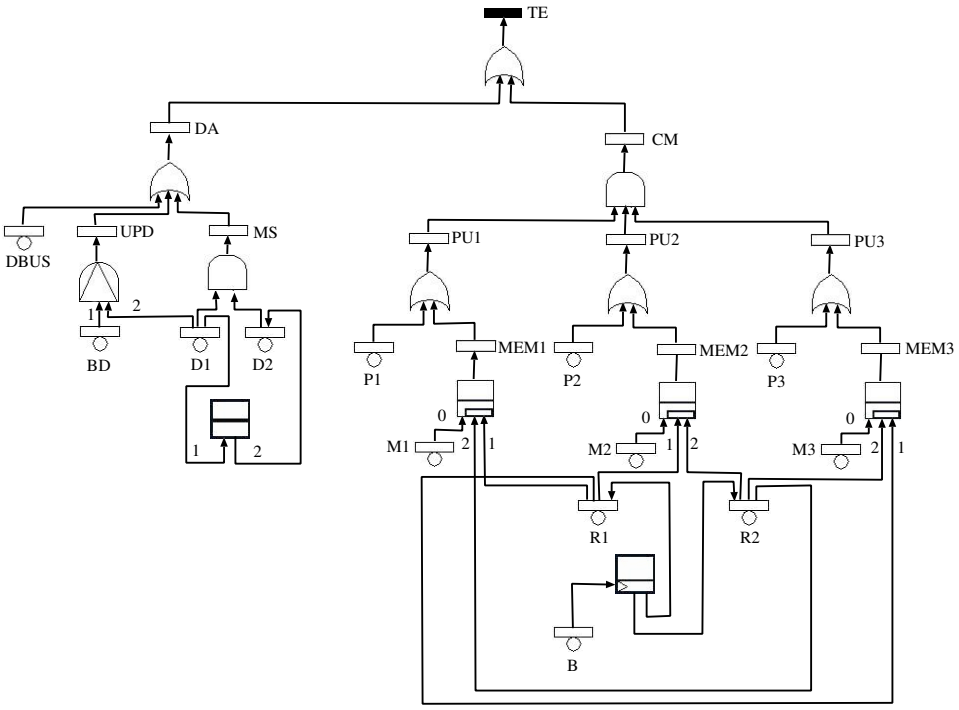

Figure 4.4: The DFT model for the Multiproc system.

Event	Component / Subsystem
DA	Disk Access
$DBUS$	Disk Bus
UPD	Backup disk Update
BD	Device for the Backup Disk update
MS	Mass Storage
$D1$	Primary Disk
$D2$	Backup Disk
CM	Computing module
$PU1$	Processing Unit 1
$P1$	Processor of the Processing Unit 1
$MEM1$	Memory access of the Processing Unit 1
$M1$	Internal Memory of the Processing Unit 1
$PU2$	Processing Unit 2
$P2$	Processor of the Processing Unit 2
$MEM2$	Memory access of the Processing Unit 2
$M2$	Internal Memory Module of the Processing Unit 2
$PU3$	Processing Unit 3
$P3$	Processor of the Processing Unit 3
$MEM3$	Memory access of the Processing Unit 3
$M3$	Internal Memory Module of the Processing Unit 3
$R1$	Spare Memory 1
$R2$	Spare Memory 2
B	Memory Bus

Table 4.2: Correspondence between the events and the components or subsystems.

of moving tokens when transitions fire. Inhibitor arcs connect a place to a transition with the aim of disabling the transition if the place is not empty. A cardinality can be associated to an arc; in the case of oriented arcs, the cardinality indicates the number of tokens to be moved on that arc when the transition fires; in the case of inhibitor arcs, the cardinality indicates the number of tokens inside the place, necessary to disable the transition.

In a GSPN, two or more immediate transitions may be enabled at the same time; in this case, *weights* and *priorities* can be used to rule the firing of such transitions. A weight and/or a priority can be assigned to an immediate transition. Using weights, given several immediate transitions enabled to fire, higher is the weight of a transition, higher is its probability to fire. Using priorities, given several immediate transitions enabled to fire, the transition with highest priority fires.

4.4.1 GSPN analysis

The analysis of a GSPN is performed on the corresponding CTMC; the GSPN analysis provides measures such as the probability of a certain marking of place or the throughput of a transition. The analysis of a GSPN can be transient or steady-state; in the first case, the measures are computed at a given finite time t; in the second case they are computed for $t = +\infty$.

In order to obtain the CTMC corresponding to a GSPN, first the *reachability graph* is generated. The reachability graph expresses all the possible markings of the GSPN, which are reachable from the initial marking through the firing of transitions. In the reachability graph, we distinguish between *vanishing* markings and *tangible* markings. A vanishing marking enables one or more immediate transitions to fire; a tangible marking enables the firing of one or more timed transitions, and of no immediate transitions. By reducing the reachability graph to contain only tangible markings, we obtain the CTMC corresponding to the GSPN, where a state is given by a marking, while a state transition is given by the firing of a timed transition. The rates associated with the state transitions are the firing rates of the timed transitions in the GSPN.

The measures returned by the GSPN analysis, can be obtained also by means of GSPN simulation. The use of simulation instead of analysis, is useful when the number of states (and state transitions) in the CTMC, is very high. In this situation, the analysis becomes computationally expensive, or even unfeasible. The *GreatSPN* tool [26] allows to draw, analyze and simulate GSPNs.

Further information on the GSPN formalism, analysis and simulation, can be found in [1].

4.4.2 GSPN formal definition

The GSPN formalism is given by the tuple
$GSPN = (P, T, A, m, \lambda, w, \pi, card)$
where

- P is the set of the places.
- $T = T_i \cup T_t$ is the set of the transitions; it is the union of two sets:
  - T_i is the set of the immediate transitions;
  - T_t is the set of timed transitions.
- A is the set of arcs; it is the union of two sets:
  - $A_d \subseteq (P \times T) \cup (T \times P)$ is the set of the oriented arcs;
  - $A_h \subseteq P \times T$ is the set of the inhibitor arcs.
- $m : P \to \mathbb{N}$ is the function returning the marking of a place.
- $\lambda : T_t \to \mathbb{R}^+$ is the function returning the firing rate of a timed transition.
- $w : T_i \to \mathbb{R}^+$ is the function returning the weight of an immediate transition.
- $\pi : T_i \to \mathbb{N} - \{0\}$ is the function returning the priority of an immediate transition.
- $card : A \to \mathbb{N} - \{0\}$ is the function returning the cardinality of an arc.
- Given $t \in T$
  - $\bullet_d t = \{p \in P : \exists (p,t) \in A_d\}$ is the set of the places such that an oriented arc is drawn between each of them and t. Such places are referred as input places.
  - $t\bullet_d = \{p \in P : \exists (t,p) \in A_d\}$ is the set of places such that an oriented arc is drawn between t and each of them. Such places are referred as output places.
  - $\bullet_h t = \{p \in P : \exists (p,t) \in A_h\}$ is the set of places such that an inhibitor arc is drawn between each of them and t. Such places are referred as inhibitor places.
- Given $t \in T$, t is enabled to fire if all the following conditions hold:
  - $\forall p \in \bullet_d t, m(p) \geq card((t,q))$
  - $\forall p \in \bullet_h t, m(p) < card((t,q))$

  If $m'(p)$ is the marking of $p \in P$ before the firing of t, and $m''(p)$ is the marking of p after the firing of t, then the effect of the firing of t is the following:
  - $\forall p \in \bullet_d t, m"(p) = m'(p) - card((p,t))$
  - $\forall p \in t\bullet_d, m"(p) = m'(p) + card((p,t))$

4.5 Concepts of model-to-model transformation

A model-to-model transformation consists of mapping a model (*source model*) to another model (*target model*) according to a set of rules.

A model-to-model transformation can be described by specifying how a model conforming a certain formalism (source formalism), is mapped into a corresponding model conforming another formalism (target formalism). Several approaches were proposed to specify model-to-model transformations; the graph transformation approach is the natural candidate to describe how a model-to-model transformation must be performed, when we have to deal with graphical models (such as DFTs and GSPNs) since they are forms of graph.

Using the graph transformation approach, we specify how the elements of the source model are mapped into elements of the target model, by means of a set of graph transformation rules. A model (graph) is derived from another model by applying one rule after another. The same rule may be applied several times.

4.5.1 Graph transformation rules

A graph transformation rule has the form $r = (L, R, K, glue, emb, appl)$ [3], where

- L is the graph called *left hand side* of the the rule r.
- R is the graph called *right hand side* of r.
- K is the *interface graph*, i. e. a common subgraph of L and R.
- $glue$ is an occurrence of K in R.
- emb is the embedding relation.
- $appl$ is a set of application conditions.

The application of a rule r to the graph G, follows the following steps [3]:

1. SEARCH the occurrences of L in G. If any, the application of r can go on, else it ends producing no modification to G.
2. CHOOSE an occurrence of L in G.
3. CHECK $appl$ in the chosen source match; if all the conditions in $appl$ are satisfied, the application of r to G can go on, else it ends producing no modification to G.
4. REMOVE from G the occurrence of L up to the occurrence of K ($L - K$); *dangling edges*, i. e. edges incident to removed nodes, are removed as well. This step generates the *context* graph D. D contains an occurrence of K.

5. GLUE D and R according to the occurrences of K in D and R. This means adding $R - K$ to D. This step generates the *gluing* graph E.

6. EMBED R in D according to emb: for each removed dangling edge incident with a node v in D and with the image of a node v' of L in G, and each node v" in R, a new edge incident with v and v" is created in E, provided that (v', v'') belongs to emb.

In the steps listed above, we use the concept of *occurrence*. We have an occurrence of the graph A in B if there is a mapping $occ : A \rightarrow B$ such that occ maps the nodes and the edges of L to the nodes and edges in G. This means that occ maps each node v in A to its image v' in B, and occ maps each edge (v_1, v_2) in A to the edge (v_1', v_2') in B such that v_1' is the image of v_1 and v_2' is the image of v_2. Often, in order to define an occurrence, the *injectivity condition* is used: an occurrence of L in G is a subgraph of G isomorphic to L.

If A and B are labelled graphs, i. e. graphs whose elements (nodes or edges) are identified by a label, occ must preserve labels: if x is an element of A, and x' is the image of x in B, x and x' must have the same label.

If A and B are attributed graphs, i. e. graphs whose elements are equipped with attributes (numbers, strings, ...), occ must preserve the attributes and their values: if x is an element of A, and x' is the image of x in B, each attribute specified for x' must be specified for x, and the value of the attribute $x'.a$ of x' must be the same value of $x.a$ of x.

The application of r to G produces the graph H; we call such application a *direct derivation* from G to H trough r, and is denoted by $G \Rightarrow_r H$. If emb is empty, the EMBED step is not performed, so no additional edges are inserted. If K is empty, in the GLUE step, R is added disjointly to D.

A set of graph transformation rules is a *graph transformation system*. If P is a graph transformation system, $G_0 \Rightarrow G_1 \Rightarrow \ldots \Rightarrow G_n$ is the *derivation* from G_0 to G_n through the rules in P.

A graph transformation system may be non-deterministic. Given a certain graph, several rules may be applicable to it. In other words, the graph may contain the occurrences of several left hand sides of rules in the graph transformation system. Moreover, if we choose to apply a certain rule among the applicable ones, several occurrences of the left hand side of the rule may present in the graph. So, we might have to choose which rule has to be applied to the graph, and to which occurrence in the graph, of its left hand side.

The result of the graph transformation may change according to such arbitrary choices. The non-determinism of a graph transformation system can be limited or avoided by means of some expedients [3]:

- setting an order in which rules must be applied;
- the next rule to apply depends on the currently applied rule;
- assigning a priority to each rule.

4.5.2 Properties of graph transformation

This section provides some properties that a graph transformation system may have:

- *Invertability*. During a graph derivation, if the current graph is G_i, the *undo* operation allows to go backward to graph G_{i-1}. In other words, the undo operation cancels the effects of the last rule application. The undo operation consists of an inverted rule application, and it is possible if the rule satisfies the following conditions:

  - *Contact condition*: no dangling edges arise in the REMOVE step. The contact condition holds if whenever a node v in the occurrence of L in G, contacts some edge not in the occurrence of L in G, v must be in the occurrence of K ($K \subseteq L$).
  - *Identification condition*: an occurrence of L in G may only identify nodes and edges in K.
  - $glue$ is injective.
  - emb is empty.

  When we invert the application of a rule, the set of application conditions $appl$ of L, becomes the set of application conditions of R.

- *Confluence*. Two direct derivations $G \Rightarrow_{r_1} G_1$ and $G \Rightarrow_{r_2} G_2$ *commute* if a graph H exists such that $G_1 \Rightarrow_{r_2} H \wedge G_2 \Rightarrow_{r_1} H$. A graph transformation system is *confluent* if for each two derivations $G \Rightarrow^* G_1$ and $G \Rightarrow^* G_2$, a graph H exists such that $G_1 \Rightarrow^* H \wedge G_2 \Rightarrow^* H$. The confluence property implies that every graph can be transformed into at most one irreducible graph.

- *Termination*. A graph transformation system is called *terminating* if infinite derivations $G_1 \Rightarrow G_2 \Rightarrow \ldots$ are impossible.

Several graph transformation approaches exist; one of them is called *Double Push Out* (DPO) [49]. In this approach, all the following properties hold for all the rules: the *Contact condition*, *Identification condition*, emb is empty.

4.5.3 Rule based model-to-model transformation

A model-to-model transformation based on the use of graph transformation rules, can be performed by means of a set of *compound rules* [68]. A compound rule r consists of two graph transformation rules and is expressed as

$$r : (r_s, r_t)$$

where r_s is the transformation rule for the source model, and r_t is the transformation rule for the target model. r_s can be expressed as

$$r_s = (L_s, R_s, K_s, glue_s, emb_s, appl_s)$$

r_t can be expressed as

$$r_t = (L_t, R_t, K_t, glue_t, emb_t, appl_t)$$

(see section 4.5.1).

r_s and r_t have the ability of sharing *variables*; variables are useful to transfer information from the source model to the target model. For instance, we can use a variable to set the value of an attribute in the target model, to the value (or to the modified value) of an attribute in the source model.

The application of a compound rule consists of the application of r_s to the source model, and the application of r_t to the target model. In other words, the application of a compound rule consists of the execution of two parallel direct derivations performed on the source model through r_s, and on the target model through r_t. A model-to-model transformation is realized through the application of several compound rules: if Q is the set of compounds rules, a model-to-model transformation is the execution of two parallel derivations performed on the source model and on the target model respectively, through Q.

If $V = \{v_1, \ldots v_n\}$ is the set of the variables, the application of the compound rule r to the source model S and to the target model T, follows these steps [68]:

1. SEARCH the *source matches*, i. e. the occurrences of L_s in S. If any, the application of r can go on, else it ends.

2. CHOOSE a *source match*.

3. INSTANTIATE the variables according to the attribute values in the chosen source match. This leads to the variable instantiation denoted by V^I.

4. INSTANTIATE L_s, K_s, R_s, L_t, K_t, R_t according to V^I; this means assigning to the variables in L_s, K_s, R_s, L_t, K_t, R_t the corresponding values in V^I, obtaining $L_s(V^I)$, $K_s(V^I)$, $R_s(V^I)$, $L_t(V^I)$, $K_t(V^I)$, $R_t(V^I)$, respectively.

5. APPLY $r_s(V^I) = (L_s(V^I), R_s(V^I), K_s(V^I), glue_s, emb_s, appl_s)$ to S, and $r_t(V^I) = (L_t(V^I), R_t(V^I), K_t(V^I), glue_s, emb_s, appl_s)$ to T.

We define as *model transformation system*, a set of compound rules to convert models conforming a source formalism, into models conforming the target formalism.

4.6 Converting a DFT model into a GSPN

This section provides a model transformation system to map DFT models to the equivalent GSPNs. For each type of event and for each type of gate, one or several compound rules are defined. The source model is a DFT, while the target model is a GSPN; initially, the target model is empty; at each step of the model-to-model transformation, an event or a gate of the target model is considered, and mapped in a net which is added to the target model.

The compound rules proposed in this section are in the form described in section 4.5.3. The graph transformation rules for the source model and the target model are in the form described in section 4.5.1. We indicate a graph transformation rule by means of three boxes containing the left hand side of the rule (L), the interface graph (K), and the right hand side of the rule (R), respectively. This graphical notation was introduced for the DPO approach [49].

The model transformation system proposed in this section, requires the definition of two new functions in the DFT formalism:

- $conv : \mathcal{E} \rightarrow \mathbb{B}$ is the function returning the $true$ value if an event in the source DFT model, has already been mapped in the GSPN target model, and returning the $false$ value if an event in the source model has not yet been mapped in the target model.
- $lab : \mathcal{E} \rightarrow \{A, \ldots, Z\}^+$ is the function returning the label assigned to an event, where $\{A, \ldots, Z\}^+$ is the set of all the possible non empty strings we can compose with the alphabet $\{A, \ldots, Z\}$.

The lab function has been defined also in the GSPN formalism:

- $lab : P \cup T \rightarrow \{A, \ldots, Z\}^+$ is the function returning the label assigned to a place or transition.

In the compound rules in our model transformation system, labels are used to identify the nodes inside the source model and the target model:
$\forall e, e' \in \mathcal{E} : e \neq e', lab(e) \neq lab(e')$
$\forall p, p' \in P : p \neq p', lab(p) \neq lab(p')$
$\forall t, t' \in T : t \neq t', lab(t) \neq lab(t')$

The $conv$ function is useful to avoid the repeated conversion of the same event of the DFT source model. In our compound rules we use variables to transfer information from the graph transformation rule for the source model, to the graph transformation rule for the target model.

We can classify DFTs and GSPNs as labelled attributed oriented graphs (section 4.5.1); labels are returned by the function lab, while attributes are returned by other functions defined in the DFT formalism (section 4.3) and in the GSPN formalism (section 4.4.2).

So, in the transformation rules in our model transformation system, L, K, R must be labelled attributed oriented graph. In the box containing L we have the

structure of L in terms of nodes and oriented edges, together with the values that the labels and the attributes of L must have in order to find an occurrence of L. The value of a label or an attribute in L is indicated by an expression beginning with "if". For instance, in the compound rule in Fig. 4.5, in L_s, the value of the attribute $conv(E)$ of the event E, is expressed in this way: $if\ conv(E) = false$. In the compound rule in Fig. 4.9, in L_t, the value of the label of the place P ($lab(P)$) is expressed in this way: $if\ lab(P) =< name1 > +$"$\_dn$", where $< name1 >$ is a variable and the $+$ operator allows to append a string to another.

In the box containing K, the values of the labels and of the attributes are not repeated, but they are the same as in L. In the box containing R, we indicate only the value of the attributes that must be be changed; for instance, in the compound rule in Fig. 4.5, in R_s, the attribute $conv(E)$ of the event E, is set to $true$.

In a compound rule, some variables may be used, they are declared in the box containing L_s, together with the value they must be instantiated to. For instance, in the compound rule in Fig. 4.5, in L_s, we have $< name >:= lab(E)$; $< name >$ is a variable whose value must be instantiated to $lab(E)$.

Some compound rules may have an higher priority with respect to other ones.

There is a general correspondence between DFT elements and the elements of the equivalent GSPN obtained through our model transformation system:

- generic event $\Leftrightarrow$ place
- not occurred event $\Leftrightarrow$ empty place
- occurred event $\Leftrightarrow$ marked place
- basic event occurrence $\Leftrightarrow$ timed transition firing
- gate $\Leftrightarrow$ set of immediate transitions

The description of all the compound rules used in our model transformation system, follows. In the target graph (GSPN) transformation rules, whenever it is not differently indicated, we have that

- the marking of a place (number of tokens in a place) is equal to 0;
- the priority of an immediate transition is equal to 1;
- the cardinality of an oriented arc is 1;
- the cardinality of an inhibitor arc is 1.

4.6.1 Events conversion

IE conversion

Fig. 4.5 is the compound rule to map in the GSPN, an IE of the DFT. r_s acts on the DFT source model and can be applied to $E \in \mathcal{IE} : conv(E) = false$; in other

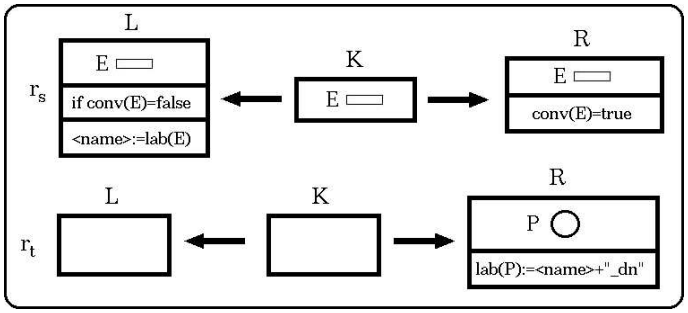

Figure 4.5: Compound rule for an IE.

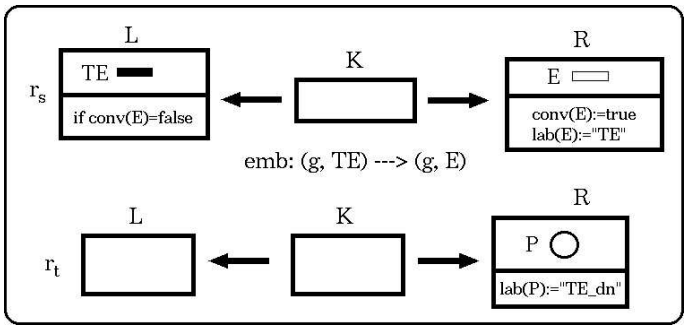

Figure 4.6: Compound rule for the TE.

words, r_s can be applied to an IE which has not been yet mapped in the GSPN target model. The variable $< name >$ is used to transfer the label of E from the DFT to the GSPN and must be set to the value of $lab(E)$.

Since $L_s = K_s = R_s$, no element is removed or added in the DFT, so r_s does not modify the structure of the DFT; r_s only changes the value of $conv(E)$ setting $conv(E)$ to the $true$ value, as indicated in R_s. In this way, this model transformation rule can not be applied again to the same IE.

r_t acts on the GSPN target model; since L_t is empty, the effect of this rule is the addition of new elements in the GSPN model, with no consideration about its current composing elements. More precisely, r_t adds in the GSPN a place P whose label is given by $< name > +"\_dn"$, where the variable $< name >$ contains the label of the IE E. This place is the mapping of E in the GSPN.

TE conversion

Fig. 4.6 shows the compound rule to map in the GSPN the TE of the DFT. r_s can be applied to the TE of the DFT, if $conv(TE) = false$, i. e. if the TE has not been already mapped in the GSPN. The effect of r_s is removing the TE from the DFT and replacing it with an IE indicated by E, such that $conv(E) = true$ and

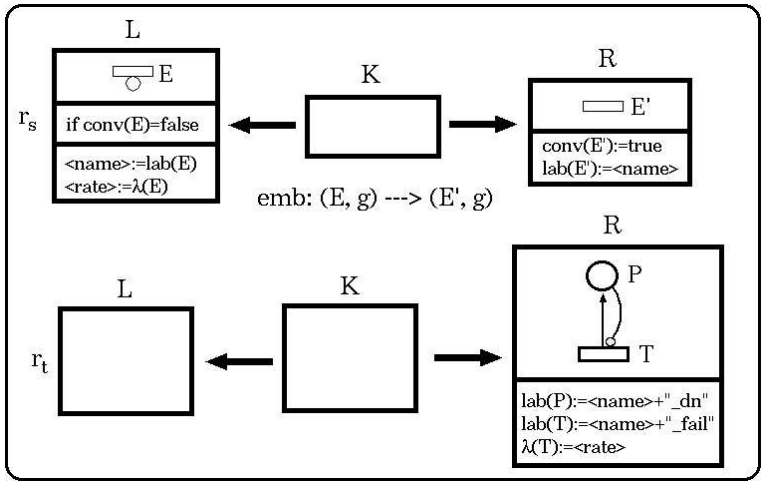

Figure 4.7: Compound rule for a BE.

$label(E) = "TE"$.

The replacement of the TE with E avoids the repeated application of the rule in Fig. 4.6. Moreover, since $conv(E) = true$, the compound rule in Fig. 4.5 can not be applied to E. In other words, only one rule is applied to map the TE in the GSPN, and only once.

The removal of the TE determines the removal of the arc pointing the TE (dangling edge). For this reason, an embedding relation is defined in r_s: given the removed arc (g, TE) such that $g \in \mathcal{G}$, an arc (g, E) is created in the DFT.

The effect of r_t on the GSPN, is adding a place P such that $lab(P) = "TE\_dn"$. Such place is the mapping of the TE in the GSPN.

BE conversion

Fig. 4.7 shows the compound rule to map in the GSPN a BE of the DFT. r_s can be applied to a BE E of the DFT such that E has not already been mapped in the GSPN ($conv(E) = false$). Two variables are present: $< name >$ and $< rate >$; they are instantiated to the label of the E and to its failure rate, respectively.

The effect of r_s on the DFT is the replacement of the BE E with the IE E' such that E' has the same label of E and $conv(E') = true$. The removal of E determines the removal of the arc(s) touching E; for this reason, an embedding relation is present in r_s: for each removed arc (E, g) such that $g \in \mathcal{G}$, an arc (E', g) is created.

The replacement of E with E' avoids the repeated application of the rule in Fig. 4.7 to the same BE. Moreover, since $conv(E') = true$, the compound rule in Fig. 4.5 can not be applied to E'. So, only one rule is applied to map a BE in the GSPN, and only once.

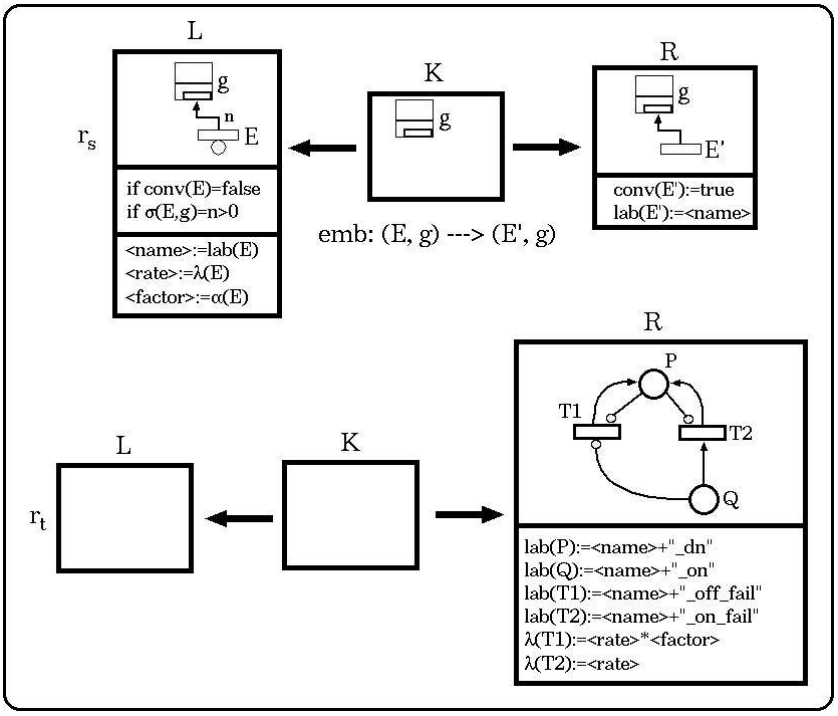

Figure 4.8: Compound rule for a BE relative to a spare component.

r_t adds in the GSPN a place P and a timed transition T, together with an oriented arc (T, P) and an inhibitor arc (P, T). The label of P is $< name >$ $+$"$\_dn$", where $< name >$ contains the label of E. The label of T is $< name >$ $+$"$\_fail$", while its firing rate is the same as the failure rate of the BE E in the DFT.

The net in R_t is the mapping of a BE in the GSPN. The place P represents the BE E; this place contains no token to indicate that the event has not yet occurred. The transition T models the occurrence of the event; when it fires, it puts one token in P through (T, P), to represent that the event has occurred. The firing of T is not repeatable due to the inhibitor arc (P, T).

Conversion of BEs relative to spare components

Fig. 4.8 shows the compound rule to map in the GSPN a BE relative to a spare component. We can apply r_s to a BE E which is the input event of a gate of type WSP, and has not yet been mapped to the GSPN. Moreover, E must be connected to the WSP gate through an arc having order number $n > 0$ indicating that E is the BE relative to a spare component.

Three variables are present: $< name >$, $< rate >$, $< factor >$; $< name >$ is set to the label of E, $< rate >$ to the failure rate of E, and $< factor >$ to the

dormancy factor of E.

In the DFT, r_s removes the BE E and replaces it with the IE E' such that $conv(E') = true$; moreover E' has the same label of E. The removal of E determines the removal of any arc touching E; so, an embedding relation is present also in this rule: for each removed arc (E, g) such that $g \in \mathcal{G}$, an arc (E', g) is created.

The replacement of E with E' avoids the repeated application of the rule in Fig. 4.8 to the same BE. Moreover, since $conv(E') = true$, the compound rule in Fig. 4.5 can not be applied to E'. So, only one rule is applied to map a BE relative to a spare, in the GSPN, and only once.

The action of r_t on the GSPN, is the addition of a net composed by the places P and Q, and by the timed transitions $T1$ and $T2$. The label of the place P is $< name >$ +"_dn"; this place represents the failed state of the spare: if P is marked, the spare is failed. The label of the place Q is $< name >$ +"_on"; this place represents the working state of the spare: if Q is marked, the spare is working.

The label of the transition $T1$ is $< name >$ +"$\_off\_fail$", while the value of its firing rate is given by the product of the failure rate of E and the dormancy factor of E. The label of the transition $T2$ is $< name >$ +"$\_on\_fail$", while the value of its firing rate is the value of the failure rate of E. The transition $T1$ represents the failure of the spare during the dormant state; $T1$ can fire if Q is not marked, i. e. if the spare is not working. When $T1$ fires, it puts one token in P to indicate the failed state of the spare. The transition $T2$ represents the failure of the spare during the working state; $T2$ can fire only if Q is marked, i. e. if the spare is working. As $T1$, when $T2$ fires, it puts one token in P to indicate the failed state of the spare. $T1$ and $T2$ can not be enabled at the same time, and are both disabled by the presence of one token in P. Such net is the mapping in the GSPN of a BE relative to a spare component.

Given a BE E in the DFT, such that E is relative to a spare component and $conv(E) = true$, both the compound rule in Fig. 4.7 and the compound rule in Fig. 4.8 can be applied to E, but the correct rule to be applied to E, is in Fig. 4.8. For this reason, we set for the rule in Fig. 4.8 an higher priority with respect to the rule in Fig. 4.7. In this way, the rule in Fig. 4.8 is surely applied to E.

4.6.2 Boolean gates conversion

AND gate conversion

Fig. 4.9 shows the compound rule for a gate of type AND and its output event. r_s can be applied to a gate g of type AND such that its output event E has already been mapped in the GSPN (by means of the compound rule in Fig. 4.5 or by means of the compound rule 4.6). The output event E is identified through the arc connecting g to E; such arc must be drawn as a continuous line. The variable $< name >$ is instantiated to the label of E.

The effect of r_s on the DFT, is the removal of the arc (g, E), where g is the

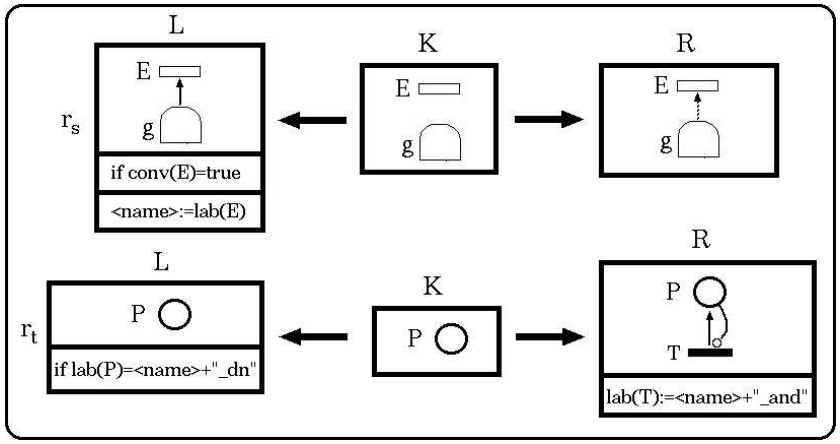

Figure 4.9: Compound rule for of a gate of type AND and its output event.

gate of type AND. Such arc is replaced with another arc still connecting g to E, but drawn as a dashed line. In this way, the rule in Fig. 4.9 is applied to the same AND gate and the same output event, only once.

r_t can be applied to the place P such that its label is given by $< name >$ +"$\_dn$"; this means that P is the place generated by the mapping of E in the GSPN. Since $L_s = K_s$, nothing is removed from the GSPN; an immediate transition T is added to the GSPN; the label of T is $< name >$ +"$\_and$". An oriented arc (T, P) and an inhibitor arc (P, T) are also created.

Fig. 4.10 shows the compound rule for a gate of type AND and one of its input events. r_s can be applied to a gate g of type AND such that

- its output event Y has already been mapped in the GSPN (by means of the compound rule in Fig. 4.5 or by means of the compound rule 4.6);
- the gate g has already been partially mapped in the GSPN through the rule in Fig. 4.9 (dashed arc connecting g to Y).
- the input event X has already been mapped in the GSPN (by means of the compound rule in Fig. 4.5 or by means of the compound rule 4.7). X must be connected to g through an arc drawn as a continuous line.

Two variables are present: $< name1 >$ and $< name2 >$. They are instantiated to the label of Y and to the label of X, respectively.

The effect of r_s on the DFT is the removal of the arc (X, g) and its replacement with another arc from X to g, but drawn as a dashed line. In this way, we apply the rule in Fig. 4.10 to the same AND gate and to the same input event, only once.

r_t must be applied to a subnet of the GSPN composed by the following nodes:

- a place P whose label is equal to $< name1 >$ +"$\_dn$"; this means that P is the mapping of Y in the GSPN;

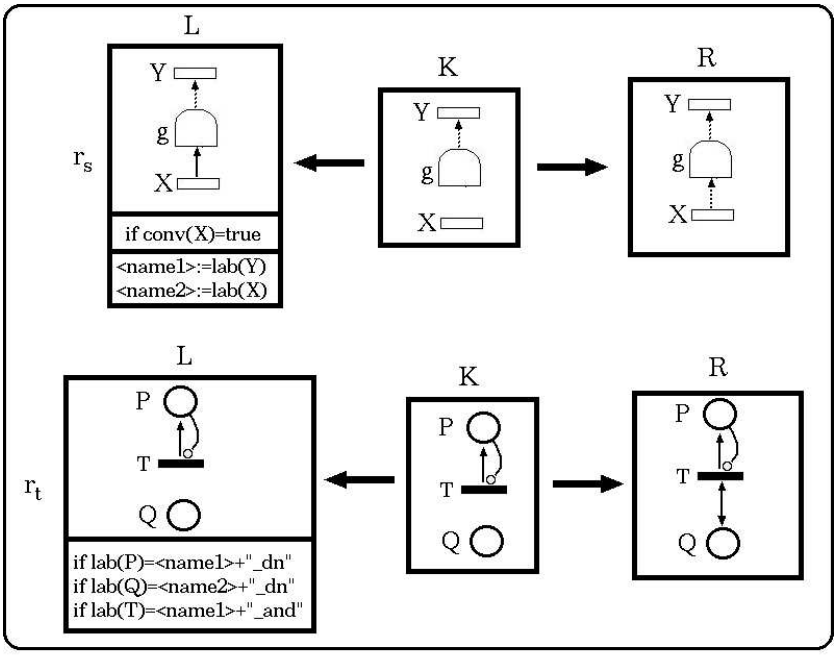

Figure 4.10: Compound rule for a gate of type AND and one of its input events.

- a place Q whose label is equal to $< name2 > +$"$\_dn$"; this means that Q is the mapping of X in the GSPN;
- an immediate transition T whose label is $< name1 > +$"$\_and$"; this means that this transition was created by the application of the rule in Fig. 4.9 to g and Y.

The effect of r_t on the GSPN is the addition of a couple of oriented arcs: (Q, T) and (T, Q).

Example. In Fig. 4.11, we can see how a gate of type AND whose output event is labelled as MS and its input events are labelled as $D1, D2$, is converted in form of GSPN, through the application of the compound rules in Fig. 4.5, Fig. 4.7, Fig. 4.9, Fig. 4.10. This gate belongs to the DFT model in Fig. 4.4. The BE labelled as $D1$ becomes in the GSPN the place labelled as $D1\_dn$, and the timed transition labelled as $D1\_fail$; the BE labelled as $D2$ becomes the place labelled as $D2\_dn$, and the timed transition labelled as $D2\_fail$. The AND gate is converted into the immediate transition MS_and which fires when both $D1\_dn$ and $D2\_dn$ are marked, i. e. when all the input events of the gate have occurred. When MS_and fires, one token appears in the place MS_dn modelling the occurrence of the event MS.

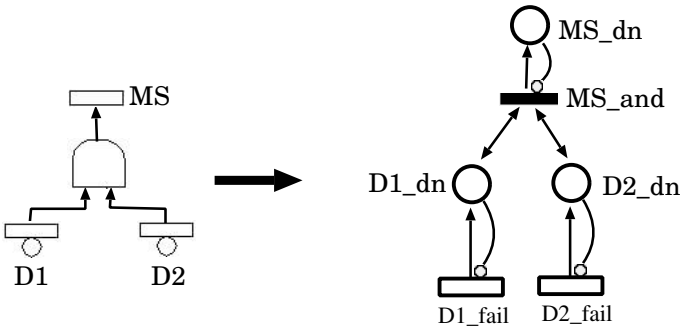

Figure 4.11: Conversion of a gate of type AND in GSPN form.

OR gate conversion

Fig. 4.12 shows the compound rule for a gate of type OR and one of its input events. r_s can be applied to a gate g of type OR such that

- its output event Y has already been mapped in the GSPN (by means of the compound rule in Fig. 4.5 or by means of the compound rule 4.6);
- the gate g has already been partially mapped in the GSPN through
- the input event X has already been mapped in the GSPN (by means of the compound rule in Fig. 4.5 or by means of the compound rule 4.7). X must be connected to g through an arc drawn as a continuous line.

Two variables are present: $< name1 >$ and $< name2 >$. They are instantiated to the label of Y and to the label of X, respectively.

The effect of r_s on the DFT is the removal of the arc (X, g) and its replacement with another arc from X to g, but drawn as a dashed line. In this way, we apply the rule in Fig. 4.12 to the same OR gate and to the same input event, only once.

r_t must be applied to a subnet of the GSPN composed by the following nodes:

- a place P whose label is equal to $< name1 > +$"$\_dn$"; this means that P is the mapping of Y in the GSPN;
- a place Q whose label is equal to $< name2 > +$"$\_dn$"; this means that Q is the mapping of X in the GSPN.

The effect of r_t on the GSPN is the addition of an immediate transition T and of several oriented arcs ((T, Q), (Q, T), (T, P)) and one inhibitor arc ((P, T)).

Example. In Fig. 4.13, we can see how a gate of type OR whose output event is labelled as DA and its input events are labelled as $DBUS$, UPD, MS, is converted in form of GSPN, through the application of the compound rules in Fig.

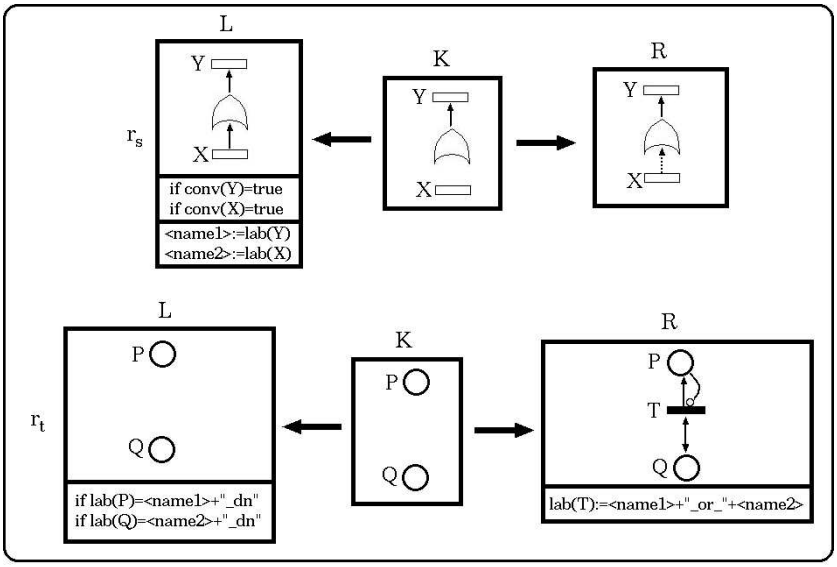

Figure 4.12: Compound rule for a gate of type OR and one of its input events.

4.5, Fig. 4.7, Fig. 4.12. This gate belongs to the DFT model in Fig. 4.4. The IEs labelled as DA and UPD become in the GSPN the places labelled as DA_dn and UPD_dn; the IE MS was already mapped in the place MS_dn during the conversion of the gate in Fig. 4.11. The BE labelled as $DBUS$ is converted into the subnet whose nodes are the place labelled as $DBUS\_dn$ and the timed transition labelled as $DBUS\_fail$. For each place corresponding to an input event of the OR gate ($DBUS\_dn$, UPD_dn, MS_dn), an immediate transition is created. For instance, the transition labelled as DA_or_DBUS puts one token inside the place DA_dn, as soon as $DBUS\_dn$ becomes marked. So, one token appears in DA_dn as soon as one of the places $DBUS\_dn$, UPD_dn, MS_dn, becomes marked, in other words, when one of the gate input events occurs.

4.6.3 Dynamic gates conversion

$FDEP$ gate conversion

Fig. 4.14 shows the compound rule for a gate of type $FDEP$ and one of its dependent events. r_s can be applied to a gate g of type $FDEP$ such that

- its trigger event T has already been mapped in GSPN (by means of the compound rule in Fig. 4.5 or by means of the compound rule 4.7); T is identified by the presence of an arc (T, g) (drawn as a continuous line);
- the dependent event D has already been mapped in the GSPN (by means of the compound rule in Fig. 4.5 or by means of the compound rule 4.7). D is

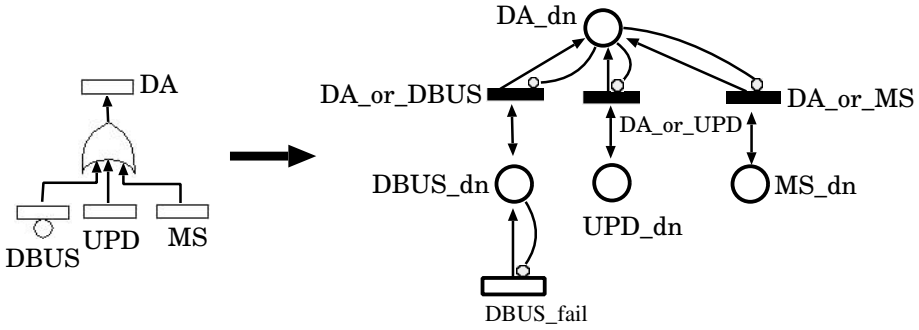

Figure 4.13: Conversion of a gate of type OR in GSPN form.

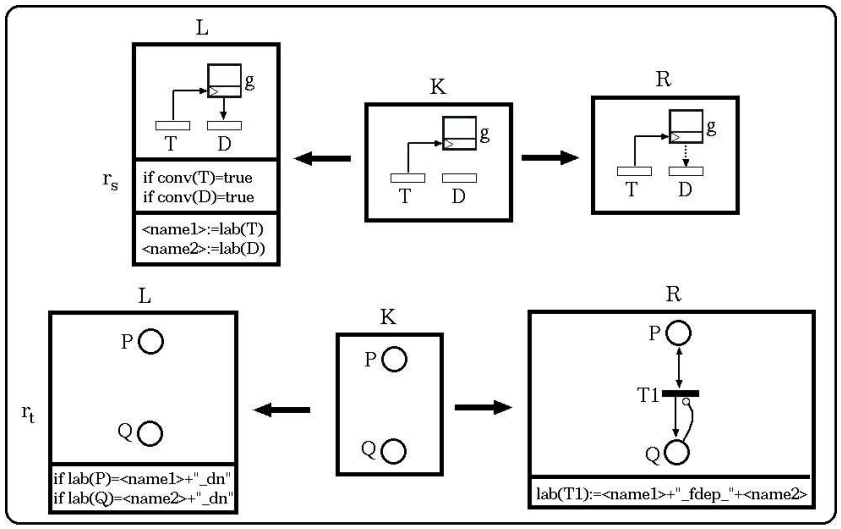

Figure 4.14: Compound rule for a gate of type $FDEP$ and one of its dependent events.

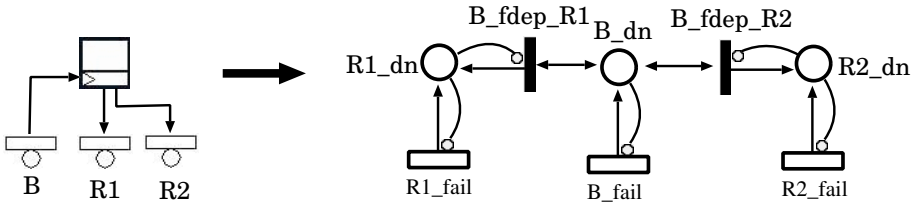

Figure 4.15: Conversion of a gate of type $FDEP$ in GSPN form.

identified by the presence of an arc (g, D) (drawn as a continuous line).

Two variables are present: $< name1 >$ and $< name2 >$. They are instantiated to the label of T and to the label of D, respectively.

The effect of r_s on the DFT is the removal of the arc (g, D) and its replacement with another arc from g to D, but drawn as a dashed line. In this way, we apply the rule in Fig. 4.14 to the same $FDEP$ gate and to the same dependent event, only once.

r_t must be applied to a subnet of the GSPN composed by the following nodes:

- a place P whose label is equal to $< name1 >$ +"$\_dn$"; this means that P is the mapping of T in the GSPN;
- a place Q whose label is equal to $< name2 >$ +"$\_dn$"; this means that Q is the mapping of D in the GSPN.

The effect of r_t on the GSPN is the addition of an immediate transition $T1$ and of several oriented arcs ($(T1, P)$, $(P, T1)$, $(T1, Q)$) and one inhibitor arc ($(Q, T1)$).

Example. In Fig. 4.15, we can see how a gate of type $FDEP$ having the BE labelled B as trigger event, and the BEs labelled $R1, R2$ as dependent events, is converted in form of GSPN, through the application of the compound rules in Fig. 4.7 and in Fig. 4.14. This gate belongs to the DFT in Fig. 4.4. The BE labelled as B is mapped to the place labelled as B_dn and to the timed transition B_fail; the BE labelled as $R1$ is mapped to the place labelled as $R1\_dn$ and to the timed transition $R1\_fail$; the BE labelled as $R2$ is mapped to the place labelled as $R2\_dn$ and to the timed transition $R2\_fail$. For each place corresponding to a dependent event of the $FDEP$ gate ($R1\_dn$, $R2\_dn$), an immediate transition is created. Such transitions fire as soon as the place labelled as B_dn (modelling the occurrence of the trigger event) becomes marked, with the effect of putting one token inside $R1\_dn$, $R2\_dn$ (modelling the occurrence of the dependent events), unless these places are already marked (the dependent events may have already occurred for another cause).

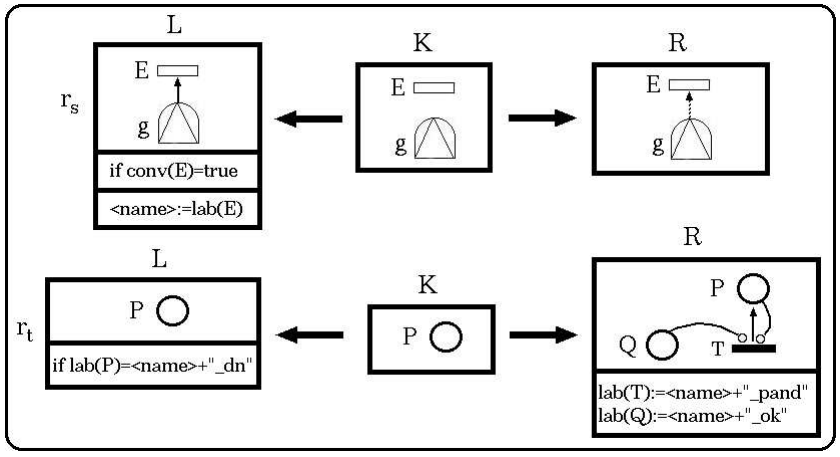

Figure 4.16: Compound rule for a gate of type $PAND$ and its output event.

$PAND$ gate conversion

The mapping of a gate of type $PAND$ is realized by means of three compound rules shown in Fig. 4.16, Fig. 4.17, Fig. 4.18, respectively.

Fig. 4.16 shows the compound rule for a gate of type $PAND$ and its output event. r_s can be applied to a gate g of type $PAND$ such that its output event E has already been mapped in the GSPN (by means of the compound rule in Fig. 4.5 or by means of the compound rule 4.6). The output event E is identified through the arc connecting g to E, drawn as a continuous line. The variable $< name >$ is present and is instantiated to the label of E.

The effect of r_s on the DFT, is the removal of the arc (g, E), where g is the gate of type AND. Such arc is replaced with another arc still connecting g to E, but drawn as a dashed line. In this way, the rule in Fig. 4.16 is applied to the same $PAND$ gate and the same output event, only once.

r_t can be applied to the place P such that its label is given by $< name >$ +"$\_dn$"; this means that P is the place generated by the mapping of E in the GSPN. The place Q and the immediate transition T are added to the GSPN by means of r_t, together with one oriented arc $((T, P))$ and two inhibitor arcs $((Q, T)$, $(P, T))$. The label of T is $< name >$ +"$\_pand$"; the label of Q is $< name >$ +"$\_ok$".

Fig. 4.17 shows the compound rule for a gate of type $PAND$ and two input events connected to the gate by means of two arcs having the order numbers 1 and 2 respectively. r_s can be applied to a gate g of type $PAND$ such that

- its output event Y has already been mapped in the GSPN (by means of the compound rule in Fig. 4.5 or by means of the compound rule 4.6);
- the gate g has already been partially mapped in the GSPN through the rule in Fig. 4.16 (dashed arc connecting g to Y).

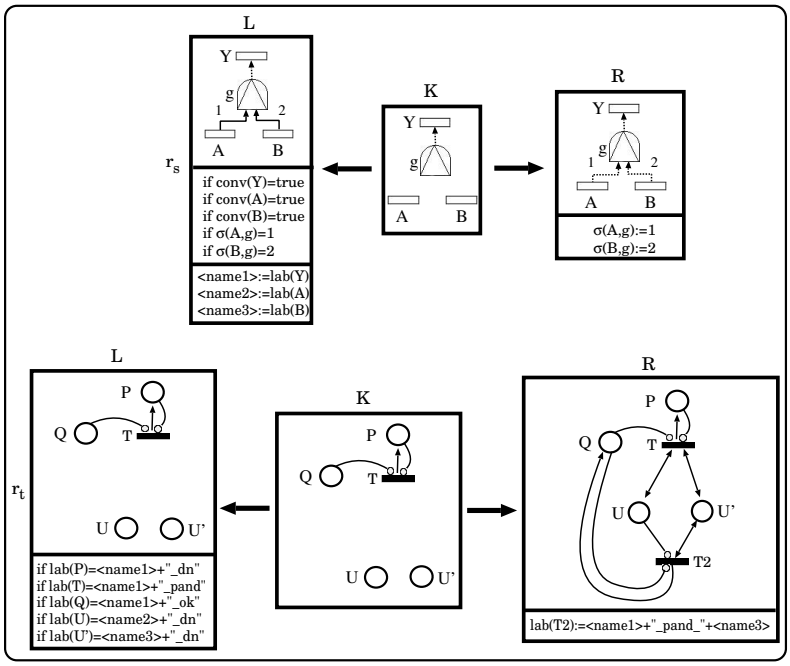

Figure 4.17: Compound rule for a gate of type $PAND$ and its input events with order number 1 and 2.

- the input events A and B have already been mapped in the GSPN (by means of the compound rule in Fig. 4.5 or by means of the compound rule 4.7). The event A must be connected to g through an arc (A, g) (drawn as a continuous line) such that $\sigma((A, g)) = 1$; B must be connected to g through an arc (B, g) (drawn as a continuous line) such that $\sigma((B, g)) = 2$.

Three variables are present: $< name1 >$, $< name2 >$ and $< name2 >$. They are instantiated to the label of Y, to the label of A and to the label of B, respectively.

The effect of r_s on the DFT is the removal of the arcs (A, g) and (B, g) and their replacement with other arcs, from A to g (with order number 1) and from B to g (with order number 2) respectively; the new arcs are drawn as a dashed line. In this way, we apply the rule in Fig. 4.17 to the same AND gate and to the same input events connected to the gate with arcs having order number 1 and 2, only once.

r_t must be applied to a subnet of the GSPN composed by the following nodes:

- a place P whose label is equal to $< name1 >$ +"$\_dn$"; this means that P is the mapping of Y in the GSPN;

- a place Q whose label is equal to $< name1 >$ +"$\_ok$";

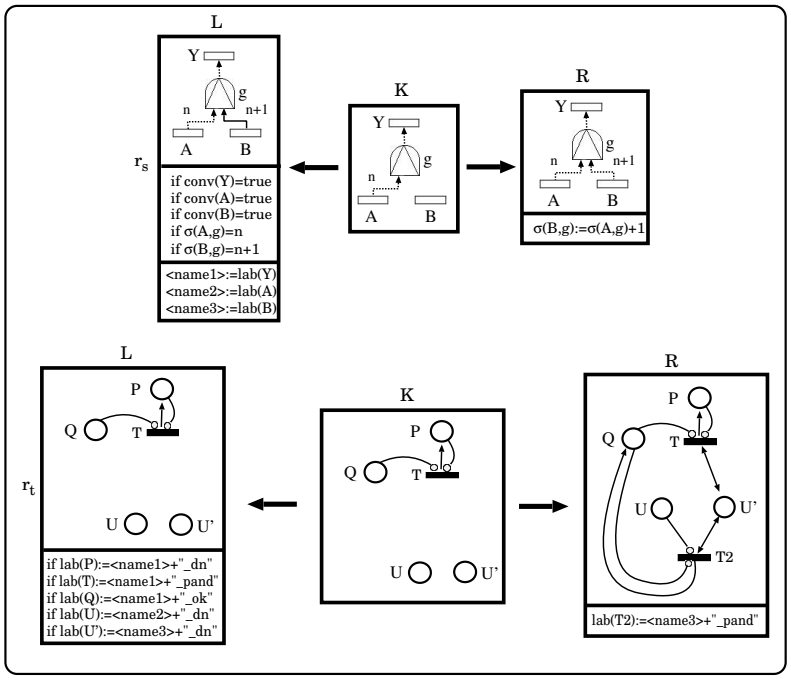

Figure 4.18: Compound rule for a gate of type $PAND$ and its input events with order number n and $n+1$ ($n \geq 2$).

- a place U whose label is equal to $< name2 >$ +"$\_dn$"; this means that U is the mapping of A in the GSPN;
- a place U' whose label is equal to $< name3 >$ +"$\_dn$"; this means that U' is the mapping of B in the GSPN;
- an immediate transition T whose label is $< name1 >$ +"$\_pand$"; this means that this transition was created by the application of the rule in Fig. 4.16 to g and Y.

The effect of r_t on the GSPN is the addition of the immediate transition $T2$, of several oriented arcs $((T,U), (U,T), (T,U'), (U',T), (U',T2), (T2,U'), (T2,Q))$, and of several inhibitor arcs $((U,T2),\ (Q,T2))$. The label of $T2$ is given by $< name1 >$ +"$\_pand\_$"+ $< name3 >$.

Fig. 4.18 shows the compound rule for a gate of type $PAND$ and two input events connected to the gate by means of two arcs having the order numbers n and $n+1$ respectively ($n \geq 2$).

Example. In Fig. 4.19, we can see how a gate of type $PAND$ having the event labelled as UPD as output event, and the events labelled as $BD, D1$ as input, is converted in form of GSPN, through the application of the compound rules in Fig.

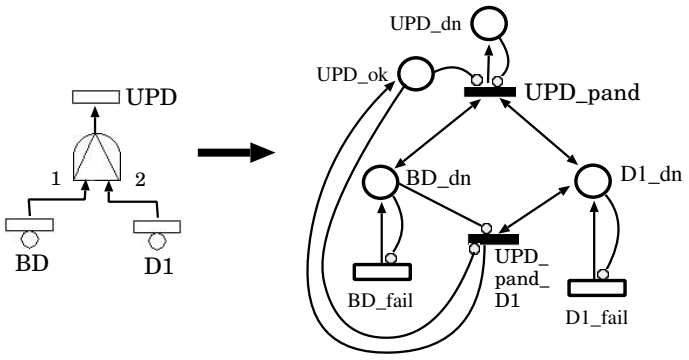

Figure 4.19: Conversion of a gate of type $PAND$ in GSPN form.

4.5, in Fig. 4.7, in Fig. 4.16, and in Fig. 4.17. This gate belongs to the DFT in Fig. 4.4. The BE labelled as BD is mapped to the place labelled as BD_dn and the timed transition labelled as BD_fail. The BE labelled as $D1$ was already mapped in the GSPN, during the conversion of the gate in Fig. 4.11.

The $PAND$ gate is mapped to a set of immediate transitions with the purpose of verifying that the specified order of the input events is respected: the transition UPD_pand_D1 fires if $D1\_dn$ becomes marked and BD_dn is empty, in other words if the event $D1$ occurs before the event BD. One token appears in the place labelled as UPD_ok after the firing of UPD_pand_D1, meaning that the specified order has not been respected. In general, when an input event occurs, we verify if its predecessor in the specified failure order, has already occurred or not.

When both BD_dn and $D1\_dn$ are marked, only if the place labelled as UPD_ok is not marked, one token appears in the place labelled as UPD_dn (the mapping in the GSPN of the output event UPD), by means of the immediate transition UPD_pand. The transition UPD_pand verifies the $PAND$ gate condition about the occurrence of all the input events, while the transition UPD_pand_D1 verifies the condition about the order of occurrence of the input events of the gate (section 4.2).

SEQ gate conversion

The mapping of a gate of type SEQ is realized by means of the compound rules shown in Fig. 4.20 and in Fig. 4.21.

Fig. 4.20 shows the compound rule for a gate of type SEQ and two input events connected to the gate by means of two arcs having the order numbers 1 and 2 respectively. r_s can be applied to a gate g of type SEQ such that the input events A and B have already been mapped in the GSPN (by means of the compound rule 4.7). A must be connected to g through an arc (A, g) (drawn as a continuous line) such that $\sigma((A, g)) = 1$; B must be connected to g through an arc (B, g) (drawn as a continuous line) such that $\sigma((B, g)) = 2$.

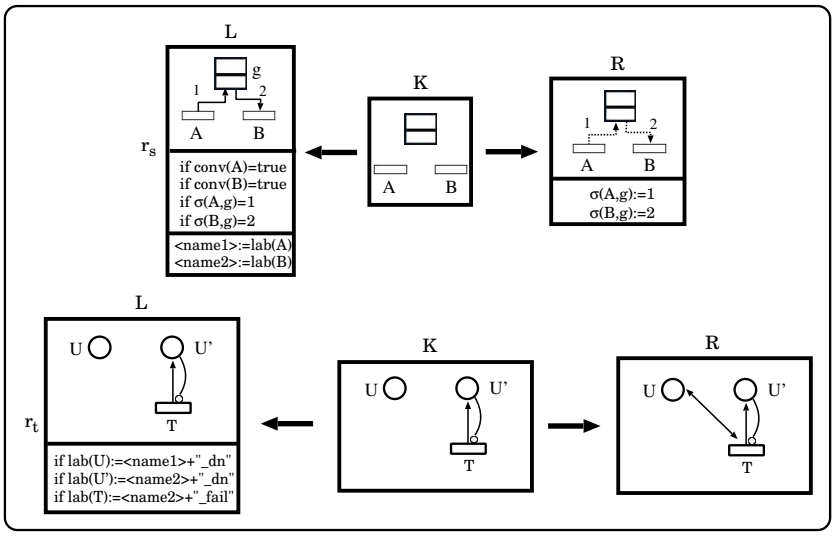

Figure 4.20: Compound rule for a gate of type SEQ, its trigger event (with order number 1) and the first of its dependent events (with order number 2).

Two variables are present: $< name1 >$ and $< name2 >$. They are instantiated to the label of A and to the label of B, respectively.

The effect of r_s on the DFT is the removal of the arcs (A, g) and (B, g) and their replacement with other arcs, from A to g (with order number 1) and from B to g (with order number 2) respectively; the new arcs are drawn as a dashed line. In this way, we can apply the rule in Fig. 4.17 to the same SEQ gate and to the same input events connected to the gate with arcs having order number 1 and 2, only once.

r_t must be applied to a subnet of the GSPN composed by the following nodes:

- a place U whose label is equal to $< name1 > +$"$\_dn$"; this means that U is the mapping of A in the GSPN;
- a place U' whose label is equal to $< name2 > +$"$\_dn$"; this means that U' is the mapping of B in the GSPN;
- a timed transition T whose label is $< name2 > +$"$\_fail$"; this means that this transition represents the occurrence of the event B in the DFT. B was originally a BE, but it has been converted to an IE by the previous application of the rule in Fig. 4.7.

The effect of r_t on the GSPN is the addition of a couple of oriented arcs: (U, T), (T, U).

Fig. 4.21 shows the compound rule for a gate of type SEQ and two input events connected to the gate by means of two arcs having the order numbers n and $n + 1$ respectively ($n \geq 2$).

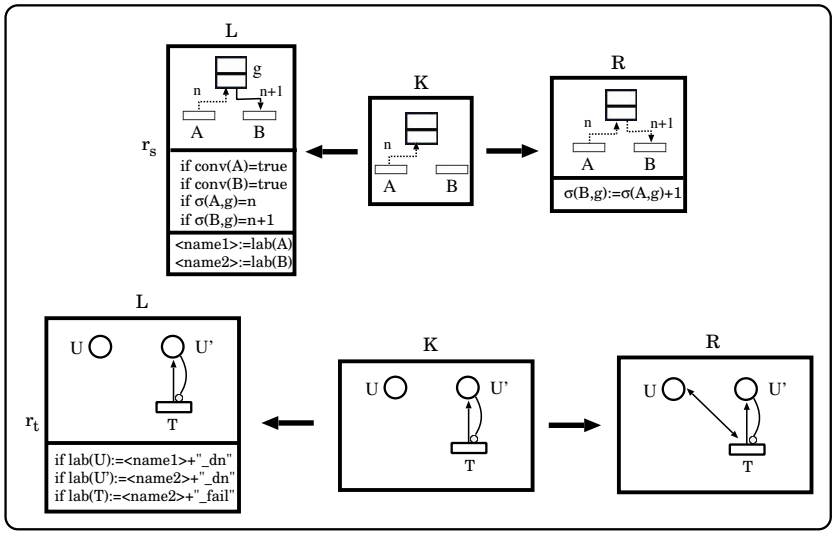

Figure 4.21: Compound rule for a gate of type SEQ and its dependent events with order number n and $n+1$ ($n \geq 2$).

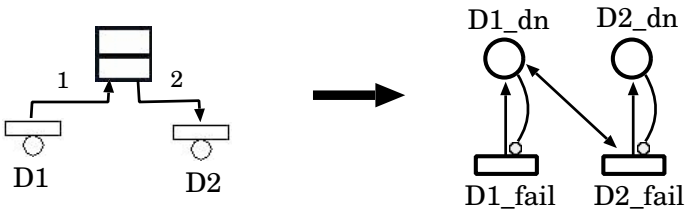

Figure 4.22: Conversion of a gate of type SEQ in GSPN form.

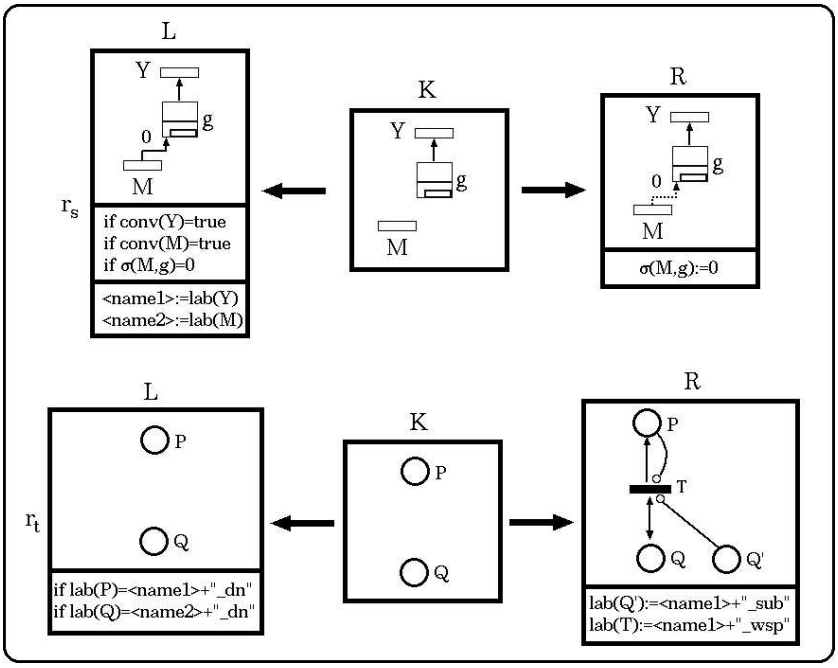

Figure 4.23: Compound rule for a gate of type WSP, its output event and its input event relative to the main component.

Example. Fig. 4.22 shows the conversion in the GSPN form of a gate of type SEQ with the BE labelled as $D1$ as trigger event, and the BE labelled as $D2$ as dependent event. This gate belongs to the DFT model in Fig. 4.4. The conversion is performed by means of the rule in Fig. 4.20. $D1$ and $D2$ were already mapped in the GSPN, during the conversion of the gate in Fig. 4.11. In the GSPN resulting from the SEQ gate mapping, the timed transition $D2\_fail$ (modelling the occurrence of the BE $D2$), can fire only if the place $D1\_dn$ is marked, i. e. if the BE $D1$ has already occurred. In this way, the events connected to the SEQ gate are forced to fail in the specified order, in the equivalent GSPN.

WSP gate conversion

The mapping of a gate of type WSP is realized by means of two compound rules shown in Fig. 4.23 and in Fig. 4.24 respectively. Fig. 4.23 shows the compound rule for a gate g of type WSP, its output event Y and its input event M relative to a main component. M is identified by an arc (M, g) having order number 0. r_s can be applied to a gate g of type WSP such that

- the input event M has already been mapped in the GSPN (by means of the compound rule in Fig. 4.7). M must be connected to g through an arc (M, g) (drawn as a continuous line) such that $\sigma((M, g)) = 0$;

- the input event Y has already been mapped in the GSPN (by means of the compound rule in Fig. 4.5 or by means of the compound rule in Fig. 4.6).

Two variables are present: $< name1 >$ and $< name2 >$. They are instantiated to the label of Y and to the label of M, respectively.

The effect of r_s on the DFT is the removal of the arc (M, g) and its replacement with another arc from M to g (with order number 0); the new arc is drawn as a dashed line. In this way, we can not apply again the rule in Fig. 4.23 to the same WSP gate, the same output event and to the same input event relative to the main component.

r_t must be applied to a subnet of the GSPN composed by the following nodes:

- a place P whose label is equal to $< name1 > +$"$\_dn$"; this means that P is the mapping of Y in the GSPN;

- a place Q whose label is equal to $< name2 > +$"$\_dn$"; this means that Q is the mapping of M in the GSPN.

The effect of r_t on the GSPN is the addition of the place Q', of the immediate transition T, of three oriented arcs ((T, P), (T, Q), (Q, T)) and of two inhibitor arcs ((P, T), (Q', T)). The label of Q' is given by $< name2 > +$"$\_sub$"; the label of T is given by $< name1 > +$"$\_wsp$".

Fig. 4.24 shows the compound rule for a gate g of type WSP, and an input event S relative to a spare component. This rule considers also the output event Y of g and its input event M relative to the main component. M is identified by an arc (M, g) having order number 0. S is identified by an arc (S, g) having order number $n > 0$. r_s can be applied to a gate g of type WSP such that

- the input event S has already been mapped in the GSPN (by means of the compound rule in Fig. 4.7). S must be connected to g through an arc (S, g) (drawn as a continuous line) such that $\sigma((M, g)) = 0$;

- g must have already been partially mapped in the GSPN by means of the rule in Fig. 4.23; this property holds if the arc (M, g) is drawn as a dashed line.

Several variables are present:

- $< name1 >$ is the label of Y;

- $< name2 >$ is the label of M;

- $< name3 >$ is the label of S;

- $< num\_s >$ is set to the number of input events of g ($| \bullet g|$);

- $< num\_a >$ is set to the order number of the arc connecting S to g ($\sigma((S, g))$).

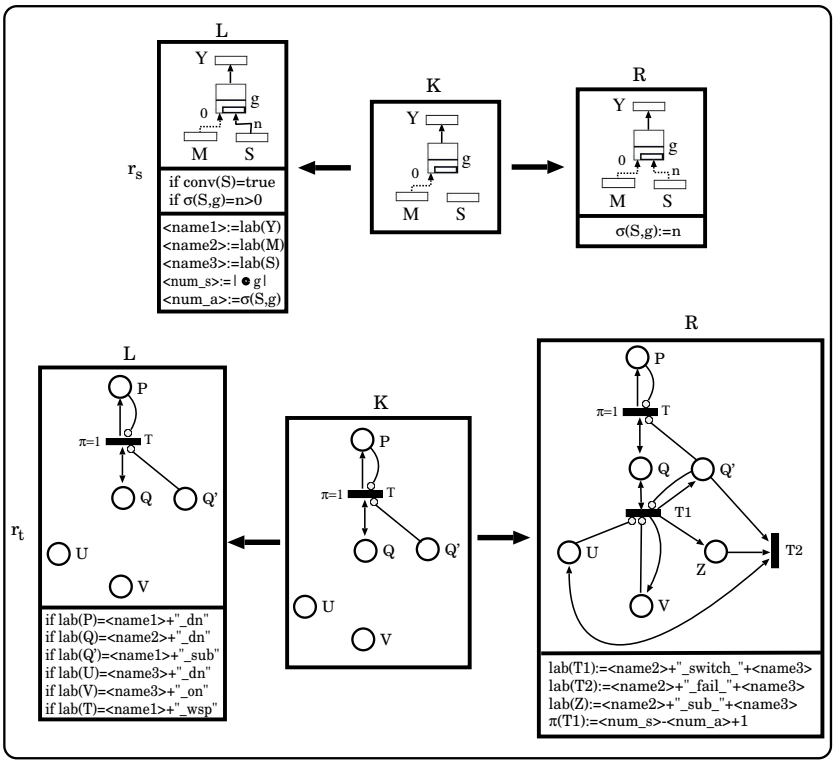

Figure 4.24: Compound rule for a gate of type WSP and an input event relative to a spare component.

The effect of r_s on the DFT is the removal of the arc (S, g) and its replacement with another arc from S to g (with the same order number of the removed one); the new arc is drawn as a dashed line. In this way, we can not apply again the rule in Fig. 4.24 to the same WSP gate and the same input event relative to a spare component.

r_t must be applied to a subnet of the GSPN composed by the following nodes:

- a place P whose label is equal to $< name1 > +$"$\_dn$"; this means that P is the mapping of Y in the GSPN;
- a place Q whose label is equal to $< name2 > +$"$\_dn$"; this means that Q is the mapping of M in the GSPN;
- a place Q' whose label is equal to $< name2 > +$"$\_sub$"; this place was generated by the application of the compound rule in Fig. 4.23 to g, Y and M;
- a place U whose label is equal to $< name3 > +$"$\_dn$"; this place was generated by the application of the compound rule in Fig. 4.8 to g and S;
- a place V whose label is equal to $< name3 > +$"$\_on$"; this place was generated by the application of the compound rule in Fig. 4.8 to g and S;
- the immediate transition T whose label is $< name1 > +$"$\_wsp$"; this place was generated by the application of the compound rule in Fig. 4.23 to g, Y and M.

The effect of r_t on the GSPN is the addition of the place Z, of the immediate transitions $T1$ and $T2$, of several oriented arcs ($(T1, Q')$, $(T1, V)$, $(T1, Z)$, $(T1, Q)$, $(Q, T1)$, $(Q', T2)$, $(Z, T2)$, $(U, T2)$, $(T2, U)$) and inhibitor arcs ($(Q', T1)$, $(U, T1)$, $(V, T1)$). The label of Z is $< name2 > +$"$\_sub\_$"$+ < name3 >$"; the label of $T1$ is $< name2 > +$"$\_switch\_$"$+ < name3 >$; the label of $T2$ is $< name2 > +$"$\_fail\_$"$+ < name3 >$. The priority of the immediate transition $T1$ is set to $< num\_s > - < num\_a > +1$.

Example. Fig. 4.25 shows the conversion in the GSPN form of a gate of type WSP having $MEM1$ as output event, $M1$ as main component, $R1$ and $R2$ as spare components of $M1$. This gate belongs to the DFT model in Fig. 4.4. The conversion of this gate is realized by applying the compound rules in Fig. 4.7, in Fig. 4.8, in Fig. 4.23, in Fig. 4.24. In the equivalent GSPN, several timed transitions and places modelling the failure of the components, are present. The transition $M1\_fail$ models the occurrence of the failure of $M1$; when it fires, it puts one token in the place $M1\_dn$ modelling the failed state of $M1$. The place $R1\_on$ indicates if $R1$ is dormant ($m(R1\_on) = 0$) or if it is working ($m(R1\_on) = 1$).

The transition $R1\_off\_fail$ models the occurrence of the failure of $R1$ while it is in the dormant state; this transition can fire only if $m(R1\_on) = 0$. The

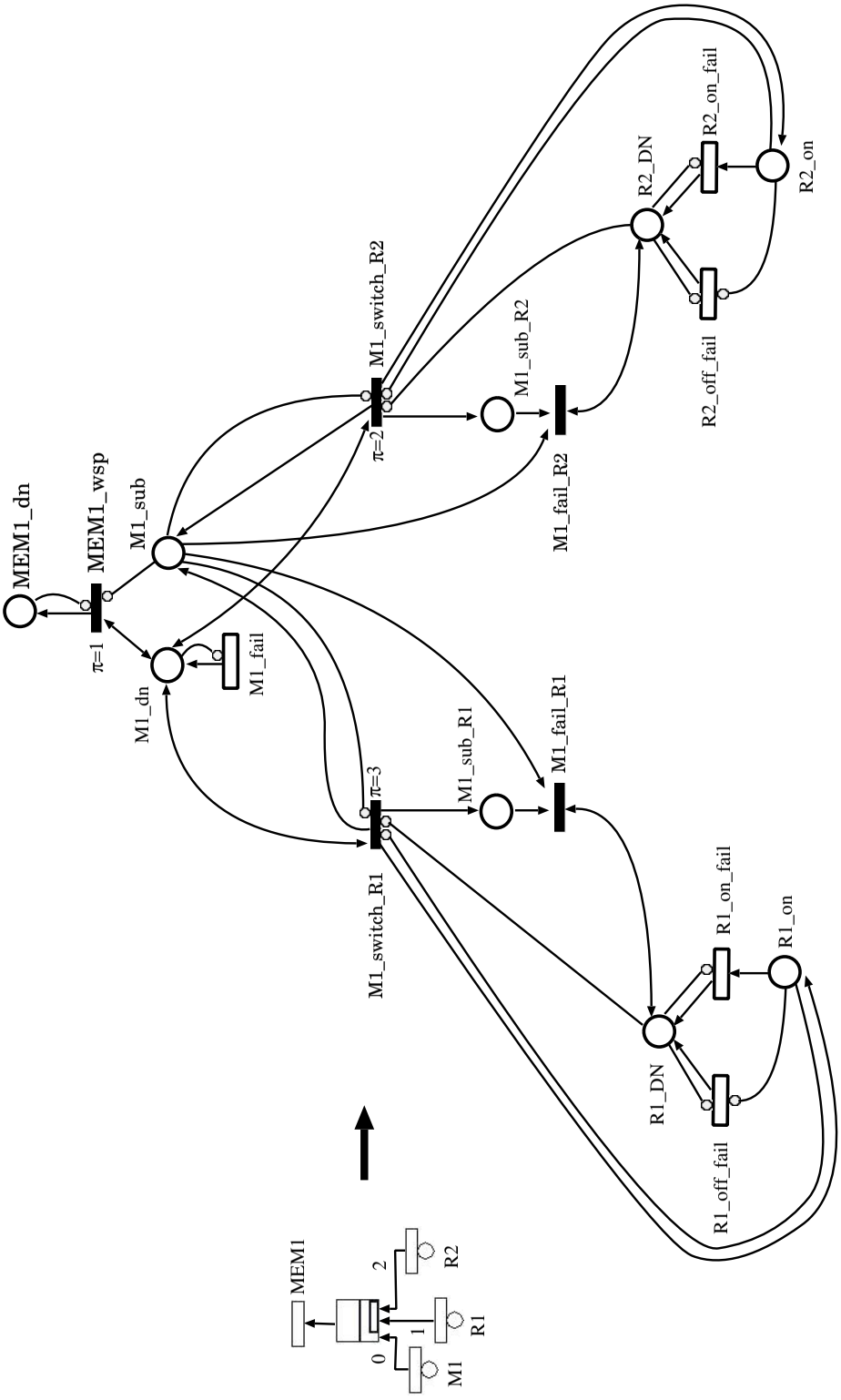

Figure 4.25: Conversion of a gate of type WSP in GSPN form.

transition $R1\_on\_fail$ models the occurrence of the failure of $R1$ while it is in the working state; this transition can fire only if $m(R1\_on) = 1$. The effect of the firing of both the transitions $R1\_off\_fail$ and $R1\_on\_fail$ is the appearance of one token in the place $R1\_dn$ modelling the failed state of $R1$. The failure of the spare component $R2$ is modelled in an analogous way by the places $R2\_on$, $R2\_dn$, $R2\_off\_fail$, $R2\_on\_fail$.

If $m(M1\_dn) = 1$ ($M1$ is failed), $m(M1\_sub) = 0$ ($M1$ is not currently replaced by any spare), $m(R1\_dn) = 0$ ($R1$ is not failed) and if $m(R1\_on) = 0$ ($R1$ is dormant), then the immediate transition $M1\_switch\_R1$ fires with the following effects: $m(R1\_on) = 1$ to indicate that $R1$ is working; $m(M1\_sub) = 1$ to indicate that $M1$ is currently replaced by any spare; $m(M1\_sub\_R1) = 1$ to indicate that $M1$ is currently replaced by the spare $R1$. In this way, the replacement of $M1$ by $R1$ is modelled in the GSPN. If later $R1\_dn$ becomes marked ($R1$ is failed), the immediate transition $M1\_fail\_R1$ fires removing the token inside $M1\_sub$ and $M1\_sub\_R1$.

If $m(M1\_dn) = 1$ ($M1$ is failed), $m(M1\_sub) = 0$ ($M1$ is not currently replaced by any spare), $m(R1\_dn) = 1 \vee m(R1\_on) = 1$ ($R1$ is failed or is replacing another main component), $m(R2\_dn) = 0$ ($R2$ is not failed), and if $m(R2\_on) = 0$ ($M1$ is dormant), then the immediate transition $M1\_switch\_R2$ fires with the following effects: $m(R2\_on) = 1$ to indicate that $R2$ is working; $m(M1\_sub) = 1$ to indicate that $M1$ is currently replaced by any spare; $m(M1\_sub\_R2) = 1$ to indicate that $M1$ is currently replaced by the spare $R2$. In this way, the replacement of $M1$ by $R2$ is modelled in the GSPN. If later $R2\_dn$ becomes marked ($R2$ is failed), the immediate transition $M1\_fail\_R2$ fires removing the token inside $M1\_sub$ and $M1\_sub\_R2$.

If $m(M1\_dn) = 1$ ($M1$ is failed), $m(M1\_sub) = 0$ ($M1$ is not currently replaced by any spare), $m(R1\_dn) = 1 \vee m(R1\_on) = 1$ ($R1$ is failed or is replacing another main component), $m(R2\_dn) = 1 \vee m(R2\_on) = 1$ ($R2$ is failed or is replacing another main component), then the immediate transition $MEM1\_wsp$ fires putting one token inside $MEM1\_dn$; this place is the mapping in the GSPN of the output event $MEM1$ of the gate in Fig. 4.25. In this way, we model the occurrence of the output event of the gate, in the case the main component $M1$ is failed and there is no available spare components to replace it in its function.

Actually, the immediate transitions $M1\_switch\_R1$, $M1\_switch\_R2$, $M1\_wsp$ may be enabled to fire at the same time; their priority values determine which transition has to fire.

4.6.4 Conversion steps

Given a DFT model to be mapped in a GSPN, the only compound rules which can be initially applied, concern the events; these rules are in Fig. 4.5, Fig. 4.6, Fig. 4.7, Fig. 4.8. The TE and the IEs are simply mapped into places of the GSPN, while the conversion of the BEs generates also timed transitions modelling their occurrence. A compound rule for a non internal event, maps the event in the

GSPN, and replaces in the DFT the event with an internal event. This replacement is useful because in this way, all the input and output events are IEs; so, we have to define compound rules only for gates having IEs as input and output events, avoiding to distinguish the events according to their type (BEs, IEs, TE) in the compound rules for the gates mapping.

The mapping of an event in the GSPN, is independent from the type of gate the event is input or output of, with the exception of the case of an event relative to a spare component. For this reason, in the rules in Fig. 4.5, Fig. 4.6, Fig. 4.7 no gates are present in the source graph transformation rule (r_s). In the special case of an event relative to a spare component, we have to apply the rule in Fig. 4.8, where the WSP gate appears. The priorities established among all these rules, allow to apply the correct rule to each event of the DFT.

The mapping of a gate in the GSPN target model can be performed only when the mapping of the input events and the output event of the gate, has been performed. The conversion of a gate generates immediate transitions, with the possible creation in the GSPN of other places necessary to model in the GSPN the gate semantic.

Several compound rules may be enabled at the same time; the final result of the DFT mapping into GSPN is not influenced by the order of application of the compound rules (confluence property (section 4.5.2)). However, priorities among compound rules must be respected in any order of application.

The use of the attribute *conv* for the events, and the replacement of some arcs drawn as continuous lines, with arcs drawn as dashed lines, is a way to allow to perform the mapping of an event or a gate only once. If we did not use this trick, it would be possible to apply the same rule to the same event or gate for an infinite number of times. Moreover, the compound rules have been expressed in a such a way that whenever the source graph transformation rule can be applied to the DFT, the target transformation rule can be applied as well. In other words, whenever a match for L_s exists in the DFT, a match for L_t exists in the GSPN.

Since the DFT is composed by a finite number of nodes, and each event and gate is mapped in the GSPN only once, the conversion process ends after a finite number of steps (termination property (section 4.5.2)). More precisely, the conversion process ends when no rules can be applied to the DFT target model.

4.6.5 Running example

Through the application of the transformation rules described in section 4.6, we can convert the DFT model in Fig. 4.4, into the GSPN in Fig. 4.26. The probability of the TE (Unreliability of the system) at time t can be computed on the GSPN model as the probability of the place TE_dn to be marked by one token at time t:

$$Pr\{TE, t\} = Pr\{m(TE\_dn) = 1, t\}$$

In the case of this example, and according to the failure rates and dormancy factors indicated in Tab. 4.1, the probability of the TE versus time, obtained on the GSPN,

time t	$Pr\{TE, t\}$
1000 h	2.347929E-06
2000 h	5.391436E-06
3000 h	9.130105E-06
4000 h	1.356352E-05
5000 h	1.869125E-05
6000 h	2.451289E-05
7000 h	3.102801E-05
8000 h	3.823620E-05
9000 h	4.613702E-05
10000 h	5.473005E-05

Table 4.3: Unreliability values for the Multiproc system with dependencies.

is shown in Tab. 4.3. These values have been computed by means of the *GreatSPN* tool; the state space derived from the GSPN by *GreatSPN*, is composed by 7806 states (7806 tangible markings in the reachability graph of the GSPN). The time to perform the GSPN analysis, has been 12 seconds[1].

4.7 Module based DFT analysis

The state space analysis of a DFT through its mapping into a GSPN model (or in a CTMC), is typically computationally expensive, since the number of states tends to grows exponentially with the number of components. Moreover, the state space analysis is strictly necessary for the subtrees containing dynamic gates, since we can not express in Boolean formulas the semantic of dynamic gates. At the same time, for the subtrees with only Boolean gates, the standard combinatorial analysis (through the BDD generation) would be enough. So, a way to reduce the computational cost of the state space analysis, consists of using this technique only with the subtrees with dynamic gates, and using the combinatorial technique with the rest of the DFT.

Such approach for the DFT analysis requires the solution of some subtrees in isolation; in order to analyze a subtree in isolation, the subtree must be independent from the rest of the DFT (section 2.5). So, the modules (independent subtrees) detection on a DFT becomes necessary to this aim.

In a FT, a module is a subtree which is structurally independent from the rest of the FT; this happens if the events contained in the subtree, do not occur elsewhere in the FT [44]. This notion of module is still valid on a DFT, but we need also a way to classify the modules according to the analysis technique they require (combinatorial or state space analysis).

[1]The GSPN analysis has been performed on a personal computer equipped with a Pentium 4 2.4 MHz processor and with 512MB of RAM.

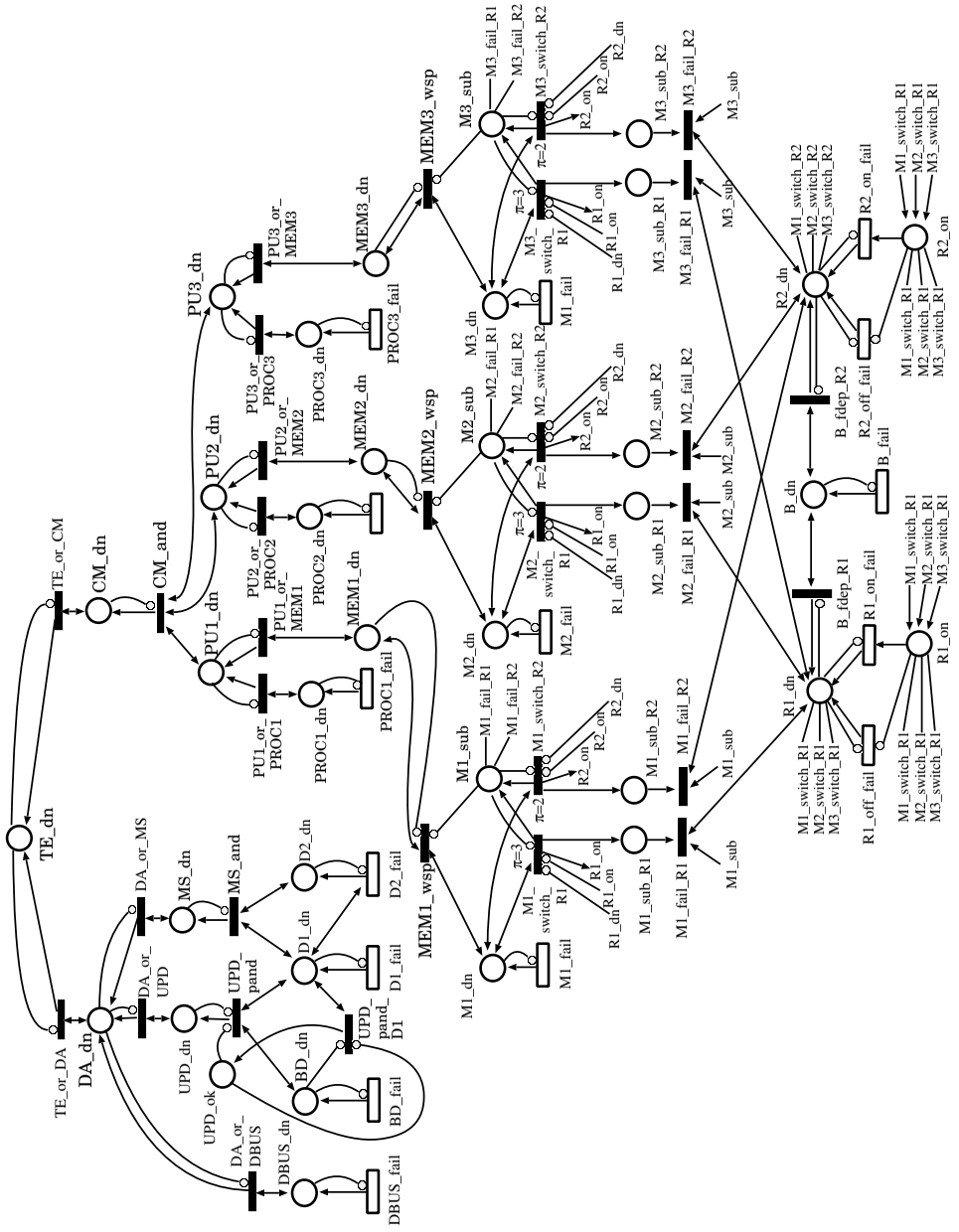

Figure 4.26: The GSPN model corresponding to the DFT in Fig. 4.4.

4.7.1 Modules detection and classification

The modules detection algorithm [44] for FT models proposed in section 2.5.2, has the purpose of verifying the independence of a subtree of the FT, in structural terms. This algorithm is still useful on DFT models, to verify the structural independence of the subtrees. However, this algorithm is applied on FTs interpreting the FT as a tree structure by reversing the orientation of the FT arcs; if we perform the reversing of the orientation of the DFT arcs, we can apply such algorithm to DFTs too.

By definition the BEs of a FT are modules; in the case of DFTs, we do not consider the BEs as modules. Moreover, in a DFT, a BE may not be a terminal node; for instance, the BE e may be the dependent event of a gate g of type $FDEP$; in this case, an arc (g, e) is present. So we have that $|\bullet e| \neq 0$; in other words e can be considered the root of a subtree $\widehat{e}$. However, in a DFT, we always consider a module as a subtree $\widehat{e'}$ rooted in an event $e' : e' \in \mathcal{IE} \cup \{TE\}$. In this sense, the modules detection algorithm verifies if an event e' is the root of a module, only if $e' \in \mathcal{IE} \cup \{TE\}$ (section 2.5.2).

After the modules detection through this algorithm, DFT modules can be classified according to the solution method they need:

- the module $\widehat{e}$ is a *Combinatorial Solution Module* (CSM) if $\forall g \in \mathcal{G} : g \in \circ e, \gamma(g) \in \mathcal{BG}$. In other words, the module $\widehat{e}$ is a CSM if it does not contain any dynamic gate.

- the module $\widehat{e}$ is *State space Solution Module* (SSM) if $\exists g \in \mathcal{G} : \gamma(g) \in \mathcal{DG} \wedge g \in \circ e$. In other words, the module $\widehat{e}$ is a SSM if it contains at least one dynamic gate.

For the CSMs the state space analysis is not necessary; for them, the less expensive combinatorial analysis is enough. The state space analysis is essential for the SSMs, since they contain dependencies due to the presence of dynamic gates [2]. We indicate with $\mathcal{M}$ the set of the modules in the DFT.

Given a dynamic gate g of the DFT, a *Dynamic Module* (DM) is the smallest SSM $\widehat{e}$ in the DFT such that g is contained in $\widehat{e}$. Formally, $\widehat{e} \in \mathcal{M}$ is a DM if

$$\exists g \in \mathcal{G} : \gamma(g) \in \mathcal{DG} \wedge g \in \circ e \wedge \nexists \widehat{e'} \in \mathcal{M} : g \in \circ e' \wedge e' \in \circ e$$

We indicate with $\mathcal{DM}$ the set of the DMs of a a DFT. A module $\widehat{e}$ is a *Maximal Dynamic Module* (MDM) if $\widehat{e}$ is a DM and is not contained inside another DM; formally, $\widehat{e} \in \mathcal{DM}$ is a MDM if

$$\nexists \widehat{e'} \in \mathcal{DM} : e \in \circ e'$$

The definition of DM and MDM is useful in the DFT analysis by modularization.

4.7.2 DFT modularization

The main reason to perform the DFT analysis by modularization, i. e. exploiting modules, is applying the state space analysis to the subtrees of the DFT requiring it, while the less expensive combinatorial analysis can be performed on the rest of DFT. We limit our attention on the quantitative analysis of a DFT at a certain mission time t.

As in the case of the FTs (section 2.5.3), the main steps of the modularization of a DFT are: modules detection, modules classification, decomposition, modules analysis and aggregation. As described in the previous section, the modules detection algorithm for FTs, is still valid for the modules detection on DFTs. Still in the previous section, the modules classification is provided. In the decomposition step, we have to detach from the DFT, modules needing the state space analysis, in other words, we have to detach SSMs. The state space analysis of a SSM can be realized by mapping the DFT module in a GSPN as shown in section 4.6; then, the resulting GSPN can be analyzed returning the probability to be marked at time t, of the GSPN place corresponding to root event of the module.

In the aggregation step, we have to replace each of the detached SSM with a BE having such probability, instead of a failure rate. At this point a problem arises: if we iterate the modularization steps and a BE having a probability is part of a detached SSM to be converted in GSPN, we can not convert such BE in the GSPN, since it does not have a failure rate: in the model transformation system from DFT to GSPN (section 5.4), the compound rule to map a BE in GSPN form (Fig. 4.7) requires the BE to have a failure rate.

In order to avoid this drawback, we have to choose the SSMs to be detached in such a way to avoid that any BE replacing a SSM, belongs to another SSM. The solution to this problem, is choosing the MDMs of the DFT as the modules to be detached from the DFT. Any dynamic gate belongs to a DM and to a MDM. Choosing the MDMs as the modules to be detached and analyzed in isolation in the state space, all the dynamic gates in the DFT are included in one of the detached modules. After the aggregation step, no BEs replacing modules, will be part of a SSM, since no dynamic gates will be present in the DFT.

Following this strategy, the modularization steps do not need to be iterated, but it is necessary to perform them only once in order to analyze in the state space any subtree needing this kind of analysis. If the detached modules are MDMs, after the aggregation step, the DFT does not contain any dynamic gate, so it is now a FT and can be analyzed with the combinatorial technique (BDD generation). We can not map a BE having a probability instead of a failure rate in a GSPN, but a BDD can deal with such a BE.

If the whole DFT is a MDM, we convert the whole DFT in a GSPN, and we analyze the resulting GSPN computing the probability of the GSPN place TE_dn to be marked at the mission time t.

The steps to perform the modularization of a DFT follow:

1. Modules detection

2. Modules classification: for each module, we verify if it is a MDM.
3. if $\widehat{TE}$ is a MDM go to step 11, else go to step 4.
4. Decomposition: each MDM is detached from the DFT.
5. MDM conversion to GSPN: each MDM is converted to GSPN.
6. GSPN analysis: each MDM in GSPN form, is analyzed.
7. Aggregation: each detached MDM is replaced in the DFT, by a BE (we obtain a FT).
8. FT conversion to BDD
9. BDD quantitative analysis returning the final result.
10. end.
11. DFT conversion to GSPN.
12. GSPN analysis: the DFT in GSPN form, is analyzed returning the final result.
13. end.

4.7.3 Running example

In this section, we perform the quantitative analysis by modularization of the DFT in Fig. 4.4, for a mission time of $10000h$. Fig. 4.27 shows the MDMs of the DFT; they are $\widehat{DA}$ and $\widehat{CM}$. Both of them must be converted to GSPN; Fig. 4.28 shows the module $\widehat{DA}$ in GSPN form; Fig. 4.29 shows the module $\widehat{CM}$ in GSPN form.

The state space analysis of the module $\widehat{DA}$ and of the module $\widehat{CM}$ at time $t = 10000h$ returned these probabilities:
$Pr\{DA, t\} = Pr\{m(DA\_dn) = 1, t\} = 5.460599E - 5$
$Pr\{CM, t\} = Pr\{m(CM\_dn) = 1, t\} = 1.240670E - 7$
The state space size of the module $\widehat{DA}$, computed on the equivalent GSPN, is 14 states; the state space size of the module $\widehat{CM}$ is 487 states. In both cases, the time required to perform the analysis of the MDM in GSPN form, is less than one second. In section 4.6.5, the state space derived from the GSPN in Fig. 4.26 equivalent to the whole DFT in Fig. 4.4, was composed by 7806 states and required 12 seconds to be analyzed. Thus, it is evident the reduction of the computational cost if the DFT analysis is performed by modularization, instead of converting the whole DFT into GSPN.

The replacement of the MDMs with BEs having the probability indicated above, produces the DFT in Fig. 4.30; this DFT is actually a FT so we can generate and analyze the corresponding BDD. Such BDD is shown in Fig. 4.31, and its quantitative analysis returns the probability of the TE ($5.473005E - 5$).

Tab. 4.4 reports the probabilities of the MDMs and of the TE, versus time.

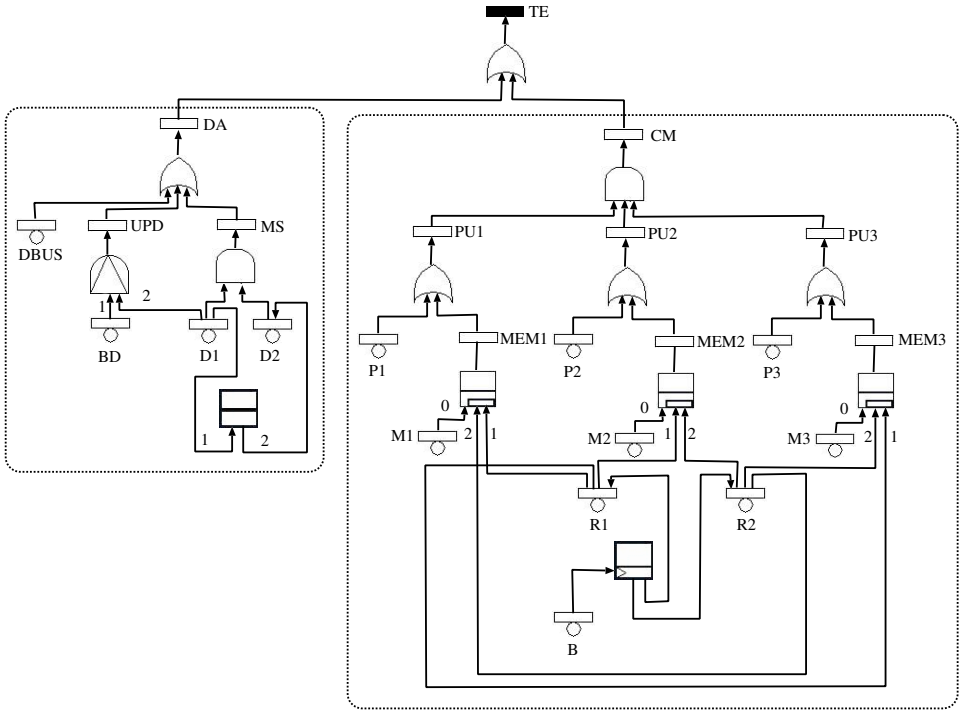

Figure 4.27: The MDMs in the DFT model of the Multiproc system.

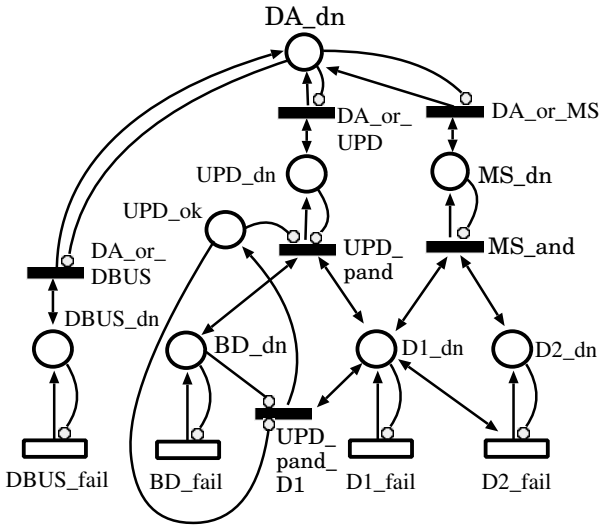

Figure 4.28: The GSPN corresponding to the module $\widehat{DA}$.

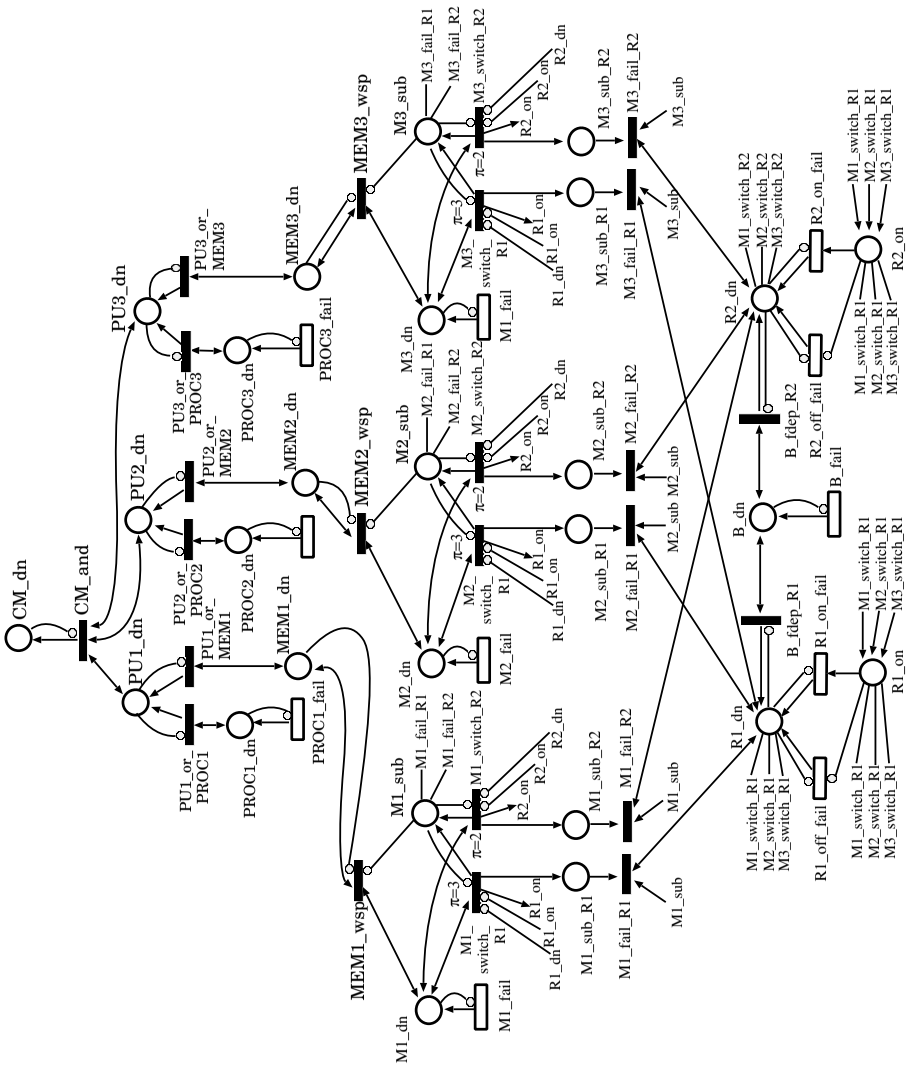

Figure 4.29: The GSPN corresponding to the module $\widehat{CM}$.

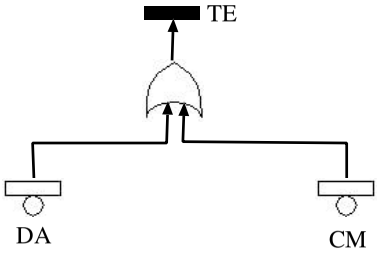

Figure 4.30: The DFT model of the Multiproc system after the MDMs analysis.

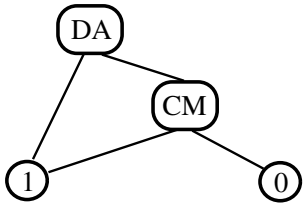

Figure 4.31: The BDD equivalent to the (D)FT in Fig. 4.30.

time t	$Pr\{DA, t\}$	$Pr\{CM, t\}$	$Pr\{TE, t\}$
1000 h	2.347804E-6	1.25E-10	2.347929E-06
2000 h	5.390438E-6	9.99E-10	5.391436E-06
3000 h	9.126738E-6	3.367E-9	9.130105E-06
4000 h	1.355554E-5	7.976E-9	1.356352E-05
5000 h	1.867569E-5	1.5567E-8	1.869125E-05
6000 h	2.448601E-5	2.6879E-8	2.451289E-05
7000 h	3.098536E-5	4.2651E-8	3.102801E-05
8000 h	3.817258E-5	6.3617E-8	3.823620E-05
9000 h	4.604651E-5	9.0513E-8	4.613702E-05
10000 h	5.460599E-5	1.24067E-7	5.473005E-05

Table 4.4: Unreliability values for the Multiproc system with dependencies.

Chapter 5

Modelling repair processes using RFT

5.1 Introduction to Repairable Fault Trees

As mentioned in section 3.1, a way to improve the Reliability of the system, consists of replicating its critical components or subsystems. Moreover, the Reliability can be improved by providing spare components able to replace a failed component in its function (section 4.2). Another way to improve the system Reliability is introducing repair or recovery processes with the purpose of repairing the failed components.

While the failure of a component determines the component state transition from the working state to the failed state, the accomplishment of the repair process allows a failed component to turn back to the working state. The behaviour of a non repairable component is acyclic; this means that the component can not turn back to a previous state: a non repairable component is initially working and may turn to the failed state; this state is an absorbing state meaning that no state transitions are possible from the failed state. In the case of a repairable component instead, the failed state is not an absorbing state because the repairable component can turn back to the working state due to the repair process; in other words, the behaviour of a repairable components is cyclic, in the sense that the working and the failed state may be repeatedly alternated during the life time of the component.

If the repair process involves a subsystem instead of a single component, the behaviour of the subsystem becomes cyclic. In general, we talk about repairable systems, when some repair process involves single components of the system, subsystems or the whole system.

Actually, when we deal with repairable components (systems), the term *Availability* should be used instead of Reliability, to indicate the probability that the component (system) is correctly working at a certain time (see section 1.1.3).

Besides the PFT (section 3.2) and the DFT formalism (section 4.3), a further extension of the FT formalism becomes necessary to model systems characterized

by the presence of repairable components or subsystems. To this aim, the *Repairable Fault Tree* (RFT) [32, 52] formalism has been developed together with its evaluation technique: the presence of repair processes in the system and in its RFT model, establishes some dependencies among the component failure events (state transitions from the working to the failed state) and the repair events (state transitions from the failed to the working state). Moreover, the model must be able to capture all the transient behaviour that happen whenever the failure of a repairable component (subsystem) is detected with the consequent activation of the repair action. The combinatorial analysis used for FTs can not fit these aspects, so the possibility of modelling repair processes requires the state space based analysis.

The RFT formalism differs from the FT formalism, for the introduction of a new primitive called *Repair Box* (RB) [8, 32]. A RB allows the model designer to represent the presence of a repair action involving a certain set of components. The repair action is activated by the occurrence of a specific failure event (trigger event) concerning a component or a subsystem. A repair process is characterized by a repair policy describing the repair mode, and by repair rates influencing the repair time; actually the time to repair a component can be considered as non deterministic, so the duration of a repair process is a random variable. In the RFT formalism, the time to repair is ruled by a negative exponential distribution (as the time to failure).

The usefulness of the RFT formalism does not regard only the possibility to model repair processes, but also the possibility to evaluate and compare the efficiency of several repair policies if applied to the same system.

A way to perform the state space analysis of the repair process involving a subsystem modelled in the RFT, consists of mapping the failure mode and the repair mode of the subsystem in a GSPN model and exploiting the available GSPN solvers. The way to perform such mapping is described in section 5.4.

An example of RFT model is depicted in Fig. 5.1.

5.2 A new primitive: the Repair Box

In the RFT formalism, a new primitive called Repair Box (RB) is introduced with the aim of indicating the presence of a repair process in the system. A RB graphically appears as a wrench inside a square, and is connected by means of oriented arcs to the events in the RFT. In particular, the event e connected to the RB b by means of the oriented arc (e, b) is called trigger event; the aim of the trigger event is twofold: its occurrence enables the repair action modelled by b, and it is the "root" of the subtree whose BEs are influenced by the repair action. Moreover, a RB b is connected to a set of BEs representing the components to be repaired; this set is called the basic coverage set of b and is indicated by $Cov_{BE}(b)$; given the BE e' belonging to $Cov_{BE}(b)$, b is connected to e' by means of the oriented arc (b, e'). The effect of the RB b is setting the value of the BEs in $Cov_{BE}(b)$ to $true$ (working), if their current value is $false$ (failed). This happens after random

period of time starting in the moment of activation of the RB, i. e. when the trigger event occurs.

Actually, the effect of the RB does not influence only the BEs in its basic coverage set, but also all the IEs whose value can be expressed by a Boolean function over a set of BEs including at least one BE in the basic coverage set of the RB: changing the value of a BE e' in the basic coverage set, may determine the change of the value of an IE e such that a path exists from e' to e according to the logic circuit orientation of the RFT arcs connecting input events to gates and gates to output events. The coverage set of the RB b is indicated by $Cov_E(b)$ and is composed by all the BEs and IEs whose value is influenced by the action of b. $Cov_E(b)$ is the union of $Cov_{BE}(b)$ with any IE e such that a path exists from any element of $Cov_{BE}(b)$ to e.

Besides the trigger event and the basic coverage set, a RB is characterized also by a *repair rate* and a *repair policy*. The repair rate is the parameter of the negative exponential distribution ruling the time to repair of the RB or the time to detect the failure (this depends on the repair policy). The repair policy describes each aspect of the repair process, such as the maximum number of components under repair at the same time, the order of repair of the components, etc. In some repair policies, a repair rate must be set for each BE in the basic coverage set.

We assume that an event can be the trigger of only one RB. All these aspects are formally defined in section 5.3, while in section 5.3.1 three repair policies are proposed.

5.3 RFT formalism definition

The RFT formalism in an extension of the FT formalism, with the addition of the RBs. The RFT formalism is given by the tuple
$\mathcal{RFT} = (\mathcal{E}, \mathcal{G}, \mathcal{RB}, \mathcal{A}, \mathcal{BG}, \gamma, \lambda, \mu, \mathcal{RP}, \beta)$
where:

- $\mathcal{E} = \mathcal{BE} \cup \mathcal{IE} \cup \{TE\}$ is the set of the events in the RFT; it is the union of the following sets:
  - $\mathcal{BE}$ is the set of the BEs;
  - $\mathcal{IE}$ is the set of the IEs;
  - {TE} is the set composed by the unique TE.
- $\mathcal{A} \subseteq (\mathcal{E} \times \mathcal{G}) \bigcup (\mathcal{G} \times \mathcal{E}) \bigcup (\mathcal{E} \times \mathcal{RB}) \bigcup (\mathcal{RB} \times \mathcal{BE})$ is the set of the arcs.
- $\mathcal{BG} = \{AND, OR\}$ is the set of Boolean gate types.
- $\gamma : \mathcal{G} \rightarrow \mathcal{BG}$ is the function assigning to each gate its type.
- Given $g \in \mathcal{G}$,
  $\bullet g = \{e \in \mathcal{E} : \exists (e, g) \in \mathcal{A}\}$ is the set of input events of g;
  $g \bullet = \{e \in \mathcal{E} : \exists (g, e) \in \mathcal{A}\}$ is the output event of g.

- Given $e \in \mathcal{E}$,
  $\bullet e = \{g \in \mathcal{G} : \exists(g, e) \in \mathcal{A}\}$ is the gate having e as output event;
  $e\bullet = \{g \in \mathcal{G} : \exists(e, g) \in \mathcal{A}\}$ is the set of gates having e as one of their input events.

- given $b \in \mathcal{RB}$,
  $\bullet b = \{e \in \mathcal{E} : \exists(e, b) \in \mathcal{A}\}$ is the trigger event of b;
  $b\bullet = \{e \in \mathcal{BE} : \exists(b, e) \in \mathcal{A}\} = Cov_{BE}(b)$ is the basic coverage set of b.

- The following conditions about the connection of events with gates, must hold:
  - $\forall g \in \mathcal{G}, |\bullet g| \geq 2$
  - $\forall g \in \mathcal{G}, |g \bullet| = 1$
  - $\forall e \in \mathcal{BE}, |\bullet e| = 0$
  - $\forall e \in \mathcal{BE}, |e \bullet| > 0$
  - $\forall e \in \mathcal{IE}, |\bullet e| = 1$
  - $\forall e \in \mathcal{IE}, |e \bullet| > 0$
  - $|\bullet TE| = 1$
  - $|TE \bullet| = 0$

- Given $e \in \mathcal{E}$, $\circ e = \{e' \in \mathcal{E} : \exists[e' \rightarrow e]\}$.

- Given $e \in \mathcal{E}$, $\widehat{e}$ is composed by any $[b \rightarrow e] : b \in \mathcal{BE} \cup \mathcal{BRE}$ ($\widehat{e}$ indicates the subtree rooted in e).

- The following conditions about the connection of events with RBs, must hold:
  - $\forall b \in \mathcal{RB}, |\bullet b| = 1$
  - $\forall b \in \mathcal{RB}, |b \bullet| \geq 1$
  - $\forall b, b' \in \mathcal{RB} : b \neq b', \bullet b \cap \bullet b' = \emptyset$

- Given $b \in \mathcal{RB}$,
  - $\circ b = \{e \in \mathcal{E} : \exists[e' \rightarrow e''] \wedge e' \in b \bullet \wedge e'' \in \bullet b \wedge e \in [e' \rightarrow e'']\}$
  - $b\circ = \{e \in \mathcal{E} : \exists[e' \rightarrow TE] \wedge e' \in \bullet b \wedge e \in [e' \rightarrow TE]\} - \bullet b$
    $Cov_E(b) = \circ b \cup b\circ$ is the coverage set of b.

- Given $b \in \mathcal{RB}$ and $e' \in \bullet b$, the following conditions hold:
  - $Cov_{BE}(b) \subseteq \circ e'$
  - $\circ b \subseteq \circ e'$

- $Cov_{BE}(b) \subseteq Cov_E(b)$

- $\lambda : \mathcal{BE} \rightarrow \mathbb{R}^+$ is the function assigning to each BE a failure rate, assuming that BEs are ruled by the negative exponential distribution.

- $\mu : \mathcal{RB} \rightarrow \mathbb{R}^+$ is the function returning the repair rate of a RB. The repair rate of the RB rules the time to repair (or to detect) the failure according to the repair policy associated with the RB. Some repair policies require also the specification of a repair rate for each BE in the basic coverage set (see section 5.3.1).

- $\mathcal{RP} = \{GRT, SRT - I, SRT - F, \ldots\}$ is the set of repair policies. Some of them are described in section 5.3.1.

- $\beta : \mathcal{RB} \rightarrow \mathcal{RP}$ is the function assigning a repair policy to a RB.

5.3.1 Defining a repair policy

In a repair policy, we set several aspects and parameters characterizing the repair mode of a subsystem. The first aspect to be defined in a repair policy is the *trigger condition*, i. e. the condition enabling the repair action.

The trigger condition can be

(a) a failure event;

(b) a condition expressed as a function of a set of failure events.

Besides the trigger condition, we can specify in a repair policy the following aspects concerning the execution of the repair action:

(a) the repair order of repairable components;

(b) the number of available repair facilities: this number determines how many components can be repaired at the same time.

Another aspect to be defined in a repair policy, is the time to repair a component (or a subsystem); the time to repair is a random variable; if it is ruled by a negative exponential distribution, then the parameter of the distribution is the *repair rate* of the component (subsystem), equal to $1/MTTR$, where $MTTR$ is the *Mean Time To Repair* of the component (subsystem). If the repair action involves a subsystem, the time to repair can be characterized in two ways:

(a) setting a global repair rate ruling the time to repair the subsystem entirely; the global repair rate might be *state dependent*, i.e. it might depend on the actual set of components to be repaired when the repair action is triggered;

(b) setting a repair rate for each component, so that the time to repair of each component is ruled separately.

Moreover, a random period of time may be necessary to detect the trigger condition enabling the repair action.

When the repair action involves a subsystem, another important aspect is the definition of when a repair action should be considered as completed:

(a) when all components in the subsystem have been repaired;

(b) as soon as the trigger condition is no more true.

A relevant aspect in the specification of a repair policy, is the influence of the repair action on the component failure processes:

(a) they may be stopped until repair ends;

(b) they may continue during the repair action so that new failures may occur while the repair is taking place.

Global Repair Time policy

In this section, we provide the specification of the repair policy called *Global Repair Time* (GRT) [32].

According to the GRT policy, the repair of a subsystem is triggered by the occurrence of the failure of the subsystem. The detection of the trigger condition is immediate.

The repair action may involve all the failed components of the subsystem or a subset of them. The number of repair facilities is infinite, so there is no limit to the number of components under repair at the same time.

A global repair rate is defined in the GRT policy, so the recovery of all the components involved in the repair process, is considered as an atomic action which is enabled by the trigger condition, and is completed after a random period of time ruled by the negative exponential distribution having the global repair rate as parameter.

During the execution of the repair action, the GRT policy assumes that no failure events can occur.

The RFT formalism defined in section 5.3, allows to associate a GRT policy to a RB: given $b \in \mathcal{RB} : \beta(b) = GRT$,

- $\bullet g$ is the trigger condition of b, i. e. the event modelling the failure of the subsystem to be repaired, and enabling the repair action of b;
- $g\bullet = Cov_{BE}(b)$ is set of BEs modelling the failure of the components involved in the repair action modelled by b;
- the global repair rate of b is given by $\mu(b)$.

Single event Repair Time policy

In [32], another repair policy called *Single event Repair Time* (SRT) policy is defined. In this policy, the trigger condition is still the failure of the subsystem, but the time to detect the trigger condition and to enable the repair action, is a random variable. Moreover, a repair rate is defined for each repairable component, so that the end of the repair of each component, is not contemporary.

Two versions of the SRT policy are presented in [32]: with infinite repair facilities (SRT-I) and with finite repair facilities (SRT-F).

In [32] the GRT, SRT-I and SRT-F policy are evaluated and compared by means of the RFT formalism; in this chapter, we limit our attention on the GRT policy.

5.3.2 Running example

Repair mode of the system

Fig. 5.1 shows the RFT model for the Multiproc system described in section 2.2.1, with the addition of two repair processes. One process involves the shared memory $R1$ together with its memory bus $B1$; the repair process is activated when both the shared memories $R1$ and $R2$ can not be accessed by the processing units, due to the failure of the shared memories or to the failure of the memory buses. The other repair process involves both hard disks ($D1$, $D2$) together with the disk bus $DBUS$. This repair process is activated when both hard disks can not be accessed by the processing units, due to the failure of the disks or to the failure of the disk bus.

RFT model of the system

The RFT model of the system including the repair processes and shown in Fig. 5.1, has been obtained from the FT model in Fig. 2.3 with the addition of two RBs named $REP1$ and $REP2$. Tab. 2.2 indicates the correspondence between the events in the RFT and the system components and the subsystems. The failure rates of the components are the same as in Tab. 2.1. The repair policy associated to $REP1$ and $REP2$ is the GRT policy (section 5.3.1); the repair rate of $REP1$ and $REP2$ is $0.01h^{-1}$ ($\mu(REP1) = 0.01h^{-1}$, $\mu(REP2) = 0.01h^{-1}$).

$REP1$ models the repair process involving $B1$ and $R1$; its trigger event is the IE SM representing the denied access to both shared memories. The basic coverage set of $REP1$ is composed by the BEs $R1$ and $B1$ ($Cov_{BE}(REP1) = \{R1, B1\}$). So, when the IE $B1$ occurs ($B1 = true$), the RB $REP1$ is activated with the effect of repairing the BEs $R1$ and $B1$ ($R1 = false$, $B1 = false$) after a random period of time ruled by a negative exponential distribution with parameter $\mu(REP1)$. As a consequence of the repair of $B1$ and $R1$, the value of other events may become $true$. The events whose value is influenced by the RB $REP1$ is contained in the coverage set of $REP1$: $Cov_E(REP1) = \{R1, B1, BR1, SM, MEM1, MEM2, MEM3, PU1, PU2, PU3, CM, TE\}$.

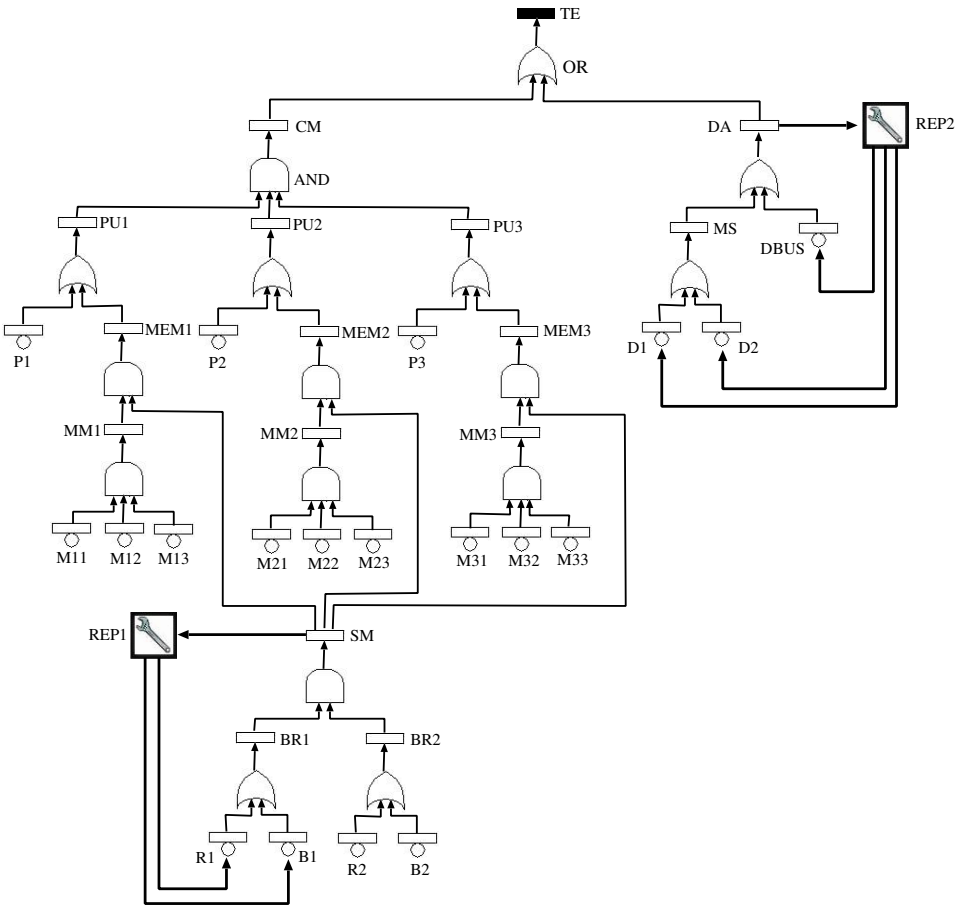

Figure 5.1: RFT model of the Multiproc system with repair processes.

The RB $REP2$ models the repair process involving directly $D1$, $D2$ and $DBUS$. The trigger event of $REP2$ is the IE DA, while $Cov_{BE}(REP2) = \{D1, D2, DBUS\}$. The repair process represented by $REP2$ is activated when the event DA occurs ($DA = true$), and after a random period of time the BEs $D1$, $D2$ and $DBUS$ are repaired ($D1 = false$, $D2 = false$, $DBUS = false$). The coverage set of $REP2$ is $Cov_E(REP2) = \{D1, D2, MS, DBUS, DA, TE\}$.

5.4 Converting a RFT model into a GSPN

This section describes the model transformation system (section 4.5) to convert a RFT model (source model) to a GSPN (target model), assuming that a GRT repair policy (section 5.3.1) is associated to each RB in the RFT.

Some of the compound rules dealing with the RB, need the specification of a new attribute for the RBs, called $RepEv$; before the begin of the model transfor-

mation process, for each RB b in the RFT, we set the attribute $RepEv(b)$ to the set of labels of the events belonging to the set $Cov_E(b) - Cov_{BE}(b)$. In other words, the attribute $RepEv$ of the RB b contains the set of the labels of the non basic events belonging to the coverage set of b.

The compound rules in the model transformation system from RFT to GSPN, are in the form described in section 4.5.3. These rules need the introduction in the RFT formalism, of some new functions:

- $conv : \mathcal{E} \rightarrow \mathbb{B}$ is the function returning the $true$ value if an event in the source RFT model, has already been mapped in the GSPN target model, and returning the $false$ value if an event in the source model has not yet been mapped in the target model.
- $lab : \mathcal{E} \rightarrow \{A, \ldots, Z\}^+$ is the function returning the label assigned to an event, where $\{A, \ldots, Z\}^+$ is the set of all the possible non empty strings we can compose with the alphabet $\{A, \ldots, Z\}$.

The lab function has been defined also in the GSPN formalism:

- $lab : P \cup T \rightarrow \{A, \ldots, Z\}^+$ is the function returning the label assigned to a place or transition.

In the compound rules in our model transformation system, labels are used to identify the nodes inside the source model and the target model:
$\forall e, e' \in \mathcal{E} : e \neq e', lab(e) \neq lab(e')$
$\forall p, p' \in P : p \neq p', lab(p) \neq lab(p')$
$\forall t, t' \in T : t \neq t', lab(t) \neq lab(t')$
We can classify RFTs and GSPNs as labelled attributed oriented graphs (section 4.5.1); labels are returned by the function lab, while attributes are returned by the other functions defined in the RFT formalism (section 5.3) and in the GSPN formalism (section 4.4.2).

In order to map in the GSPN the failure mode of the system, we can use the compound rules to convert the events and the Boolean gates, defined in the model transformation system from DFT to GSPN (section 4.6). Such rules are depicted in Fig. 4.5, Fig. 4.6, Fig. 4.7, Fig. 4.9, Fig. 4.10, Fig. 4.12. In order to map in the GSPN the repair mode of the system, we use the compound rules defined in this section.

5.4.1 RB conversion

The compound rule in Fig. 5.2 can be applied to the a RB and to its trigger event. r_s can be applied to a subgraph of the target model, composed by a RB B connected to the trigger event E by means of the arc (E, B); moreover, E must already be mapped in the GSPN ($conv(E) = true$) by means of the compound rule in Fig. 4.5, in Fig. 4.6 or in Fig. 4.7. In r_s, E is an IE, but it might be originally a BE or the TE; this is due to the previous application of the rule in Fig. 4.7 to a BE,

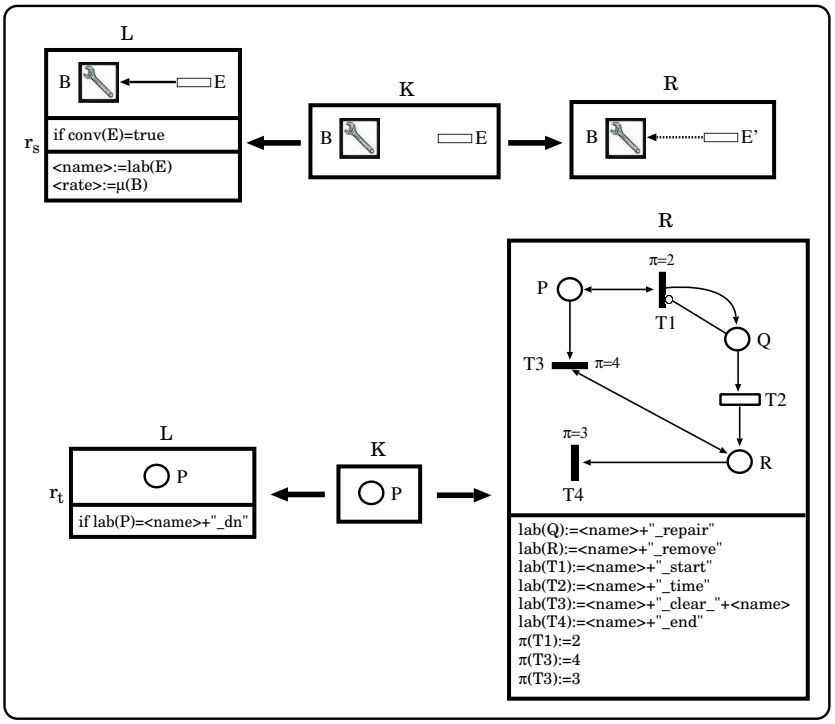

Figure 5.2: Compound for the trigger event of a RB.

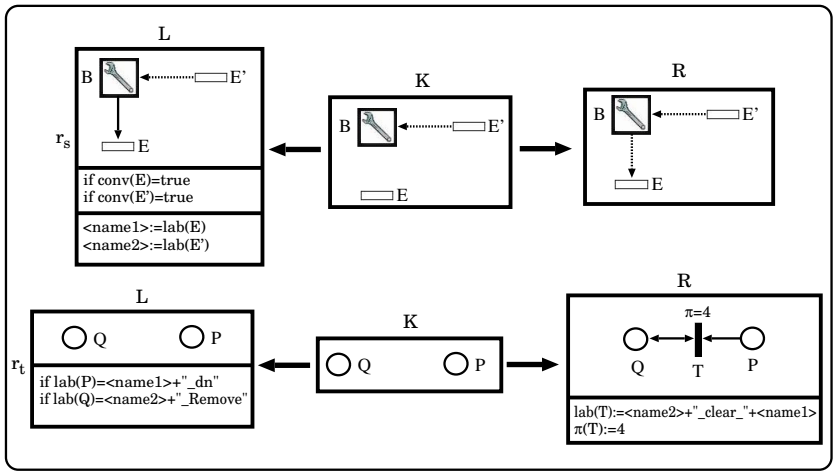

Figure 5.3: Compound for an event in the basic coverage set of a RB.

or of the rule in Fig. 4.6 to the TE. The effect of r_s on the source model is the removal of the arc (E, B) and its replacement with an arc drawn as a dashed line, still from E to B; in this way, the rule in Fig. 5.2 can not be applied again to the same RB and on the same trigger event. r_t creates in the GSPN target model, the subnet modelling the trigger condition of the RB, the begin of the repair action, the time to repair, and the end of the repair action. All these aspects respect the GRT repair policy (section 5.3.1) associated with the RB.

The compound rule in Fig. 5.3 concerns a RB and one of the events in its basic coverage set. r_s can be applied to a subgraph of the RFT, composed by the repair box B, the trigger event E' and the event E belonging to the basic coverage set of B. The trigger event E' is identified by the arc (E', B); since in r_s this arc is drawn as a dashed line, the compound rule in Fig. 5.3 can be applied only if the compound rule in Fig. 5.2 has already been applied to B and E'. The event E appears in r_s of the compound rule in Fig. 5.3 as an IE; originally, it was a BE; this is due to the previous application to E of the compound rule in Fig. 4.5, in Fig. 4.6 or in Fig. 4.7. The effect of the application of r_s of the compound rule in Fig. 5.3 to the RFT source model, is the removal of the arc (B, E) and its replacement with an arc drawn as a dashed line, still from B to E; in this way, we avoid the further application of the compound rule in Fig. 5.3 to the same RB and the same event in the basic coverage set. The effect of the application of r_t to the target model, is the creation of an immediate transition modelling the repair of the failure event in the basic coverage set of the RB.

The compound rule in Fig. 5.4 concerns a RB and one of the non basic events in its coverage set. r_s can be applied to a subgraph of the RFT composed by the RB B, the trigger event E' and the non basic event E in the coverage set of B. The trigger event E' is identified by the arc (E', B); since in r_s this arc is drawn as a

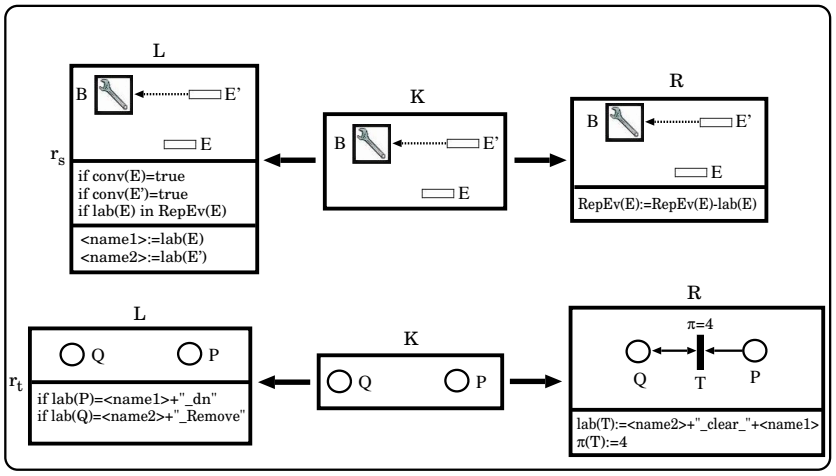

Figure 5.4: Compound for an IE in the coverage set of a RB.

dashed line, the compound rule in Fig. 5.4 can be applied only if the compound rule in Fig. 5.2 has already been applied to B and E'. The event E appears in r_s of the compound rule in Fig. 5.3 as an IE; originally, it might be the TE; in this case, it was converted to an IE by the application of the compound rule in Fig. 4.6. However, the event E must already be mapped in the GSPN ($conv(E) = true$). The non basic event E is classified as an element of the coverage set of B, if its label is contained inside the attribute $RepEv$ of B ($if\ lab(E)\ in\ RepEv(E)$). The application of r_s of the compound rule in Fig. 5.4 determines in the source RFT model, the removal of the label of E from the set contained inside $RepEv(E)$; in this way, the rule in Fig. 5.4 can be applied only once to the same RB and to the same non basic event belonging to the coverage set of the RB. The application of r_t to the target model, creates an immediate transition modelling the repair of the non basic event in the coverage set of the RB.

Any of the compound rules introduced in this section need the previous mapping of the RFT events by means of the rules in Fig. 4.5, in Fig. 4.6, in Fig. 4.7. Any of the compound rules in this model transformation system can be applied only once to the same elements of the RFT; this property combined with the fact that a RFT model is composed by a finite number of nodes and arcs, guarantees the termination of the model transformation process. The order of application of the compound rules does not influence the final result of the transformation.

5.4.2 Running example

With the purpose of clarifying the conversion of a RFT model in GSPN, we take into account the subtree $\widehat{SM}$ of the RFT model in Fig. 5.1. The conversion into GSPN of this subtree, is shown in Fig. 5.6. In $\widehat{SM}$ the RB $REP1$ is present, with $Cov_{BE}(REP1)$=$\{R1, B1\}$; the trigger event of $REP1$ is SM.

In the GSPN, the places SM_dn, $BR1\_dn$, $BR2\_dn$, $R1\_dn$, $B1\_dn$, $R2\_dn$, $B2\_dn$ and the transitions SM_and, $BR1\_or\_R1$, $BR1\_or\_B1$, $BR2\_or\_R2$, $BR2\_or\_B2$, $R1\_fail$, $B1\_fail$, $R2\_fail$, $B2\_fail$ are the conversion of the events and of the gates in $\widehat{SM}$ modelling the failure mode of the subsystem. Their conversion is realized by means of the compound rules for the events and the Boolean gates described in section 4.6. The explanation of the conversion of the RB $REP1$ in the GSPN in Fig. 5.6, follows.

The trigger condition of the RB is modelled in the GSPN by the immediate transition SM_start which puts one token in the place SM_repair as soon as the place SM_dn becomes marked (SM_dn corresponds to SM). The marking of SM_repair indicates the begin of the time to repair the subsystem modelled by $\widehat{SM}$ in the RFT. The random time to repair the subsystem is modelled in the GSPN by the timed transition SM_time whose firing rate is equal to the repair rate of the RB $REP1$. The firing of this transition determines the end of the time to repair the subsystem; the effect of the firing is moving the token from SM_repair to SM_remove; the marking of SM_remove indicates the end of the time to repair the subsystem.

At this point, the immediate transitions SM_clear_SM, SM_clear_BR1, SM_clear_R1, SM_clear_B1 are enabled to fire with the effect of removing the token inside the places SM_dn, $BR1\_dn$, $R1\_dn$, $B1\_dn$ corresponding to the events of $\widehat{SM}$ included in the coverage set of the RB $REP1$. Finally, the immediate transition SM_end removes the token inside SM_remove to complete the execution of the repair action.

The places SM_repair, SM_remove and the transitions SM_start, SM_time, SM_clear_SM, SM_end are the result of the application of the compound rule in Fig. 5.2 to the RB $REP1$ and its trigger event SM. The transitions SM_clear_R1 and SM_clear_B1 are the result of the application of the compound rule in Fig. 5.3 to $REP1$ and to the events in its basic coverage set. The application of the compound rule in Fig. 5.4 to $REP1$ and the event $BR1$, produced the transitions SM_clear_BR1.

The priorities assigned to the immediate transitions in the GSPN in Fig. 5.6 allow the correct order of the steps of the repair action. Moreover, all the immediate transitions modelling the repair process have a priority $\pi_r \geq 2$, higher than the priority $\pi_f = 1$ of the immediate transitions modelling the failure mode. In this way, we assure that no failure events can occur during the repair process.

5.5 Module based RFT analysis

In a RFT model, the state space analysis is actually required only by the subtrees where the value of the events depends on the action of some RB. The other subtrees can be solved with the standard combinatorial method (BDD based analysis). So, the evaluation of a RFT consists of the analysis in the state space of the subtrees where some RB is present, and of the combinatorial analysis of the rest of the RFT.

Such approach for the RFT analysis requires the solution of some subtrees in isolation; in order to analyze a subtree in isolation, the subtree must be independent from the rest of the RFT (section 2.5). So, the modules (independent subtrees) detection on a RFT becomes necessary to this aim.

In a FT, a module is a subtree which is structurally independent from the rest of the FT; this happens if the events contained in the subtree, do not occur elsewhere in the FT [44]. This notion of module must be extended in the case of RFTs, but we need also a way to classify the modules according to the analysis technique they require (combinatorial or state space analysis).

5.5.1 Modules detection and classification

The modules detection algorithm [44] for FT models proposed in section 2.5.2, has the purpose of verifying the independence of a subtree of the FT, in structural terms. This algorithm is still useful on RFT models, to verify the structural independence of the subtrees. However, this algorithm is applied to FTs interpreting the FT as a tree structure by reversing the orientation of the FT arcs; in order to perform the algorithm on a RFT, we have to ignore the RBs and the arcs touching the RBs, and we have to perform the reversing of the orientation of the arcs connecting events to gates and vice-versa.

In a RFT, we consider the subtree $\widehat{e}$ ($e \in \mathcal{IE} \cup \{TE\}$) as a module if two conditions hold:

- $\widehat{e}$ is structural independent from the rest of the RFT. This condition can be verified using the algorithm to detect FT modules (section 2.5.2).
- $\nexists b \in \mathcal{RB} : e \in {\circ}b - {\bullet}b$.

Once the RFT modules have been detected, we can classify the RFT modules according to the solution method they need:

- the module $\widehat{e}$ is a *Combinatorial Solution Module* (CSM) if

  $$\nexists e' \in \mathcal{E} \wedge \nexists b \in \mathcal{RB} : e' \in {\circ}e \wedge e' \in Cov_E(b)$$

  In other words, the module $\widehat{e}$ is a CSM if it any of its events does not belong to the coverage set of a RB.

- the module $\widehat{e}$ is *State space Solution Module* (SSM) if

  $$\exists e' \in \mathcal{E} \wedge \exists b \in \mathcal{RB} : e' \in {\circ}e \wedge e' \in Cov_E(b)$$

  In other words, the module $\widehat{e}$ is a SSM if it contains at least one event belonging to the coverage set of a RB.

For the CSMs the state space analysis is not necessary; for them, the less expensive combinatorial analysis is enough. The state space analysis is essential for the

SSMs, since they contain dependencies due to the presence of RBs. We indicate with $\mathcal{M}$ the set of the modules in the RFT.

Given a RB b of the RFT and its trigger event $v \in \bullet b$, a *Repairable Module* (RM) is the smallest SSM $\widehat{e}$ in the RFT such that v is contained in $\widehat{e}$. Formally, $\widehat{e} \in \mathcal{M}$ is a RM if

$$\exists b \in \mathcal{RB} \wedge \exists v \in \mathcal{E} : v \in \bullet b \wedge v \in \circ e \wedge \nexists \widehat{e'} \in \mathcal{M} : v \in \circ e' \wedge e' \in \circ e$$

We indicate with $\mathcal{RM}$ the set of the RMs of a RFT. A module $\widehat{e}$ is a *Maximal Repairable Module* (MRM) if $\widehat{e}$ is a RM and is not contained inside another RM; formally, $\widehat{e} \in \mathcal{RM}$ is a MRM if

$$\nexists \widehat{e'} \in \mathcal{RM} : e \in \circ e'$$

The definition of RM and MRM is useful in the RFT analysis by modularization.

5.5.2 RFT modularization

The main reason to perform the RFT analysis by modularization, i. e. exploiting modules, is applying the state space analysis to the subtrees of the RFT requiring it, while the less expensive combinatorial analysis can be performed on the rest of RFT. We limit our attention on the quantitative analysis of a RFT at a certain mission time t.

As in the case of the FTs (section 2.5.3), the main steps of the modularization of a RFT are: modules detection, modules classification, decomposition, modules analysis and aggregation. In the decomposition step, we have to detach from the RFT, modules needing the state space analysis, in other words, we have to detach SSMs. The state space analysis of a SSM can be realized by mapping the RFT module in a GSPN as shown in section 5.4; the SSM must include the RB(s) connected to the events of the SSM. Then, the resulting GSPN can be analyzed returning the probability to be marked at time t, of the GSPN place corresponding to root event of the module.

As in the case of DFT modularization (section 4.7.2), we have to choose the SSMs to be detached in such a way to avoid that any BE replacing a SSM, belongs to another SSM. The reason for that is the impossibility to map in GSPN a BE having a probability to fail instead of a failure rate (compound rule in Fig. 4.7). So, in the Decomposition step, the MRMs of the RFT are the modules chosen to be detached from the RFT. Any trigger event of a RB belongs to a RM and to a MRM. Choosing the MRMs (together with the RBs connected to their events) as the modules to be detached and analyzed in isolation in the state space, all the RBs in the RFT are included in one of the detached modules. After the aggregation step, no BEs replacing modules, will be part of a SSM, since no RBs will be present in the RFT.

Following this strategy, the modularization steps do not need to be iterated, but it is necessary to perform them only once in order to analyze in the state space any

subtree needing this kind of analysis. After the aggregation step, the RFT does not contain any RB, so it is now a FT and can be analyzed with the combinatorial technique (BDD generation). While we can not map a BE having a probability instead of a failure rate in a GSPN, a BDD can deal with such a BE.

If the whole RFT is a MDM, we convert the whole RFT in a GSPN, and we analyze the resulting GSPN computing the probability of the GSPN place TE_dn to be marked at the mission time t.

The steps to perform the modularization of a RFT follow:

1. Modules detection
2. Modules classification: for each module, we verify if it is a MRM.
3. if $\widehat{TE}$ is a MRM go to step 11, else go to step 4.
4. Decomposition: each MRM is detached from the RFT.
5. MRM conversion to GSPN: each MRM is converted to GSPN.
6. GSPN analysis: each MRM in GSPN form, is analyzed.
7. Aggregation: each detached MRM is replaced in the RFT, by a BE (we obtain a FT).
8. FT conversion to BDD
9. BDD quantitative analysis returning the final result.
10. end.
11. RFT conversion to GSPN.
12. GSPN analysis: the RFT in GSPN form, is analyzed returning the final result.
13. end.

5.5.3 Running example

In this section, we perform by modularization, the quantitative analysis at time $t = 10000h$ of the RFT model in Fig. 5.1.

According to the RFT module definition and classification provided in section 5.5.1, the RFT in Fig. 5.1 contains the following CSMs: $\widehat{BR2}$, $\widehat{MM1}$, $\widehat{MM2}$, $\widehat{MM3}$. The SSMs in the RFT in Fig. 5.1 are: $\widehat{SM}$, $\widehat{CM}$, $\widehat{DA}$, $\widehat{TE}$.

Fig. 5.5 shows the MRMs present in the RFT model of the Multiproc system, according to the MRM definition provided in section 5.5.1; they are $\widehat{SM}$ and $\widehat{DA}$. According to the RFT modularization steps listed in Fig. 5.5.2, each of these MRMs together with the RB connected to its events, needs to be detached from the

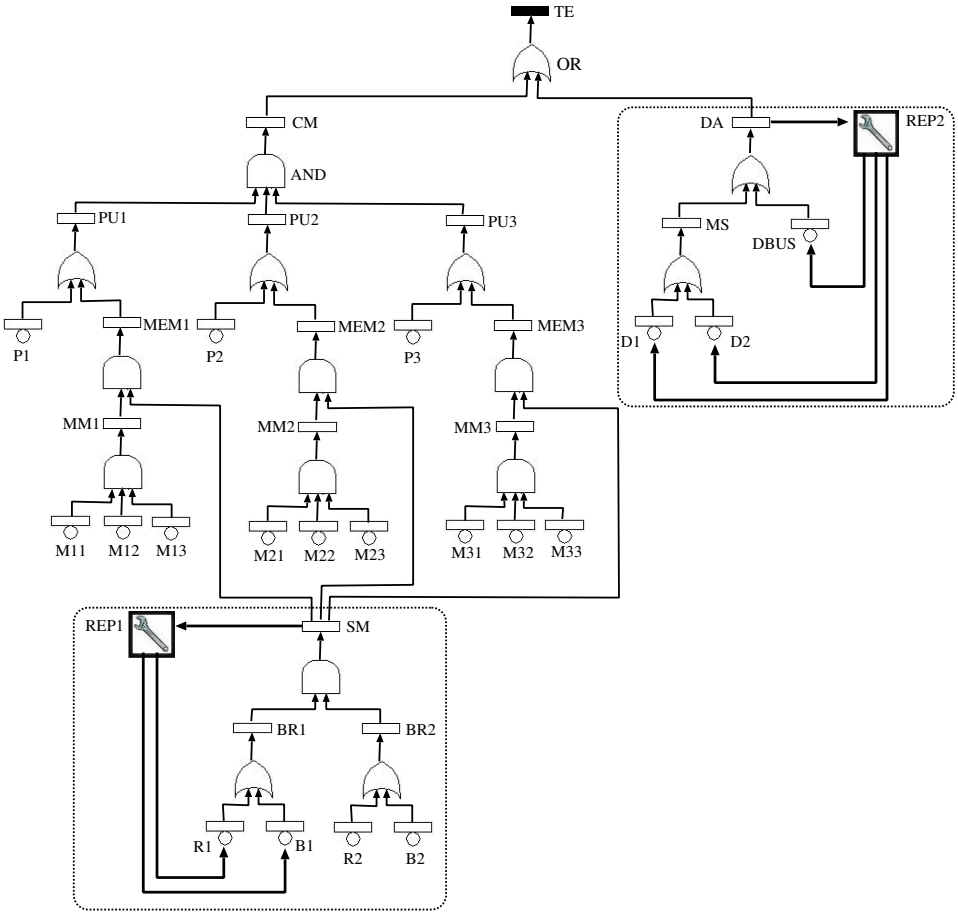

Figure 5.5: The MRMs in the RFT model of the Multiproc system.

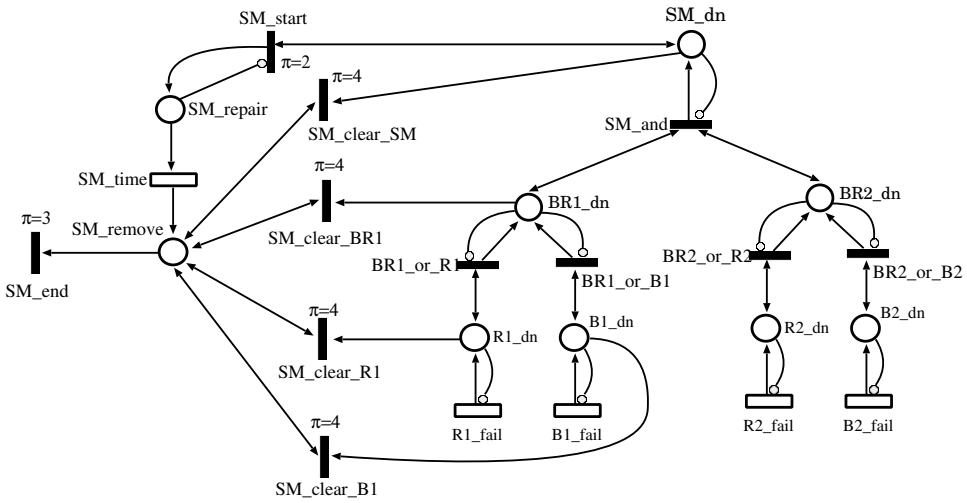

Figure 5.6: The GSPN corresponding to the module $\widehat{SM}$.

RFT (decomposition step), converted in GSPN form (conversion step), analyzed in GSPN form (analysis step) and replaced by a BE whose probability to occur is the probability of the MRM to be failed at the required mission time.

By means of the model transformation system from RFT to GSPN described in section 5.4, the MRM $\widehat{SM}$ has been converted in the GSPN depicted in Fig. 5.6, while the MRM $\widehat{DA}$ has been converted in the GSPN shown in Fig. 5.7. The probability of the MRM $\widehat{SM}$ to be failed at time $t = 10000h$ is computed as the probability of the place SM_dn in the corresponding GSPN, to be marked at time $t = 10000h$:

$$Pr\{SM, t\} = Pr\{m(SM\_dn) = 1, t\} = 2.006E - 9$$

The module $\widehat{SM}$ is replaced in the RFT model by the BE SM having this probability. The probability of the MRM $\widehat{DA}$ to be failed at time $t = 10000h$ is computed as the probability of the place DA_dn in the corresponding GSPN, to be marked at time $t = 10000h$:

$$Pr\{DA, t\} = Pr\{m(DA\_dn) = 1, t\} = 1.601743E - 4$$

The module $\widehat{DA}$ is replaced in the RFT model by the BE DA having this probability.

Replacing the MRMs of the RFT with BEs, we obtain the RFT shown in Fig. 5.8; this RFT contains no RBs, so it is actually a FT: it can be solved by generating and analyzing the corresponding BDD depicted in Fig. 5.9. The probability of the TE (Unavailability of the system) at time $t = 10000h$ is $1.602984E - 4$; Tab. 5.1 shows the results obtained on the RFT for a mission time varying between $1000h$ and $10000h$.

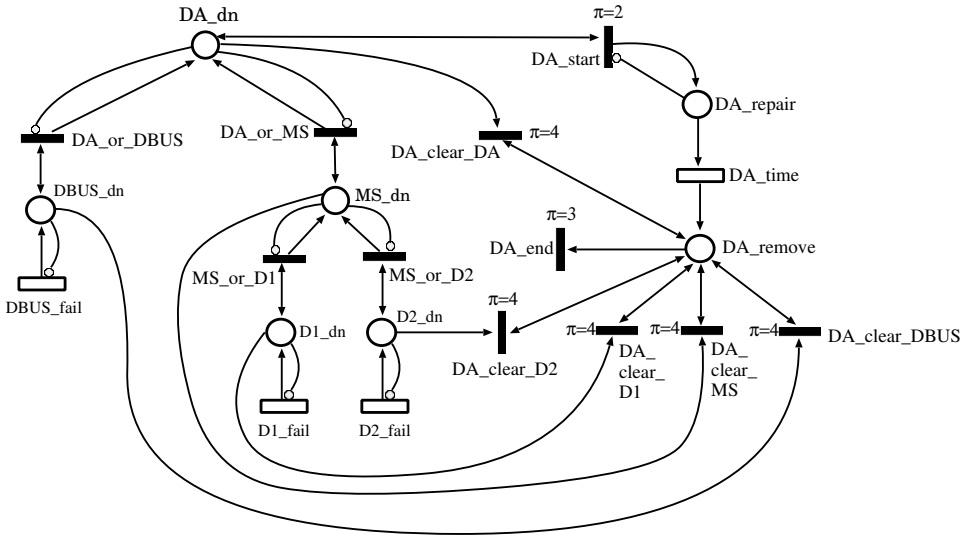

Figure 5.7: The GSPN corresponding to the module $\widehat{DA}$.

time t	$Pr\{SM, t\}$	$Pr\{DA, t\}$	$Pr\{TE, t\}$
1000 h	1.84E-10	1.601671E-4	1.601672E-4
2000 h	3.89E-10	1.601743E-4	1.601753E-4
3000 h	5.73E-10	1.601743E-4	1.601777E-4
4000 h	7.78E-10	1.601743E-4	1.601823E-4
5000 h	9.83E-10	1.601743E-4	1.601899E-4
6000 h	1.188E-9	1.601743E-4	1.602012E-4
7000 h	1.392E-9	1.601743E-4	1.602170E-4
8000 h	1.597E-9	1.601743E-4	1.602379E-4
9000 h	1.802E-9	1.601743E-4	1.602648E-4
10000 h	2.006E-9	1.601743E-4	1.602984E-4

Table 5.1: Unavailability values for the Multiproc system with repair processes.

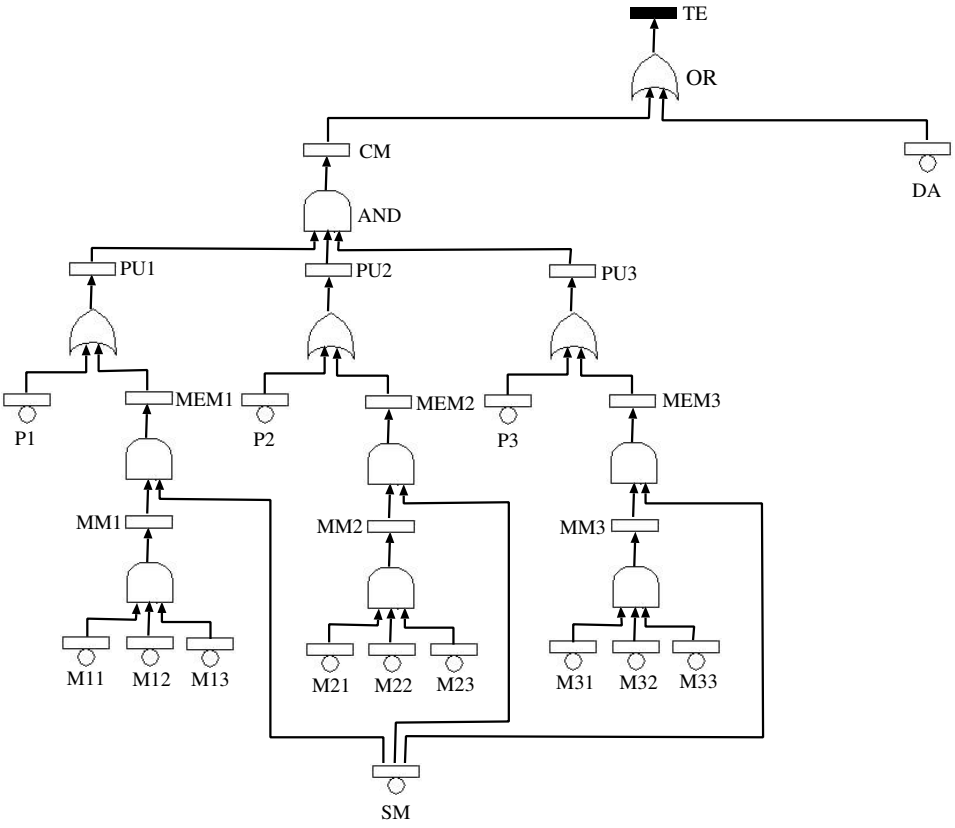

Figure 5.8: The RFT model of the Multiproc system after the analysis of the MRMs.

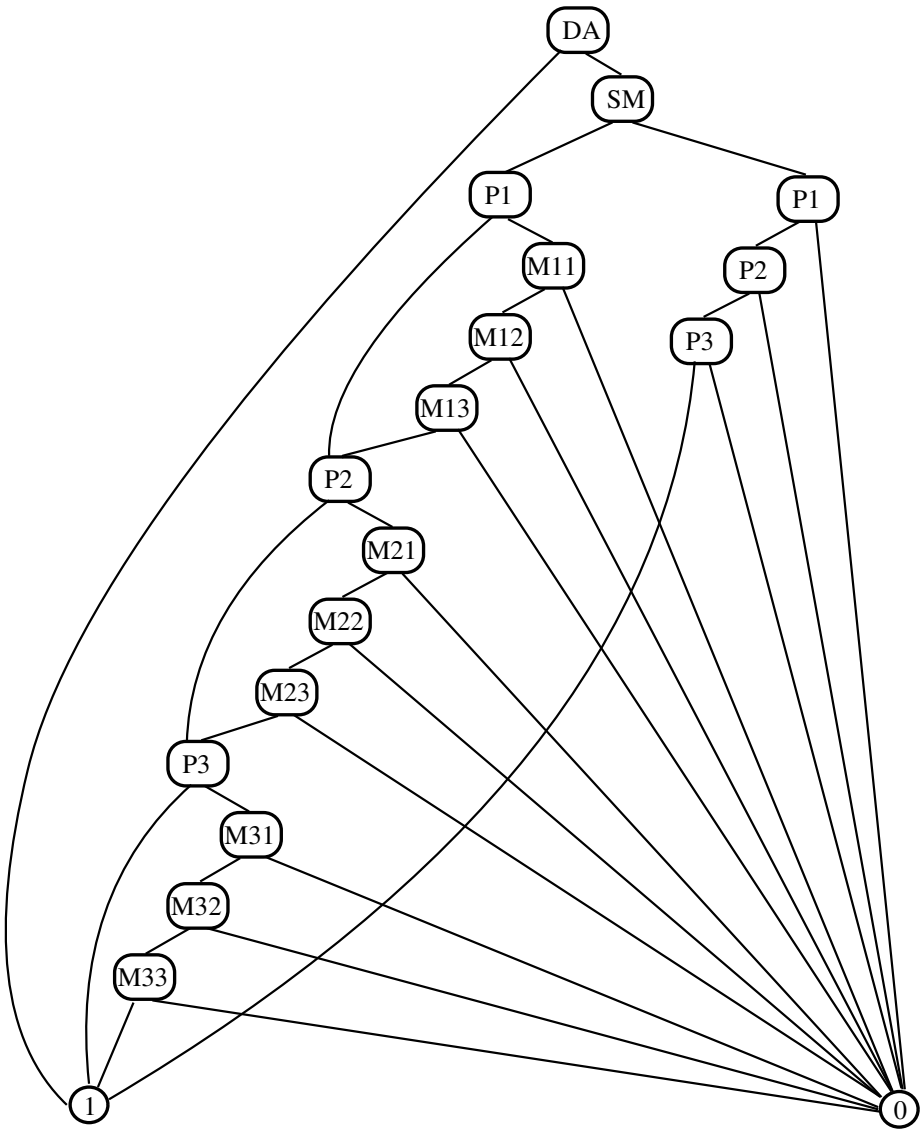

Figure 5.9: The BDD corresponding to the (R)FT in Fig. 5.8.

Chapter 6

Integration of extended FT formalisms

6.1 Integrating the PFT, DFT and RFT formalism

In this chapter, we compose the PFT, DFT and RFT formalism described in chapters 3, 4 and 5 respectively, to generate a unique formalism called *Dynamic Repairable Parametric Fault Tree* [8, 9, 29] (DRPFT). The DRPFT formalism includes and integrates the primitives of the FT formalism (BEs, IEs, TE, Boolean gates), the primitives introduced in the PFT formalism (REs, BREs, parameters, types), the primitives introduced in the DFT formalism (dynamic gates), and the primitive introduced in the RFT formalism (repair box (RB, for short)).

So, the DRPFT formalism allows to build models where replicated components and subsystems can be represented in compact form, dependencies among component failure events can be represented by means of dynamic gates, and the presence of repair processes can be modelled using RBs.

An example of DRPFT model is shown in Fig. 6.6.

Due to the presence of dynamic gates and RBs, a DRPFT model needs the state space analysis. While a DFT model or a RFT model can be converted in a GSPN model (sections 4.6 and 5.4), a DRPFT model can not be mapped in a GSPN because there is no way to maintain the parametric form using the GSPN formalism. Instead of GSPN, a DRPFT model can be mapped in a High-level Stochastic Petri Net, in form of *Stochastic Well-formed coloured Nets* (SWN) [24, 25]. The SWN formalism extends the GSPN one, with the addition of coloured tokens [23], introduced for the first time in *Coloured Petri Nets* (CPN) [65]. The conversion of a DRPFT in a SWN allows to perform the state space analysis and to maintain the representation of the redundancies in parametric form.

As in the case of GSPN, the analysis of a SWN begins with the generation of the reachability graph. SWNs have the property that they generate symbolic markings that may be viewed as a high level description of sets of actual markings. So, from a SWN, a *symbolic* reachability graph [25] can be derived; the definition

of symbolic markings allows to exploit symmetry properties in the model and to generate the underlying CTMC in lumped form. The degree of saving in the state space generation depends on the redundancies present in the system and can be very consistent [11].

The way to convert a DRPFT model in a SWN is described in appendix A.

A further reduction of the computational costs of the state space analysis of a DRPFT, can be achieved by analyzing the DRPFT by modularization, in such a way to apply the state space analysis only to the DRPFT subtrees containing dynamic gates and/or RBs, while the rest of the DRPFT (containing only Boolean gates) can be solved through the combinatorial approach (pBDD).

6.1.1 Considerations on SWN and SAN

A coloured subnet of a SWN model, folds several symmetric GSPNs representing them in a compact way; from this point of view, the SWN formalism reminds the *Stochastic Activity Networks* (SAN) [86]. The main composing elements of the SAN formalism are immediate and timed activities (similar to GSPN transitions), places (containing tokens), and gates (differing from FT gates and used to set which place markings enable activities to fire, together with effect of the activities firing on the place markings). In the SAN formalism, there are no coloured tokens.

The *Replicate/Join* formalism [85] was conceived for SAN models; such formalism allows to express by means of a tree structure, the way to compose together several SAN models in a unique large composed SAN model. In the tree structure, leaf nodes are atomic SAN models, each non leaf nodes is a $Join$ or $Replicate$ operator, and the root node is the model resulting from the composition of atomic models according to the operators in the composed model. In particular, the $Join$ operator compose two or more SAN models by superposition over their common elements; the $Replicate$ operator constructs a model consisting of a number of identical copies of a certain SAN model (copies may share common elements).

The composition and the analysis of SAN models is supported by the *Möbius* tool [27, 34, 35, 37]. As in the case of SWN models, if a composed SAN model presents structural symmetries, its analysis exploits such symmetries by solving a smaller state space than if the symmetry were not present [85].

While in a composed SAN model, the replication (folding) is obtained by means of atomic SAN models and the use of specific operators in an associated tree structure, the SWN formalism allows to specify any aspect concerning the symmetric structure of the system to be modelled, directly in the SWN model by exploiting coloured tokens, with no external composition model. In other words, a composed SAN model involves two formalisms (the SAN formalism and the *Replicate/Join* formalism), while a SWN model is based on a unique formalism.

For this reason, mapping the semantic of a DRPFT model in a SWN is easier to be implemented by a a model transformation system based on graph transformation rules (section 4.5.3), than mapping a DRPFT in a composed SAN model. Moreover, the SWN formalism can be classified as an evolution of the GSPN for-

malism, as the DRPFT formalism can be considered as an extension of the DFT and RFT formalism; since a model transformation system from DFT to GSPN (section 4.6) and from RFT to GSPN (section 5.4) were already studied, the conversion of DRPFTs into SWNs can be obtained by combining and extending such model transformation systems (appendix A).

6.1.2 Dynamic gates in parametric form

The semantic of the dynamic gates described in section 4.2, must be extended in order to cope with the presence of (B)REs among the input, trigger or dependent events of a dynamic gate.

Let us consider first the $FDEP$ gate; the trigger event of a $FDEP$ gate must not be a (B)RE; in the original notation of the this type of gate, given in the DFT formalism (section 4.3), the trigger event must be a single event; a (B)RE folds several (B)Es, so a (B)RE as trigger event of a $FDEP$ gate would not be consistent with the notation of this type of gate. The trigger event e' of a gate of type $FDEP$ can be a BE or an IE. A (B)RE e can be instead the dependent event of a $FDEP$ gate, modelling the functional dependency of several events folded in e, on the trigger event e'.

A gate of type $PAND$ can have a (B)RE as input event; in this case, we assume that it is not allowed for the gate to have other input events. Suppose that the (B)RE e is the input of a gate g of type $PAND$; p is the parameter declared in e, and the type of p is $C_p = \{1, \ldots, m\}$ $(m \geq 2)$ If e' is the output event of g, then the value of e' is $true$ if two conditions holds:

- $e(p = 1) = true, e(p = 2) = true, \ldots, e(p = m) = true$
  This means that the value of each event folded in e according to the parameter p, is $true$.

- $e(p = 1) \prec e(p = 2) \prec \ldots \prec e(p = m)$
  This means that the events folded in e according to the parameter p occurred in a certain order given by the increasing order of the elements of the type of p $(\tau(p) = C_p)$.

The output event of a gate of type $PAND$ can be an IE or a RE.

A gate g of type SEQ can be connected to a BRE e by means of the arc (g, e). In this case, we assume that the g can not be connected to other events, while e folds several BEs, including the "trigger" event and the dependent events of g. If p is the parameter declared in e, and the type of p is $C_p = \{1, \ldots, m\}$, then $e(p = 1)$ is the "trigger event" of g, and $e(p = 2), \ldots, e(p = m)$ are the dependent events of g. In other words, the BEs folded in e are forced to fail respecting the increasing order of the elements of the type of p:
$e(p = 1) \prec e(p = 2) \prec \ldots \prec e(p = m)$.

In the notation of a gate of type WSP, provided in the DFT formalism, a gate of type WSP must have as input events one BE relative to a main component, and

one or several BEs relative to the spare components. In the DRPFT formalism, a WSP gate must still have one BE relative to a main component as input event. The input events relative to spare components, can be BEs or BREs. In the case of a gate g of type WSP with the BE m and the BRE s as input events, m models the failure of the main component, while s folds several BEs modelling the failure of spare components with the same failure rate and the same dormancy factor. In this case, we assume that g can not have other B(R)Es as input events. The output event of a gate of type WSP can be an IE or a RE. Moreover, we assume that the B(R)Es concerning spare components, can not be connected also to gates of type SEQ.

The conditions ruling the use of parameters in a PFT model (section 3.2) still holds in a DRPFT.

6.1.3 Repair Box semantic in a DRPFT

The RB semantic in a DRPFT model must take into account the dependency existing among events connected to dynamic gates, together with the representation of symmetric subsystems in compact form by means of parameterization.

As in the RFT formalism (section 5.3), in the DRPFT formalism a RB must be connected to its trigger event and to the elements of its basic coverage set by means of arcs, while the coverage set of a RB is the set of events whose Boolean value is influenced by the repair action of the RB on the elements of its basic coverage set. The basic coverage set of the RB must be composed by events belonging to the subtree whose "root" is the trigger event of the RB. In a DRPFT, the trigger event of a RB can belong to these categories: BEs, BREs, IEs, REs, TE; the basic coverage set can be composed by BEs and BREs.

Given a RB b such that the parameter set of its trigger event e, is not empty, b models the presence of several repair processes, each acting on one of the symmetric subsystems represented in compact form by the subtree $\widehat{e}$. If a BRE x belongs to the coverage set of a RB b, then b repairs all the BEs folded in x.

In this chapter, we limit our attention to the GRT repair policy (section 5.3.1) associated to a RB.

6.1.4 Dynamic gates semantic in case of repair

The failure event of a repairable component, is repeatable; so, the presence of RBs in the model, influences the semantic of the dynamic gates requiring or forcing a set of events to occur in a certain order. Let us suppose the presence in the DRPFT of a gate of type $PAND$ whose output event is Y, whose input events are $X1$ and $X2$, and the failure order required to determine the occurrence of Y, is: $X1 \prec X2$. If $X2$ is repairable (i. e. $X2$ belongs to the coverage set of some RB), the event $X2$ is repeatable, i. e. the value of $X2$ can change repeatedly from $false$ to $true$, and from $true$ to $false$. Let us suppose that $X2$ fails and is repaired; then, $X1$ fails. In this case, the failure order has been respected or not? Actually, when $X1$ fails,

$X2$ is not failed; this can be interpreted as the respect of the failure order; however, when $X1$ fails, $X2$ had previously been in the failed state; from this point of view, the failure order has not been respected by the input events of the $PAND$ gate. In other words, when we deal with a gate of type $PAND$ having some repairable input event X, we have to decide if the first or the last failure of X must be taken into account in the verification of the failure order of the input events. In our interpretation, we consider the first failure of the repairable component.

A similar problem arises when a gate of type SEQ forces a set of events to occur in a certain order, and one (or more) of these events is repairable. Let us suppose that by means of a gate g of type SEQ, the events $X1$ and $X2$ are forced to fail in this order: $X1 \prec X2$; moreover, $X1$ is repairable. Let us suppose that $X1$ fails and is repaired; if $X2$ has not yet failed, is it possible the failure of $X2$ now? Two answers can be given to this question: we can say that $X2$ can not fail now because $X1$ is not currently failed, or we can say that $X2$ can fail because $X1$ failed in the past. In other words, when we deal with a gate of type SEQ acting on some repairable component X, we have to decide if the successor of X in the failure order, can fail if X is currently failed, or if X failed at least once in the past. We assume that the first interpretation holds.

Also the semantic of a gate of type WSP is influenced by the presence of some RB. We suppose that the gate g is of type WSP, M is its input event relative to the main component, and $S1$, $S2$, $S3$ are its input events relative to the spare components available to replace the main one in case of failure. Moreover, M is repairable. Let us suppose that M fails and M is replaced by $S1$; then, $S1$ fails and M is replaced by $S2$. If now M is repaired, $S2$ must turn to the dormant state. In general, if the main component is repairable, the effect of its repair is twofold: the main component turns from the failed to the working state, and the spare component replacing the main one when the repair ends, turns from the working state to the dormant state (stand-by). If later, the main component fails again, such spare component is still available to replace the main one.

6.2 The DRPFT formalism

The DRPFT formalism is the union of the PFT (section 3.2), DFT (section 4.3) and RFT (section 5.3) formalism. The DRPFT formalism can be defined by the tuple
$\mathcal{DRPFT} = (\mathcal{E}, \mathcal{G}, \mathcal{A}, \mathcal{P}, \mathcal{T}, \mathcal{RB}, \mathcal{GT}, \sigma, \gamma, \tau, \theta, \delta, \omega, \lambda, \alpha, \mu, \mathcal{RP}, \beta, \phi)$
where:

- $\mathcal{E} = \mathcal{BE} \cup \mathcal{IE} \cup \{TE\} \cup \mathcal{RE} \cup \mathcal{BRE}$ is the set of the events in the DRPFT; it is the union of the following sets:
  - $\mathcal{BE}$ is the set of the BEs;
  - $\mathcal{IE}$ is the set of the IEs;
  - $\{TE\}$ is the set composed by the unique TE;

  - $\mathcal{RE}$ is the set of the REs;
  - $\mathcal{BRE}$ is the set of the BREs.

- $\mathcal{G}$ is the set of the gates.
- $\mathcal{RB}$ is the set of the RBs.
- $\mathcal{A} \subseteq (\mathcal{E} \times \mathcal{G}) \bigcup (\mathcal{G} \times \mathcal{E}) \bigcup (\mathcal{E} \times \mathcal{RB}) \bigcup (\mathcal{RB} \times \mathcal{BE})$ is the set of the arcs.
- $\sigma : \mathcal{A} \to \mathbb{N}$ is the function returning the order number of an arc.
- $\mathcal{GT} = \mathcal{BG} \cup \mathcal{DG}$ is the set of types of gate. It is the union of two sets:
  - $\mathcal{BG} = \{AND, OR\}$ is the set of Boolean gate types.
  - $\mathcal{DG} = \{PAND, FDEP, SEQ, WSP\}$ is the set of Dynamic gate types (described in section 4.2).
- $\gamma : \mathcal{G} \to \mathcal{GT}$ is the function assigning to each gate its type.
- $\mathcal{P}$ is the set of parameters.
- $\mathcal{T}$ is the set of types.
- $\tau : \mathcal{P} \to \mathcal{T}$ is the function assigning to each parameter the corresponding type.
- $\theta : \{\mathcal{E} - \{TE\}\} \to \bigotimes\{\mathcal{P} \cup \epsilon\}$ is the function returning the set of parameters associated with an event.
- $\delta : \{\mathcal{RE} \cup \mathcal{BRE}\} \to \mathcal{P}$ is the function returning the parameter declared in a RE or in a BRE. The following properties must hold:
  - $\forall e_1, e_2 \in \{\mathcal{RE} \cup \mathcal{BRE}\}, \delta(e_1) \cap \delta(e_2) = \emptyset$
  - $\forall e \in \mathcal{RE} \forall p \in \delta(e), \exists e' : p \in \theta(e) \wedge e' \in \circ e$
  - Given $e \in \mathcal{RE}$, $\delta(e) = \{p\}$, $e' \in \circ e$, $p \in \theta(e')$, we have that $\forall e'' \in [e' \to e], p \in \theta(e'')$

  Given the event $e \in \mathcal{E}$ such that $\theta(e) \neq \emptyset$ and $\theta(e) = \{p_1, \ldots p_m\}$, with $m \geq 1$, we indicate such event in the DRPFT as $e(p_1, \ldots, p_m)$. If $e \in \{\mathcal{RE} \cup \mathcal{BRE}\}$, then $\delta(e) = \{p_m\}$.
- $\omega : \{\mathcal{E} - \{TE\}\} \to \mathbb{N}$ is the function returning the multiplicity of an event:
  - $\forall e \in \{\mathcal{E} \cup \mathcal{IE} \cup \{TE\}\}, \omega(e) = 1$
  - $\forall e \in \{\mathcal{RE} \cup \mathcal{BRE}\}, \omega(e) = |\tau(\delta(e))|$
- Given $g \in \mathcal{G} : \gamma(g) \in \mathcal{BG} \vee \gamma(g) = PAND \vee \gamma(g) = WSP$,
  $\bullet g = \{e \in \mathcal{E} : \exists (e, g) \in \mathcal{A}\}$ is the set of input events of g;
  $g\bullet = \{e \in \mathcal{E} : \exists (g, e) \in \mathcal{A}\}$ is the output event of g.

- Given $g \in \mathcal{G} : \gamma(g) = FDEP \vee \gamma(g) = SEQ$,
  $\bullet g = \{e \in \mathcal{E} : \exists (e, g) \in \mathcal{A}\}$ is the trigger event of g;
  $g\bullet = \{e \in \mathcal{E} : \exists (g, e) \in \mathcal{A}\}$ is the set of dependent events of g.

- Given $g \in \mathcal{G} : \gamma(g) = WSP$,
  $\bullet_M g = \{e \in \mathcal{E} : \exists (e, g) \in \mathcal{A} : \sigma(e, g) = 0\}$ is the input event of g relative to the main component ($| \bullet_M g| = 1$);
  $\bullet_S g = \{e \in \mathcal{E} : \exists (e, g) \in \mathcal{A} : \sigma(e, g) > 0\}$ is the set of input events of g relative to the spare components ($| \bullet_S g| \geq 1$).
  $\bullet g = \bullet_M g \cup \bullet_S g$.

- given $b \in \mathcal{RB}$,
  $\bullet b = \{e \in \mathcal{E} : \exists (e, b) \in \mathcal{A}\}$ is the trigger event of b;
  $b\bullet = \{e \in \mathcal{BE} \cup \mathcal{BRE} : \exists (b, e) \in \mathcal{A}\} = Cov_{BE}(b)$ is the basic coverage set of b.

- Given $e \in \mathcal{E}$, $\bullet e = \{g \in \mathcal{G} : \exists (g, e) \in \mathcal{A}\} = \bullet_O e \cup \bullet_D e$ is the set of gates having e as output event or dependent event; it is the union of

  - $\bullet_O e = \{g \in \mathcal{G} : (\gamma(g) \in \mathcal{BG} \vee \gamma(g) = PAND \vee \gamma(g) = WSP) \wedge \exists (g, e) \in \mathcal{A}\}$ is the set of gates having e as output event.
  - $\bullet_D e = \{g \in \mathcal{G} : (\gamma(g) = FDEP \vee \gamma(g) = SEQ) \wedge \exists (g, e) \in \mathcal{A}\}$ is the set of gates having e as dependent event.

  $e\bullet = \{g \in \mathcal{G} : \exists (e, g) \in \mathcal{A}\} = e \bullet_I \cup e\bullet_T$ is the set of gates having e as input event or trigger event; it is the union of

  - $e\bullet_I = \{g \in \mathcal{G} : (\gamma(g) \in \mathcal{BG} \vee \gamma(g) = PAND \vee \gamma(g) = WSP) \wedge \exists (e, g) \in \mathcal{A}\}$ is the set of gates having e as input event.
  - $e\bullet_T = \{g \in \mathcal{G} : (\gamma(g) = FDEP \vee \gamma(g) = SEQ) \wedge \exists (e, g) \in \mathcal{A}\}$ is the set of gates having e as trigger event.

- The following conditions about the connection of events with gates, must hold:

  - $\forall g \in \mathcal{G} : \gamma(g) \in \mathcal{BG} \cup \{PAND, WSP\}, \sum_{\forall e \in \bullet g} \omega(e) > 1$
  - $\forall g \in \mathcal{G} : \gamma(g) \in \mathcal{BG} \cup \{PAND, WSP\}, |g \bullet| = 1$
  - $\forall g \in \mathcal{G} : \gamma(g) = WSP, \bullet_M g \subset \mathcal{BE}$
  - $\forall g \in \mathcal{G} : \gamma(g) = WSP, \bullet_S g \subset \mathcal{BE} \cup \mathcal{BRE}$
  - $\forall g \in \mathcal{G} : \gamma(g) = FDEP, \bullet g \subset \mathcal{IE} \cup \mathcal{BE}$
  - $\forall g \in \mathcal{G} : \gamma(g) = FDEP, | \bullet g| = 1$
  - $\forall g \in \mathcal{G} : \gamma(g) = FDEP, |g \bullet| \geq 1$
  - $\forall g \in \mathcal{G} : \gamma(g) = SEQ, | \bullet g| = 1$
  - $\forall g \in \mathcal{G} : \gamma(g) = SEQ, |g \bullet| \geq 0$

  - $\forall e \in \mathcal{BE} \cup \mathcal{BRE}, |\bullet_O e| = 0$
  - $\forall e \in \mathcal{BE} \cup \mathcal{BRE}, |\bullet_D e| \geq 0$
  - $\forall e \in \mathcal{BE} \cup \mathcal{BRE}, |e \bullet_I| + |e \bullet_T| \geq 1$
  - $\forall e \in \mathcal{IE} \cup \mathcal{RE}, |\bullet_O e| = 1$
  - $\forall e \in \mathcal{IE} \cup \mathcal{RE}, |\bullet_D e| \geq 0$
  - $\forall e \in \mathcal{IE} \cup \mathcal{RE}, |e \bullet_I| + |e \bullet_T| \geq 1$
  - $|\bullet_O TE| = 1$
  - $|\bullet_D TE| \geq 0$
  - $|TE \bullet| = 0$

- The following conditions about the connection of events with RBs, must hold:

  - $\forall b \in \mathcal{RB}, |\bullet b| = 1$
  - $\forall b \in \mathcal{RB}, |b \bullet| \geq 1$
  - $\forall b, b' \in \mathcal{RB} : b \neq b', \bullet b \cap \bullet b' = \emptyset$

- Given $b \in \mathcal{RB}$,

  - $\circ b = \{e \in \mathcal{E} : \exists [e' \rightarrow e''] \wedge e' \in b\bullet \wedge e'' \in \bullet b \wedge e \in [e' \rightarrow e'']\}$
  - $b\circ = \{e \in \mathcal{E} : \exists [e' \rightarrow TE] \wedge e' \in \bullet b \wedge e \in [e' \rightarrow TE]\} - \bullet b$
    $Cov_E(b) = \circ b \cup b\circ$ is the coverage set of b.

- Given $b \in \mathcal{RB}$ and $e' \in \bullet b$, the following conditions hold:

  - $Cov_{BE}(b) \subseteq \circ e'$
  - $\circ b \subseteq \circ e'$
  - $Cov_{BE}(b) \subseteq Cov_E(b)$

- Given $g \in \mathcal{G} : \gamma(g) = SEQ$, we assume that

  - $\forall e \in \bullet g, e \in \mathcal{BE} \cup \mathcal{BRE}$
  - $\forall e \in g\bullet, e \in \mathcal{BE} \cup \mathcal{BRE}$
  - $\forall e \in g\bullet, \nexists g \in \mathcal{G} : \gamma(g) = WSP \wedge e \in \bullet_s g$

- $\lambda : \{\mathcal{BE} \cup \mathcal{BRE}\} \rightarrow \mathbb{R}^+$ is the function assigning a failure rate to a BE or to a BRE, if we assume its negative exponential distribution.

- $\alpha : \mathcal{BE} \cup \mathcal{BRE} \rightarrow (0, 1)$ is the function assigning to a B(R)E connected to a gate of type WSP, a dormancy factor.

- $\mu : \mathcal{RB} \rightarrow \mathbb{R}^+$ is the function returning the repair rate of a RB.

- $\mathcal{RP} = \{GRT, SRT - I, SRF - F, \ldots\}$ is the set of repair policies. However, in the DRPFT formalism, we limit our attention to the GRT policy.
- $\beta : \mathcal{RB} \rightarrow \mathcal{RP}$ is the function assigning a repair policy to a RB.
- $\phi : \mathcal{E} \rightarrow \mathbb{B} = \{true, false\}$ is the function returning the Boolean value of an event (Boolean variable).

6.3 Introduction to SWN

The SWN formalism extends the GSPN one, mainly through the introduction of *colour classes*. A colour class is a finite non empty set of colours; any two colour classes are disjoint sets. Colours are used in SWNs to identify the tokens inside a place, by assigning a colour to each token. Typically, a colour class is used to identify object of the same nature. A colour class may be ordered; in this case, given a colour in the colour class, we can determine its predecessor and its successor in the ordered colour class.

Each place of a SWN has a *colour domain* which is the Cartesian product of the set of colour classes associated with the place. The colour of a token inside a place, must belong to the colour domain of the place. Any two tokens in the same place, can not have the same colour. The colour domain of the place p is indicated by $cd(p)$.

In a SWN, also the transitions have a colour domain; the colour domain of a transition is constrained by the colour domains of the places connected to the transition by means of oriented or inhibitor arcs. The colour domain of the transition t is indicated by $cd(t)$. The relation between the the colour domain of the transition and the colour domain of such places, is defined through *arc expressions*.

An arc expression is assigned to an oriented or inhibitor arc connecting the transition to a place, or vice-versa. An arc expression consists of a weighted sum of tuples, where each element of a tuple in turn is a weighted sum of terms denoting multisets of colour classes.

A multiset a over the set A ($A \neq \emptyset$), is the mapping $a \in \{A \rightarrow \mathbb{N}\}$ ($a = Bag(A)$). In other words, a can contain several occurrences of the same element $x \in A$. Formally,

$$a = Bag(A) = \sum_{x \in A} a(x)x \tag{6.1}$$

where $a(x)$ is the coefficient expressing the *multiplicity* of x in a, i. e. the number of occurrences of x in a.

An arc expression is structured according to the colour domain of the place touched by the arc. If the colour domain of the place contains k colour classes, the arc expression is the sum of k-tuples. Given a k-tuple, each of its elements is the weighted sum of terms; the ith element of a k-tuple, is an expression denoting a multiset of C_i, where C_i is the ith colour class composing the colour domain of the

place. The ith element of a k-tuple is the weighted sum of terms; each term can be expressed by means of one of the following *basic function*:

- the *projection* function: this function is indicated by a parameter x denoting an element of C_i.
- The *successor* function: this function is indicated by $!x$, where x is a parameter; this function denotes the successor element of the element of C_i denoted by x (C_i must be an ordered class).
- the *diffusion/synchronization* function: this function is indicated by S_{C_i} and returns the whole set C_i.

A whole arc expression denotes a multiset in the Cartesian product of the colour classes composing the colour domain of the place.

The transitions in a SWN, can be considered to be procedures with parameters; these parameters are present in the arc expressions assigned to the arcs touching the transition. The parameters compose the colour domain of the transition. To each parameter, a colour class is assigned. The colour domain of a transition may be composed even by a *predicate* (guard); a predicate is a Boolean expression expressed over the parameters; given the parameters x and y, the allowed predicates are $x = y$ and $x \neq y$.

A *transition instance* is a transition whose parameters have been instanced to actual values. An actual value is an element of the colour class corresponding to the parameter. An instance of the transition t is indicated by $[t, c]$, where c is the assignment of the parameters of t, to actual values. In a SWN, we can only fire transition instances.

A transition instance $[t, c]$ is enabled to fire iff all the following conditions hold:

- the predicate of $[t, c]$ is evaluated to $true$ (in the case the predicate is present).
- For each oriented arc (p, t), the place p contains the multiset resulting from the evaluation of the arc expression assigned to (p, t).
- For each inhibitor arc (p, t), each tuple contained in p has a smaller multiplicity than the same tuple in the multiset resulting from the evaluation of the corresponding arc expression assigned to (p, t).

The effect of the firing of an instance $[t, c]$ is the following:

- for each oriented arc (p, t), the multiset resulting from the evaluation of the arc expression through the assignment c, is removed from the place p.
- for each oriented arc (t, p), the multiset resulting from the evaluation of the arc expression through the assignment c, is added to the place p.

The same parameters may appear in several arc expressions. If a parameter appears in two (or more) arc expressions assigned to arcs touching the same transition, then the parameter always refer to the same element of the corresponding colour class. If instead a parameter appears in two (or more) arc expression related to different transitions, then there is no relation among the elements referred by the parameter in every arc expression.

Many instances of the same transition may be enabled to fire concurrently; in this case, the firing of one of such transition instances is independent from the firing of the other transition instances.

However, in a SWN, the colour domain of a place may be empty (no colour classes are associated with the place). In this case, the tokens inside the place have a neutral colour such that the tokens inside the place are not distinguishable. An arc connecting a transition to a place $p : cd(p) = \emptyset$ (or vice-versa), has an empty arc expression, but it may have a cardinality. The rules for the transition firing enabling and the firing effect, defined in the GSPN formalism, still hold in a SWN, when a transition has a place with empty colour domain, as input, output or inhibitor place.

Further notions and information on the SWN formalism can be found in [25]. An example of SWN is shown in Fig. 6.2.

6.4 SWN formalism definition

The SWN formalism is given by the tuple $(P, T, A, C, cd, m, pred, f, card, \lambda, w, \pi)$ where

- P is the set of the places.
- $T = T_i \cup T_t$ is the set of the transitions; it is the union of two sets:
  - T_i is the set of the immediate transitions;
  - T_t is the set of timed transitions.
- A is the set of arcs; it is the union of two sets:
  - $A_d \subseteq (P \times T) \cup (T \times P)$ is the set of the oriented arcs;
  - $A_h \subseteq P \times T$ is the set of the inhibitor arcs.
- C is the set of colour classes. The following properties hold:
  - $\forall c \in C, c \neq \emptyset$
  - $\forall c1, c2 \in C : c1 \neq c2, c1 \cap c2 = \emptyset$
- $cd : P \cup T \rightarrow \bigotimes\{C \cup \epsilon\}$ is the function returning the colour domain of a place or a transition.
- m is the function returning the marking of a place p.

  - If $cd(p) \neq \emptyset$, then $m(p)$ is expressed as a multiset of the colour domain of p ($Bag(cd(p))$).
  - If $p : cd(p) = \emptyset$, then $m(p)$ is expressed as the number of tokens inside p.

- $pred$ is the function returning the predicate (if any) of the transition $t : cd(t) \neq \emptyset$.
- f is the function returning the arc expression of assigned to an arc touching a place $p : cd(p) \neq \emptyset$.
- $card : A \rightarrow \mathbb{N} - \{0\}$ is the function returning the cardinality of an arc touching a transition $p : cd(p) = \emptyset$.
- $\lambda : T_t \rightarrow \mathbb{R}^+$ is the function returning the firing rate of a timed transition.
- $w : T_i \rightarrow \mathbb{R}^+$ is the function returning the weight of an immediate transition.
- $\pi : T_i \rightarrow \mathbb{N} - \{0\}$ is the function returning the priority of an immediate transition.
- Given $t \in T$
  - $\bullet_d t = \{p \in P : \exists (p,t) \in A_d\}$ is the set of the places such that an oriented arc is drawn between each of them and t. Such places are referred as input places.
  - $t\bullet_d = \{p \in P : \exists (t,p) \in A_d\}$ is the set of places such that an oriented arc is drawn between t and each of them. Such places are referred as output places.
  - $\bullet_h t = \{p \in P : \exists (p,t) \in A_h\}$ is the set of places such that an inhibitor arc is drawn between each of them and t. Such places are referred as inhibitor places.

6.5 Module based DRPFT analysis

The use of SWNs instead of GSPNs to perform the state space analysis of a DRPFT, leads to reduce the computational costs; a further reduction of the computational effort can be achieved by exploiting the DRPFT modules. As in the case of DFT and RFT models, the state space analysis can be limited to the subtrees of the DRPFT containing some kind of event dependency due to the presence of dynamic gates or RBs. If the quantitative analysis of a DRPFT model is required at time t, an independent subtree (module) requiring the state space analysis can be analyzed in isolation with the proper technique, and replaced by a BE having the same probability of the module to be failed at time t.

In the previous chapters, several definitions of module have been provided according to the current formalism (FT, PFT, DFT, RFT). In each of these cases, some

conditions must hold in order to verify if a subtree is independent from the rest of the model. In order to detect the modules of a DRPFT, we must take into account all the conditions identifying a module in each of the formalisms integrated in the DRPFT formalism. We assume that the DRPFT modules are rooted in IEs, in REs or in the TE. The whole DRPFT model ($\widehat{TE}$) is a module by definition.

6.5.1 Modules detection and classification

The first step to detect the DRPFT modules consists of detecting the structurally independent subtrees; this can be performed by the modules detection algorithm for FTs (section 2.5.2), applied to the DRPFT ignoring the presence of the RBs and of the arcs connecting the RBs to the events and vice-versa, and inverting the orientation of the other arcs.

In the second step, appropriate conditions on the parameter set of the root of every structurally independent subtree, are checked in order to verify the presence of parameterized shared subtrees. So, a subtree $\widehat{e'}$ ($e' \in \mathcal{E} - (\mathcal{BE} \cup \mathcal{BRE})$) is structurally and parametrically independent if two condition holds:

- the subtree is structurally independent
- $\forall e \in \mathcal{E} : e \in {}^{\circ}e', \theta(e) \supseteq \theta(e')$

In the third and final step, we have to consider the presence of RBs; so, a structurally and parametrically independent subtree $\widehat{e'}$ is a DRPFT *module* if the following condition holds:

$$\nexists b \in \mathcal{RB} : e' \in {}^{\circ}b - b^{\bullet}$$

DRPFT modules can be classified according to the technique they need to be analyzed:

- the module $\widehat{e}$ is a *Combinatorial Solution Module* (CSM) if both the following conditions hold:
  - $\nexists e' \in \mathcal{E} \wedge \nexists b \in \mathcal{RB} : e' \in {}^{\circ}e \wedge e' \in Cov_E(b)$
  - $\forall g \in \mathcal{G} : g \in {}^{\circ}e, \gamma(g) \in \mathcal{BG}$

  In other words, the module $\widehat{e}$ is a CSM if any of its events does not belong to the coverage set of a RB, and the module does not contain any dynamic gate.
- the module $\widehat{e}$ is *State space Solution Module* (SSM) if at least one of the following condition holds:
  - $\exists e' \in \mathcal{E} \wedge \exists b \in \mathcal{RB} : e' \in {}^{\circ}e \wedge e' \in Cov_E(b)$
  - $\exists g \in \mathcal{G} : \gamma(g) \in \mathcal{DG} \wedge g \in {}^{\circ}e$

In other words, the module $\widehat{e}$ is a SSM if it contains at least one event belonging to the coverage set of a RB, or if it contains at least one dynamic gate.

For the CSMs the state space analysis is not necessary; for them, the less expensive combinatorial analysis is enough. The state space analysis is essential for the SSMs, since they contain dependencies due to the presence of dynamic gates or RBs. We indicate with $\mathcal{M}$ the set of the modules in the DFT.

The definition of MDM (section 4.7.1) and the definition of MRM (section 5.5.1) still hold in the DRPFT formalism. In a DRPFT, a module may be contemporary a MDM and a MRM.

6.5.2 DRPFT modularization

The modularization of a DRPFT model, requires some new steps with respect to the modularization of a DFT or RFT module:

1. Modules detection
2. Modules classification: for each module, we verify if it is a MDM or a MRM.
3. if $\widehat{TE}$ is a MDM or a MRM go to step 12, else go to step 4.
4. Decomposition: each module classified as MDM or MRM is detached from the DRPFT.
5. Module simplification: the parameter set of the module root event is subtracted from the parameter set of every event in the module.
6. MDM/MRM conversion to SWN.
7. SWN analysis: each MDM/MRM in SWN form, is analyzed.
8. Aggregation: each detached MDM/MRM is replaced in the DRPFT, by a BE/BRE (we obtain a PFT).
9. PFT conversion to pBDD
   (or PFT unfolding to FT followed by FT conversion to BDD).
10. (p)BDD quantitative analysis returning the final result.
11. end.
12. DRPFT conversion to SWN.
13. SWN analysis: the DPRFT in SWN form, is analyzed returning the final result.
14. end.

We limit our attention to the quantitative analysis of a DRPFT model at time t.

First, modules must be detected and classified. In the following Decomposition step, the modules to be detached are those classified as MDM or MRM (the same module may be classified as both a MDM or a MRM).

At this point, a problem arises: the root event e of a module may be an event having a non empty parameter set; in this case, e folds several events according to its parameter set, while the root event of a module should be a unique event. A way to solve this problem, consists of removing from the parameter set of every event in the module $\widehat{e'}$ (including e'), the parameter set of e'. Since a module of a DRPFT model is a parametrically independent subtree (section 6.5.1), every event e inside $\widehat{e'}$ includes the parameters of e' in its parameter set. Formally, $\forall e \in \mathcal{E}$: $e \in \widehat{e'}, \theta(e') \subseteq \theta(e)$. So, before the conversion into SWN of a MDM/MRM $\widehat{e'}$, we perform on every event $e \in \widehat{e'}$, this operation: $\theta(e) = \theta(e) - \theta(e')$. We call this new step module simplification. Moreover, in this step, if the module root event is a RE, it is converted to an IE, because its parameter set becomes empty.

Then, the module (including the RBs and the arcs touching the RBs) can be converted into SWN by means of the model transformation system reported in appendix A (Conversion step). The module in SWN form can be analyzed computing the probability of the place corresponding to the root event of the module, to be marked at time t (Analysis step).

In the Aggregation step, if the module root event before the module simplification step, was an IE, the module is replaced in the DRPFT by a BE with the probability to occur computed on the correspondent SWN; if the module root event was a RE before the simplification step, it is replaced by a BRE. In both cases, the parameter set of the B(R)E replacing the module, is equal to the parameter set of the module root event before the simplification step.

After the Aggregation step, the DRPFT contains no dynamic gates and no RBs; now, the model is actually a PFT. If the current DRPFT respects the restrictions on the use of parameters reported in section 3.5.1, the corresponding pBDD can be generated and analyzed; if such restrictions are not respected, the DRPFT can be unfolded [11] (i. e. converted to a FT model), and the corresponding BDD can be generated and analyzed.

The analysis of the (p)BDD returns the final result, i. e. the probability of the TE at time t (system Unreliability or Unavailability). It may happen that the whole DRPFT is a MDM or a MRM; in this case, the whole model is mapped to SWN and analyzed in this form.

6.6 Running example

In this section, we refer to the Multiproc system and to its DFT model described in section 4.3.1. Several redundancies and symmetries can be observed in the Multiproc system: we have three processing units ($PU1$, $PU2$, $PU3$) composed by the same type and the same number of components: one processor and one inter-

nal memory; we have two identical spare memories ($R1$, $R2$), and each of them is functionally dependent on the memory bus B connecting the shared memories to the processing units. Moreover, every spare memory is available to replace any failed internal memory ($M1$, $M2$, $M3$).

Such symmetries and redundancies in the system lead to the presence in the DFT model in Fig. 4.4, of several subtrees with the same structure; for instance, subtrees $\widehat{PU1}$, $\widehat{PU2}$, $\widehat{PU3}$ are isomorphic and their BEs concern the same types of components. Moreover, the BEs referring to the same class of components (for instance, the processors) have the same failure rate. The BEs $R1$ and $R2$ contemporary belong to the subtrees $\widehat{PU1}$, $\widehat{PU2}$, $\widehat{PU3}$. Both $R1$ and $R2$ are the input events of three gates of type WSP whose output events are the event $MEM1$, $MEM2$, $MEM3$, respectively. Moreover, both $R1$ and $R2$ are the dependent events of the same $FDEP$ gate having B as trigger event.

The symmetric structure of the DFT model in Fig. 4.4 is reflected in the equivalent GSPN in Fig. 4.26, where symmetric subnets are present and represent the failure mode of the redundant parts of the system.

6.6.1 The DRPFT model

If we model the same system as a DRPFT, the symmetric subtrees are folded in a parametric subtree, as shown in Fig. 6.1, where the DFT subtrees $\widehat{PU1}$, $\widehat{PU2}$ and $\widehat{PU3}$ are folded in $\widehat{PU(i)}$ with the parameter i of type $C1 = \{1, 2, 3\}$; the BRE $R(j)$ with $\tau(j) = C2 = \{1, 2\}$, folds the DFT BEs $R1$ and $R2$.

Fig. 6.2 shows the conversion to SWN of the DRPFT in Fig. 6.1; comparing the GSPN in Fig. 4.26 and the SWN in Fig. 6.2, we can observe that the symmetric subnets present in the GSPN are folded in a coloured subnet of the SWN. In this way, the SWN formalism allows to maintain the parametric form concerning the redundant parts of the system.

A size reduction can be observed also in the state space (CTMC) derived from the SWN, with respect to the dimensions of the state space derived from the GSPN. This is due to the fact that we can derive a symbolic state space from the SWN. The state space generated from the GSPN in Fig. 4.26 by means of the *GreatSPN* tool, is composed by 7806 states (7806 tangible markings in the GSPN reachability graph), while the symbolic state space derived from the SWN in Fig. 6.2 is composed by 1374 symbolic states (1374 tangible markings in the SWN symbolic reachability graph). Moreover, the time to analyze the GSPN by means of the *GreatSPN* tool, consists of 12 seconds; the analysis of the SWN instead, requires 4 seconds[1]. Thus the use of the parametric form allows to reduce the computational cost of the system analysis both in terms of state space size and in terms of time.

The probability of the TE is computed on the SWN in Fig. 6.2, as the probability of the place TE_dn (corresponding to the TE of the DRPFT in Fig. 6.1) to

[1]Both the GSPN and SWN analysis have been performed on a personal computer equipped with a Pentium 4 2.4 MHz processor and with 512MB of RAM.

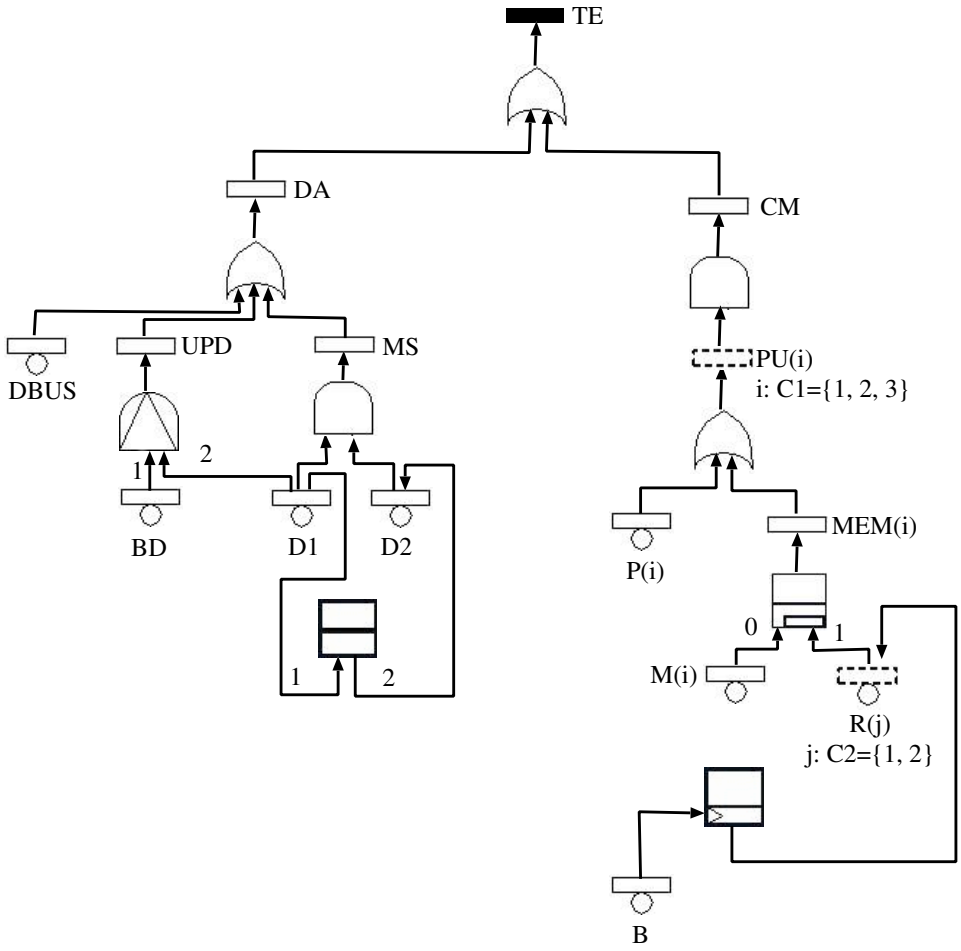

Figure 6.1: DRPFT model for the Multiproc system with dependencies (without repair).

be marked at a certain mission time:

$$Pr\{TE, t\} = Pr\{m(TE\_dn) = 1, t\}$$

We obtained the same results given by the DFT model in Fig. 4.4, and reported in Tab. 4.3, but decreasing the computational cost.

Modularization

A more evident reduction of the state space size, is achievable if we perform the analysis of the DRPFT in Fig. 6.1 exploiting modules. In such model two MDMs are present: $\widehat{DA}$ and $\widehat{CM}$ (Fig. 6.3). The module $\widehat{DA}$ is not in parametric form, so its conversion into SWN actually produces a GSPN (shown in Fig. 6.4); the

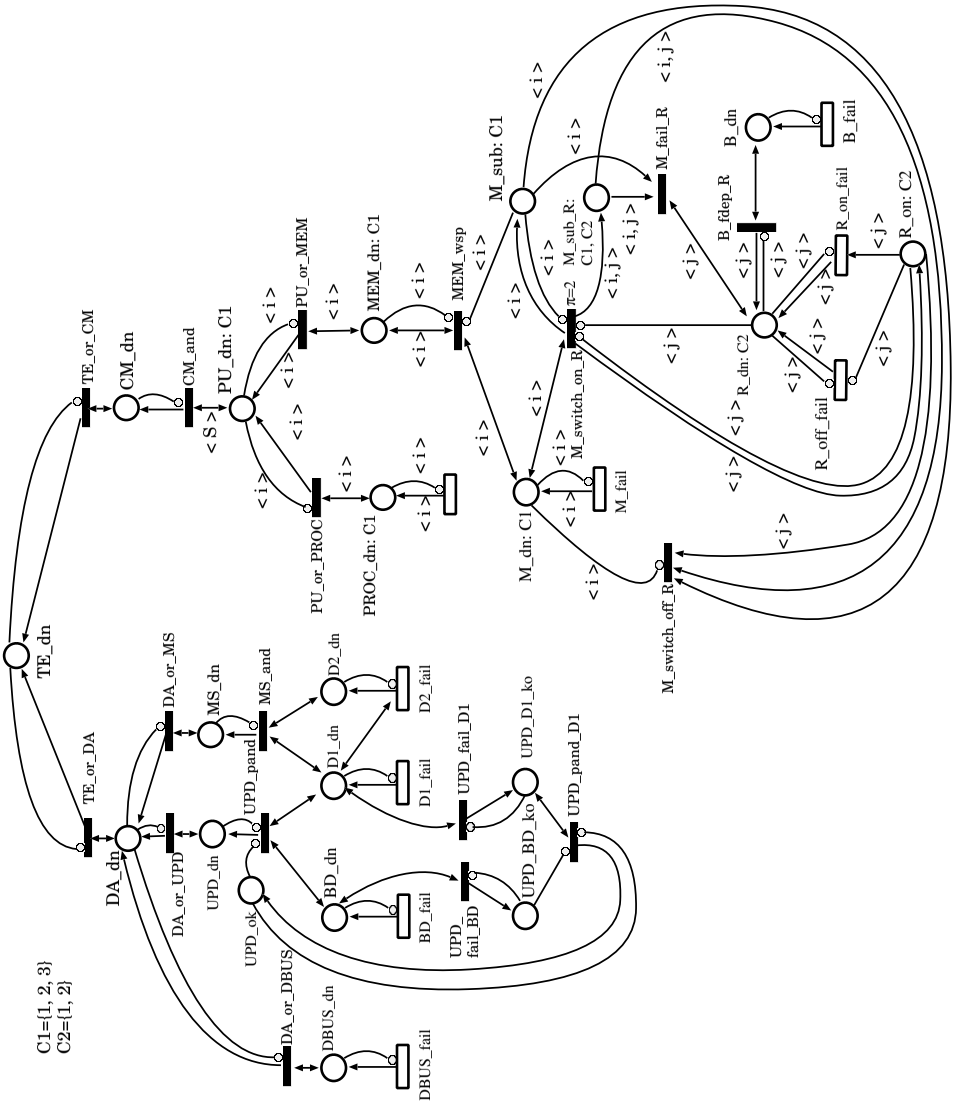

Figure 6.2: SWN equivalent to the DRPFT model in Fig. 6.1.

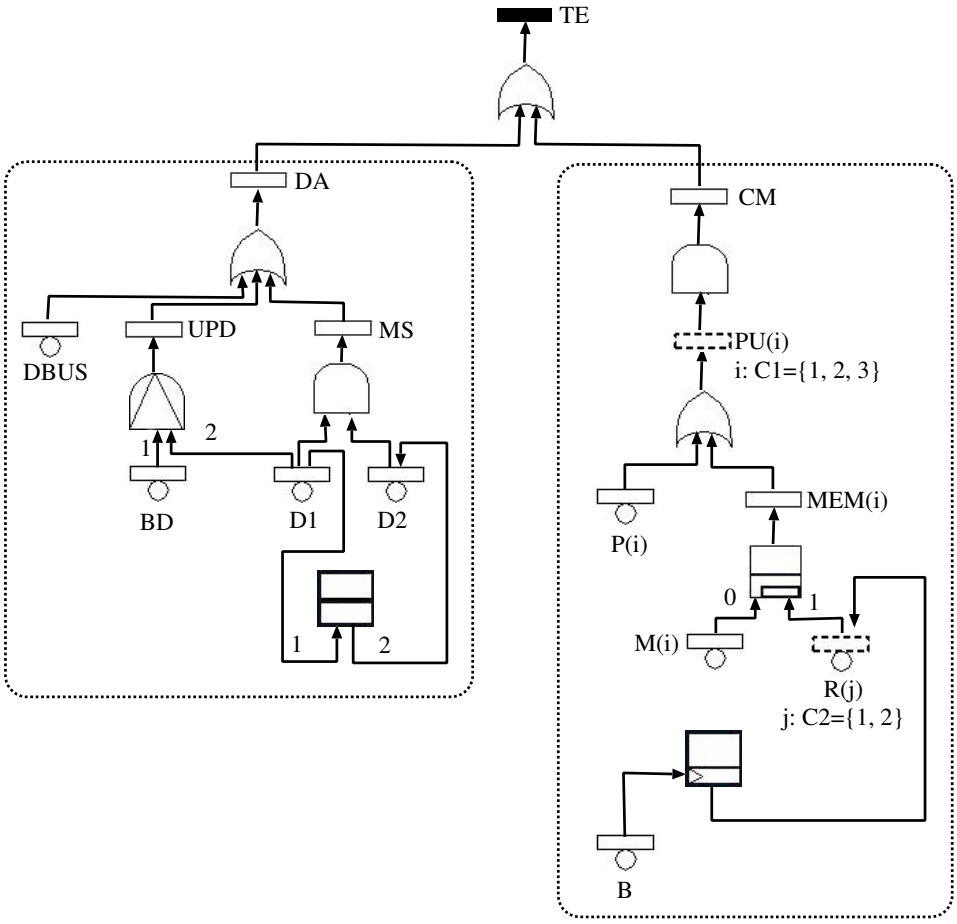

Figure 6.3: The MDMs in the DRPFT model of the Multiproc system with dependencies (without repair).

result of the conversion into SWN of the module $\widehat{CM}$, is shown in Fig. 6.5. Both modules have been converted into SWN by means of the model transformation system provided in appendix A.

The state space for the module $\widehat{DA}$ consists of 14 states, while the state space for the module $\widehat{CM}$ is composed by 85 states. The state space obtained from the complete mapping to SWN of the DRPFT model, was instead composed by 1374 states. Moreover, for both modules, the time to perform their state space analysis is less than 1 second.

The probabilities obtained for the modules and for the whole system on the DRPFT model, are the same as those reported in Tab. 4.3 and obtained by the modularization of the DFT model in Fig. 4.4 (see section 4.7.3).

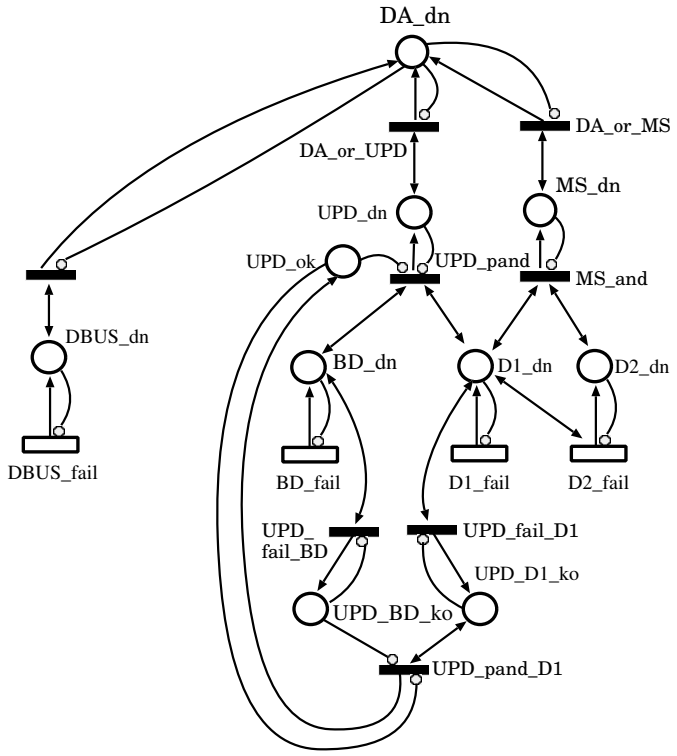

Figure 6.4: The SWN corresponding to the module $\widehat{DA}$ of the DRPFT in Fig. 6.1.

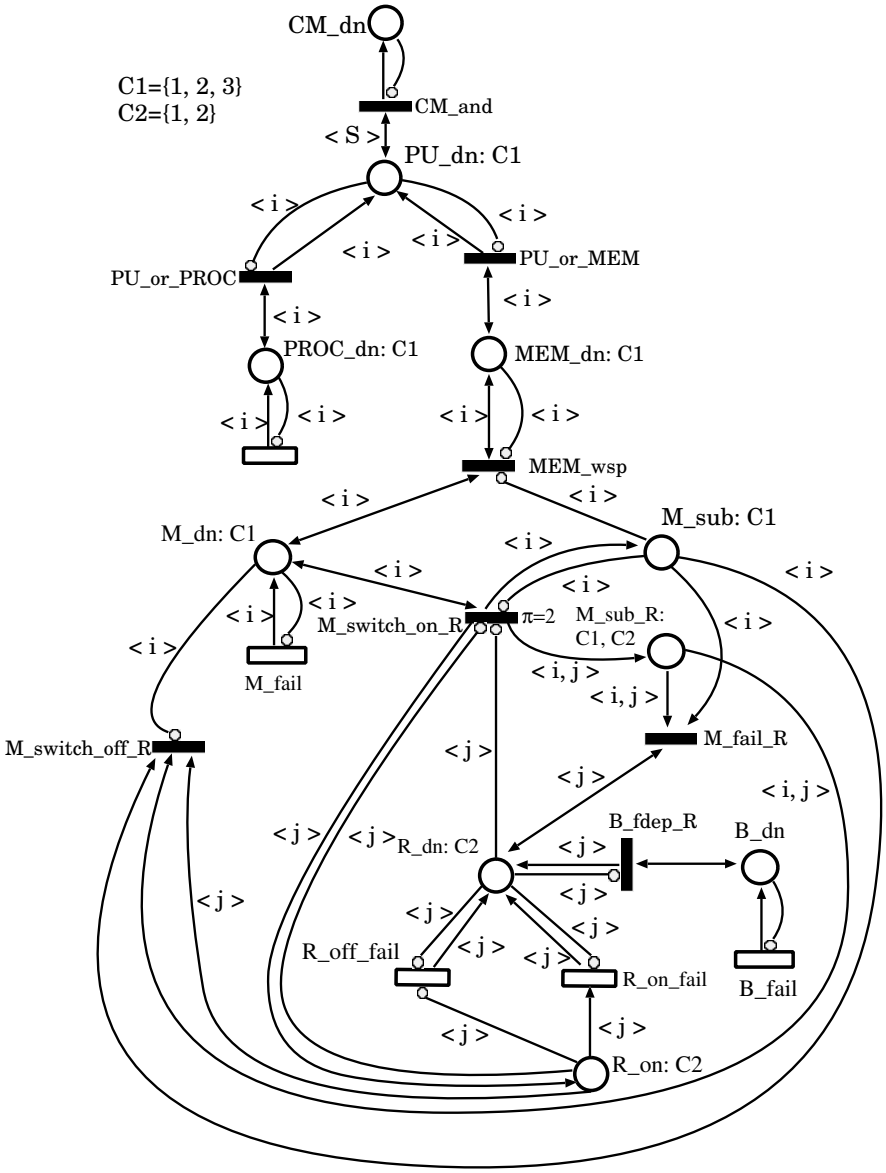

Figure 6.5: The SWN corresponding to the module $\widehat{CM}$ of the DRPFT in Fig. 6.1.

# p. u.	$\widehat{CM}$ size (DFT)	ord. st. sp. size	$\widehat{CM}$ size (DRPFT)	sym. st. sp. size
3	16	487	7	85
4	20	2479	7	157
5	24	12703	7	263
6	28	64447	7	410
7	32	321663	7	605
8	36	1577727	7	855
9	40	7612927	7	1167
10	44	36199423	7	1548

Table 6.1: State space size of the module $\widehat{CM}$ according to the number of processing units.

6.6.2 Parametric form efficiency

In this section, we investigate how the state space size of the module $\widehat{CM}$ changes if we increase the number of processing units in the system. In order to model such increase in the DRPFT of the system, we only have to change the cardinality of the type $C1$ associated with the parameter i declared in the RE $PU(i)$ and identifying the processing units. If instead, we want to model the increase of the number of the processing units in the DFT (Fig. 4.4), we have to add some new subtrees in the model.

For a number of processing units varying from 3 to 10, Tab. 6.1 indicates the following measures:

- the size of the module $\widehat{CM}$ of the DFT model in terms of number of events.
- the size of the ordinary state space derived from the module $\widehat{CM}$ of the DFT.
- the size of the module $\widehat{CM}$ of the DRPFT model in terms of number of events.
- the size of the symbolic state space derived from the module $\widehat{CM}$ of the DRPFT.

Observing the values in Tab. 6.1, the size reduction of the model and of the state space size due to the parametric form, is evident.

The size of the DFT module $\widehat{CM}$ is given by the formula

$$1 + 4 \cdot I + 3$$

This formula is the sum of the following addends:

- 1 takes into account the root event CM.
- $4 \cdot I$ is the number of events concerning the processing units; I is the number of processing units.

- 3 takes into account the events $R1$, $R2$, B.

The size of the module $\widehat{CM}$ in the DRPFT instead, is constant: only the cardinality of $C1$ changes according to the number of processing units: $I = |C1|$.

The ordinary state space is obtained from the GSPN equivalent to the DFT module $\widehat{CM}$, while the symbolic state space size is obtained from the SWN equivalent to the DRPFT module $\widehat{CM}$. The respective size have been computed by means of the *GreatSPN* tool.

6.6.3 Repairable version of the system

In this section, we add some repair processes in the Multiproc system described in section 4.3.1. The DRPFT model of this version of the Multiproc system, is shown in Fig. 6.6.

We assume the presence of a repair process recovering both the disk $D1$ and the disk $D2$, the disk bus $DBUS$, and the device BD performing the periodical update of $D2$. The repair process is activated when the access to the disks is compromised. This repair process is represented in the DRPFT model in Fig. 6.6 by the RB $REP1$ whose trigger event is the IE DA representing the impossibility to access the disks, due to the failure of the disk bus (event $DBUS$), to the failure of both disks (event MS), or to the missing update of $D2$ (event UPD). The basic coverage set of $REP1$ is $Cov_{BE}(REP1) = \{DBUS, BD, D1, D2\}$. The coverage set of $REP1$ is $Cov_E(REP1) = Cov_{BE}(REP1) \cup \{UPD, MS, DA, TE\}$.

Other repair processes are present in this version of the Multiproc system. For each processing unit, a repair process is present, acts on the processor and on the internal memory of the processing unit, and is activated as soon as the failure of the processing unit occurs. As the failure of the processing units is represented in parametric form in the DRPFT in Fig. 6.6, also their repair is represented in parametric form by the RB $REP2$ whose trigger event is the event $PU(i)$, while its basic coverage set is $Cov_{BE}(REP2) = \{P(i), M(i)\}$. In this way, we model the presence of several repair processes, each activated by the occurrence of an instance of the RE $PU(i)$ identified by a possible value of the parameter i, and acting on the instances of $P(i)$ and $M(i)$ for the same value of i. The coverage set of $REP2$ is $Cov_E(REP2) = Cov_{BE}(REP2) \cup \{MEM(i), PU(i), CM, TE\}$.

The repair policy associated to every RB (repair process) is the GRT policy (section 5.3.1) with repair rate equal to $0.001h^{-1}$.

The structurally independent subtrees in the DRPFT model in Fig. 6.6 are the following: $\widehat{TE}$, $\widehat{DA}$, $\widehat{CM}$, $\widehat{PU(i)}$, $\widehat{MEM(i)}$. The structurally and parametrically independent subtrees are: $\widehat{TE}$, $\widehat{DA}$, $\widehat{CM}$. The modules are: $\widehat{TE}$, $\widehat{DA}$, $\widehat{CM}$; all of them are SSMs. The module $\widehat{DA}$ and the module $\widehat{CM}$ can be classified as both MDM and MRM. Fig. 6.6 shows the MDM/MRM in the DRPFT model. The conversion of $\widehat{DA}$ in SWN is shown in Fig. 6.7, while the conversion of $\widehat{CM}$ in SWN is shown in Fig. 6.8.

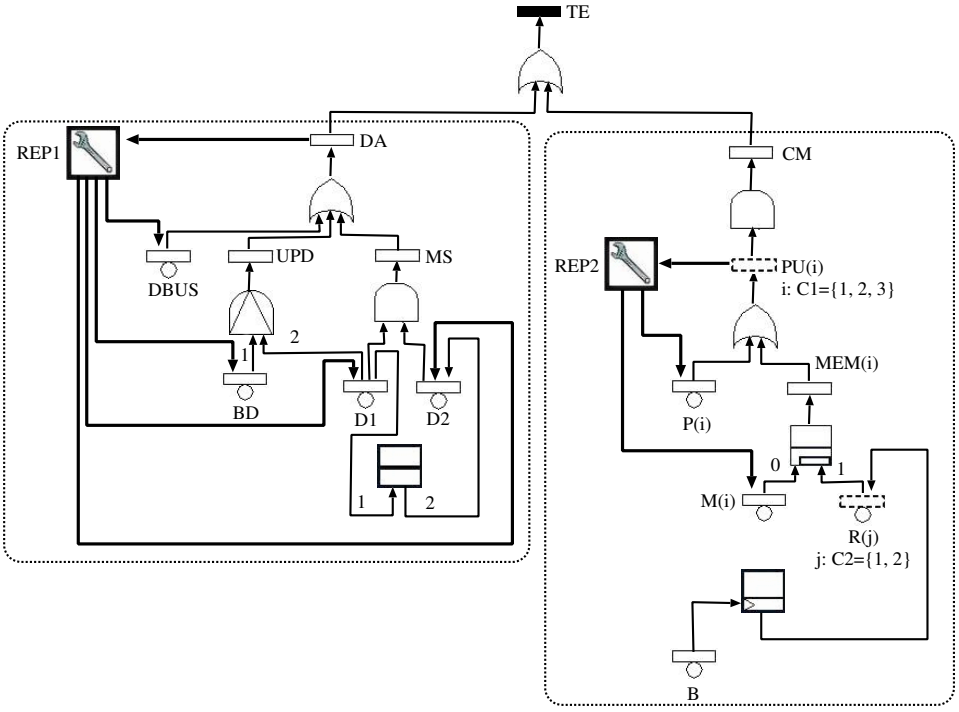

Figure 6.6: The DRPFT model for the Multiproc system with dependencies and repair.

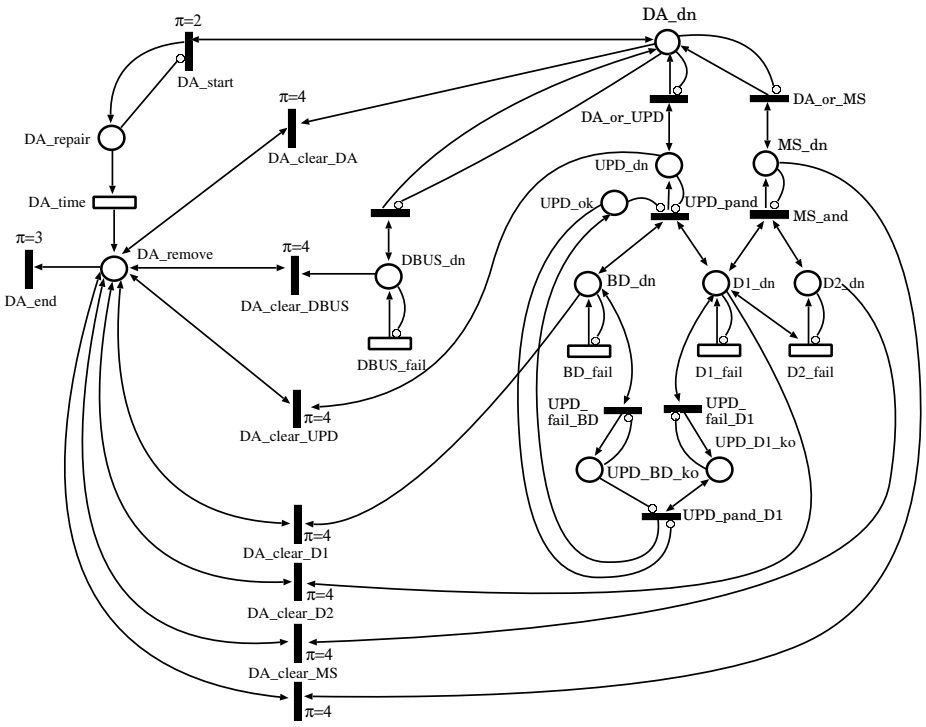

Figure 6.7: The SWN corresponding to the module $\widehat{DA}$ of the DRPFT in Fig. 6.6.

In the SWN corresponding to the module $\widehat{DA}$ (Fig. 6.7), the subnet representing the repair process, clears the places corresponding to the events belonging to the module and included in the coverage set of the RB $REP1$.

In the SWN in Fig. 6.8 (corresponding to the module $\widehat{CM}$), the subnet generated by the conversion of the RB $REP2$, is "coloured"; in this way, this subnet models several repair processes, each involving one of the processing units. The aim of the immediate transition called $M\_switch\_off\_R$ is modelling the state transition from the working state to the dormant state of the spare component replacing the main one when the repair of the latter is completed.

If we want to perform the quantitative analysis of the system at time $t = 10000h$, we have to compute on the SWN in Fig. 6.7 the probability of the place DA_dn to be marked at time $t = 10000h$, while we have to compute on the SWN in Fig. 6.8 the probability of the place CM_dn to be marked at the same time. We obtain these probability values:

$$Pr\{DA, t\} = Pr\{m(DA\_dn) = 1, t\} = 8.216427E - 6$$
$$Pr\{CM, t\} = Pr\{m(CM\_dn) = 1, t\} = 1.25E - 10$$

In the DRPFT, the module $\widehat{DA}$ is replaced by the BE DA having $8.216427E - 6$ as probability to occur; the module $\widehat{CM}$ is replaced by the BE CM whose prob-

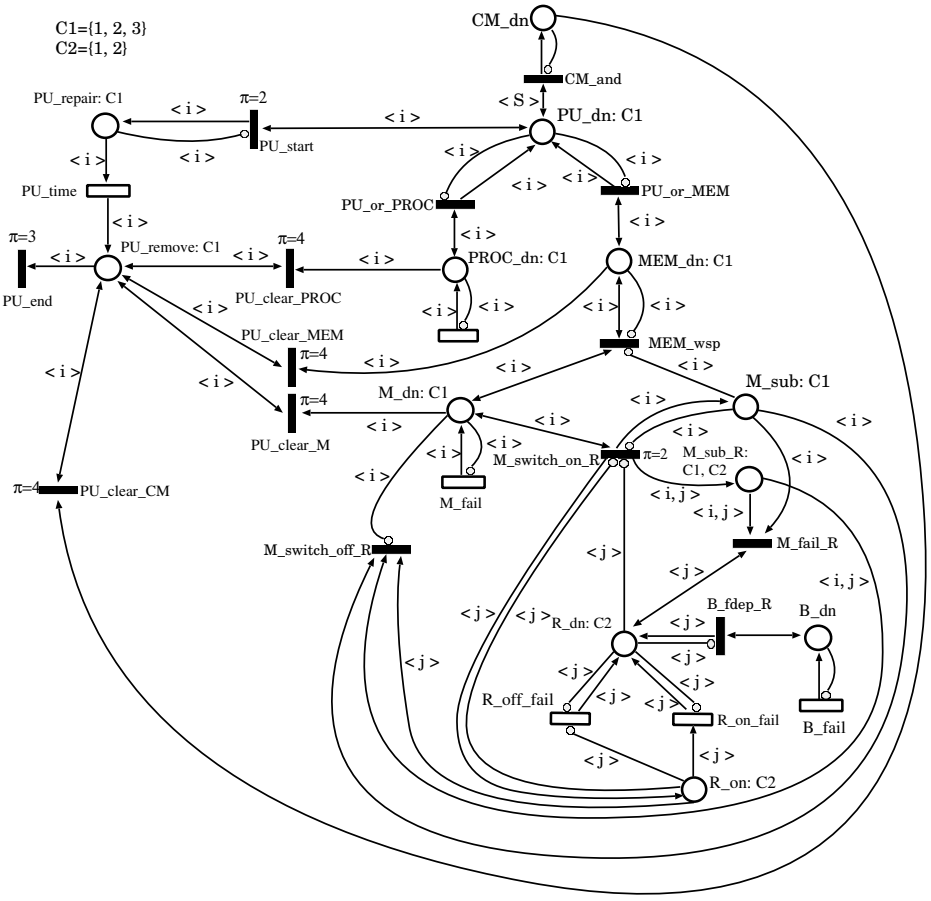

Figure 6.8: The SWN corresponding to the module $\widehat{CM}$ of the DRPFT in Fig. 6.6.

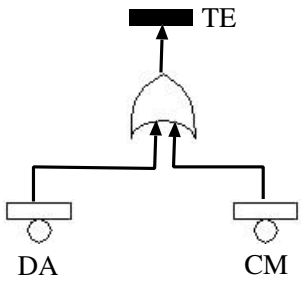

Figure 6.9: The DRPFT model after the analysis of the MDM/MRMs.

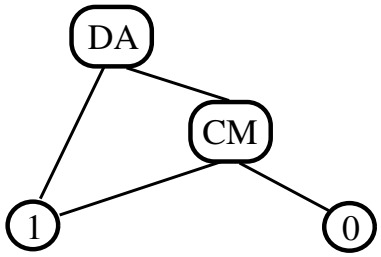

Figure 6.10: The pBDD corresponding to the (DR)PFT model in Fig. 6.9.

ability is $1.25E-10$. In this way, we obtain the DRPFT in Fig. 6.9; from it, the corresponding pBDD can be generated and is shown in Fig. 6.10. The analysis of this pBDD returns the unavailability of the system: $8.216552E-6$.

Tab. 6.2 reports the probabilities obtained for the modules and for the whole system, and computed on the DRPFT model in Fig. 6.6, for a mission time varying from $1000h$ to $10000h$.

6.7 A software framework for DRPFT analysis

All the concepts introduced so far on the DRPFT analysis, have been implemented in a software framework [8, 31] for the quantitative analysis of DRPFT models. Fig. 6.11 shows the architecture of the framework.

The DRPFT models are drawn by means of the customizable graphical user interface called *Draw-Net* [53, 58, 59, 60, 106, 109]; *Draw-Net* can be adapted to draw any kind of graph based models; the formalisms must be defined in XML format files; in a formalism, several aspects have to be defined, such as the primitives of the formalism, the measure to be computed on the models, and the graphical representation of the formalism primitives. The most recent version of *Draw-Net* allows to draw multi-formalism models [58]. Once the model has been drawn by the user, the model can be saved in a file, still in XML format, together with the

time t	$Pr\{DA, t\}$	$Pr\{CM, t\}$	$Pr\{TE, t\}$
1000 h	1.520130E-6	3.2E-11	1.520162E-6
2000 h	2.518513E-6	8.1E-11	2.518594E-6
3000 h	3.324223E-6	1.07E-10	3.324330E-6
4000 h	4.058319E-6	1.18E-10	4.058437E-6
5000 h	4.765339E-6	1.22E-10	4.765461E-6
6000 h	5.461667E-6	1.24E-10	5.461791E-6
7000 h	6.153333E-6	1.24E-10	6.153457E-6
8000 h	6.842556E-6	1.25E-10	6.842681E-6
9000 h	7.530153E-6	1.25E-10	7.530278E-6
10000 h	8.216427E-6	1.25E-10	8.216552E-6

Table 6.2: Unavailability values for the Multiproc system with dependencies and repair.

specification of the results request.

In our case, *Draw-Net* has been adapted to draw DRPFT models; the user can specify at which mission time the quantitative analysis of the DRPFT must be performed. When the user invokes the analysis of the DRPFT model, *Draw-Net* saves the model and the mission time specification in a XML file (indicated as $DRPFT.xml$ in Fig. 6.11), and is passed to the *DRPFTproc* block for the modules detection and classification (section 6.5.1); each MDM/MRM in the model is saved in a XML file and passed to the *DRPFT2SWN* block (in Fig. 6.11 the XML files for the modules are indicated as $mod1.xml$, $mod2.xml$, ...). In the XML file containing the module, also the mission time is indicated.

The block $DRPFT2SWN$ implements the model transformation system from DRPFT to SWN, expressed by means of compound rules in appendix A. The block $DRPFT2SWN$ receives one or several modules, each contained in a XML file; $DRPFT2SWN$ translates every module in a SWN which is saved in a couple of files according to the *GreatSPN* tool [22, 26] format. For instance the module inside $mod1.xml$ is converted into the SWN described in the files $mod1.net$ and $mod1.def$; $mod1.net$ contains the specification of the places, the transitions and the arcs of the SWN, while $mod1.def$ contains the specification of the colour classes used in the SWN, and of the results to be computed on the SWN.

$DRPFT2SWN$ specify in the $.def$ file of each module that the following result has to be computed: the probability to be marked at the mission time of the place corresponding to the module root event.

GreatSPN is a tool developed for the design, the analysis and the simulation of GSPN or SWN models. In our framework, the SWN analyzer developed for *GreatSPN* is exploited. The $.net$ and $.def$ files containing the description of every SWN are passed to the *GreatSPN* SWN analyzer computing the result indicated in the $.def$ file. Once all the SWNs corresponding to the DRPFT modules, have been processed by the SWN analyzer, the obtained result for each SWN is passed back

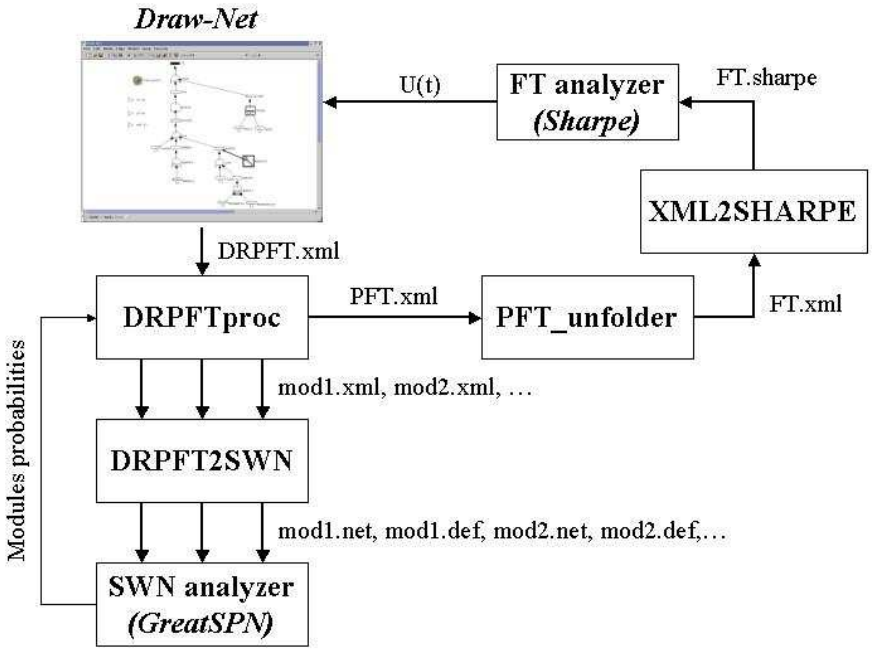

Figure 6.11: The architecture of the framework for the DRPFT analysis.

to the $DRPFTproc$ block.

Now, $DRPFTproc$ replaces the MDM/MRMs in the DRPFT with B(R)Es; the probability to occur of every B(R)E replacing a module, is the probability computed on the SWN corresponding to the module. By replacing the MDM/MRMs in the DRPFT, a PFT is obtained. The PFT is saved by $DRPFTproc$ in a XML file (indicated as $PFT.xml$ in Fig. 6.11) containing also the indication of the mission time. This file is passed to $PFT\_unfolder$ performing the PFT unfolding (PFT conversion to FT); the equivalent FT is saved by $PFT\_unfolder$ in a XML file (indicated as $FT.xml$ in Fig. 6.11). This XML file is passed to the $XML2SHARPE$ block which converts the FT from the XML format to the *SHARPE* [82, 83] tool format.

SHARPE is a tool developed for the analysis of several kinds of models, such as FTs, CTMCs and GSPNs. In our case, *SHARPE* is used to perform the quantitative analysis of the final FT at the required mission time. The result returned by *SHARPE* is the probability of the TE at the required mission time; this value is passed back to *Draw-Net* to be showed to the user.

Fig. 6.12 shows a screenshot of *Draw-Net* used as graphical interface in our framework for the quantitative analysis of DRPFT models.

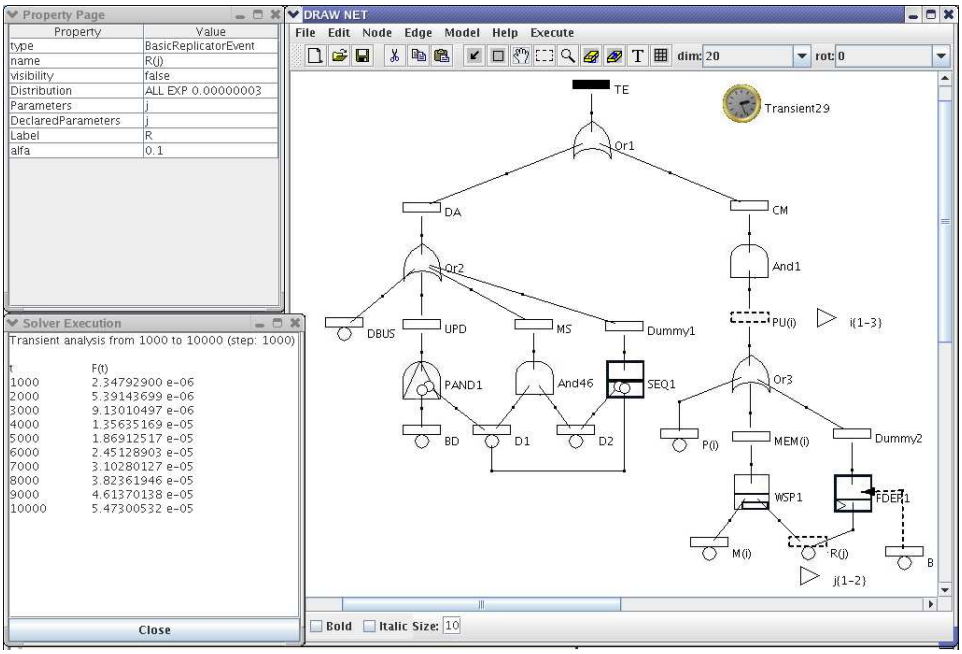

Figure 6.12: A screenshot of *Draw-Net*.

Chapter 7

Conclusions

In this work, we presented several extensions to the FT formalism; each extension introduces new primitives with the purpose of increasing the FT modelling capacity. Each extended FT formalism (PFT, DFT, RFT) is oriented to the modelling of particular features characterizing the behaviour or the failure mode of the system: the PFT formalism is oriented to represent systems with redundant parts, the DFT formalism is oriented to model dependencies in the failure mode, and the RFT formalism allows to model both the failure and repair mode of a system.

For each extended FT formalism, we proposed a way to perform the analysis of the models built according to that formalism; in the case of the PFT formalism, we introduced a new version of BDD called pBDD with the aim of performing the analysis of a PFT model exploiting both the efficiency of BDDs and the parametric form for the compact representation of the redundancies in the system. In the case of the DFT and RFT formalism, our solution method is based on the conversion of DFT and RFT models (or of their modules) into GSPNs by means of model-to-model transformation based on graph transformation rules. In the solution methods proposed for each formalism, our intent is reducing the computational cost of the analysis; we achieved this purpose by means of the parametric form or by means of the model modularization.

Finally, we integrated all the extended FT formalisms described in this work, in a unique formalism called DRPFT collecting and integrating all the primitives introduced in each extended FT formalism. The analysis of DRPFT models is supported by the conversion of DRPFT models (or of their modules) into SWNs (the "coloured" version of GSPNs).

Actually, a model designer could represent the system behaviour directly in form of SWN, but this way of modelling the system, is usually rather complicated with respect to the more practical and intuitive modelling in form of DRPFT; for instance, if we compare the module $\widehat{CM}$ in the DRPFT model in Fig. 6.6 with the equivalent SWN depicted in Fig. 6.8, we can observe how the DRPFT is easier to be drawn, decisely more intuitive to be interpreted, and more compact in terms of model size.

So, the DRPFT formalism can be interpreted as an high level language allowing the model designer to express in a straightforward way the relations among the failure and repair events in the system, while their modelling in form of SWN is obtained by the automatic conversion of the DRPFT model (or modules) into SWN.

Open problems

Some aspects concerning the formalisms described in this work, can be the object of further studies and development. For instance, in the case of the PFT models, their analysis by resorting to pBDDs is possible if some restrictions about the use of parameters are respected; the pBDD based technique for the analysis of PFT models could be extended in order to remove such restrictions. Moreover, the pBDD technique have been developed for PFT models where the gates can be of type AND or of type OR; actually the PFT formalism includes also the $K : N$ gate, so the PFT solution method based on pBDD needs to be extended in order to deal with gates of type $K : N$ as well.

In this work, the $K : N$ gate has been omitted in the DFT, RFT and DRPFT formalism, but the mapping of this type of gate to GSPN (or SWN), can be easily implemented by introducing other compound rules in the model transformation systems from DFT (or RFT) to GSPN, and from DRPFT to SWN: in [30], the transformation rule to convert to GSPN a $K : N$ gate of a DFT, is provided; in [11], the way to map in SWN form a $K : N$ gate, is described, but the authors limit their attention to the case of a $K : N$ gate having a unique (B)RE as input event (such input event folds N input events).

In the case of RFTs, new repair policies might be defined and represented: we limited our attention to policies where the trigger condition is a single failure event; it would be of interest evaluating policies where the recovery process is oriented to the preventive maintenance of the system, instead of the repair of the failed subsystem.

The evaluation methods that we proposed in this work for DFT, RFT and DRPFT models, concern exclusively the quantitative analysis of the models with the aim of computing the probability of the TE at a given time. The computation of the importance measures (section 2.3.3), the qualitative analysis and the computation of the MCS probabilities, are still open problems for these kinds of model.

Future directions

A parallel research direction on DFT analysis, is oriented to the use of *Bayesian Networks* (BN) and *Dynamic Bayesian Networks* (DBN) [13, 73, 74, 77], instead of CTMCs or GSPNs. BNs and DBNs are based on the concept of conditional probability and their use to support DFT analysis has several advantages such as the computation of diagnostic and importance indices in a straightforward way. Exploiting DBNs for the analysis of DRPFT models would be of interest.

Though the DRPFT formalism collects all the primitives introduced in the PFT, DFT and RFT formalism, the modelling capacity of a DRPFT model may not be enough to fit the behaviour of a very complex system. In this case, we can build *multi-formalism* [58, 84, 107] models; this means creating a model composed by several submodels, each conforming to a certain formalism. In this way, instead of referring to a single formalism, several formalisms are involved in the model. Every aspect of the system behaviour is modelled according to the formalism which is the most suitable to fit that aspect. The submodels of a multi-formalism model interact with each other by exchanging measures or by means of common nodes. The extended FT formalisms can be involved in a multi-formalism model, together with other formalisms not deriving from the FT one.

Bibliography

[1] M. Ajmone-Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. J. Wiley and Sons, 1995.

[2] Anand and A. K. Somani. Hierarchical Analysis of Fault Trees with Dependencies, Using Decomposition. In *Proc Annual Reliability and Maintainability Symposium*, pages 69–75, 1998.

[3] M. Andries, G. Engels, and A. Habel. Graph Transformation for Specification and Programming. *Science of Computer Programming*, 34(1):1–54, April 1999.

[4] T. Assaf and J. B. Dugan. Automatic generation of Diagnostic Expert Systems from Fault Trees. In *Annual Reliability and Maintainability Symposium 2003 Proceedings*, Tampa, FL USA, January 2003.

[5] T. Assaf and J. B. Dugan. Diagnostic Expert Systems from Dynamic Fault Trees. In *Annual Reliability and Maintainability Symposium 2004 Proceedings*, pages 444–450, Los Angeles, CA USA, January 2004.

[6] L. Baresi and R. Heckel. Tutorial introduction to graph transformation: A software engineering perspective. In *Proceedings of the International Conference on Graph Transformations*, Barcelona, Spain, 2002.

[7] Z. W. Birnbaum. On the importance of different components and a multicomponent system. In P. R. Korishnaiah, editor, *Multivariable analysis II*. Academic Press, New York, 1969.

[8] A. Bobbio and D. Codetta-Raiteri. Parametric Fault-trees with dynamic gates and repair boxes. In *Proc. Reliability and Maintainability Symposium*, pages 459–465, Los Angeles, CA USA, January 2004.

[9] A. Bobbio, D. Codetta-Raiteri, M. De Pierro, and G. Franceschinis. System Level Dependability Analysis. In M. Violante M. Sonza Reorda, Z. Peng, editor, *System-level Test and Validation of Hardware/Software Systems*, volume 17 of *Springer Series in Advanced Microelectronics*. Springer, 2005.

[10] A. Bobbio, D. Codetta-Raiteri, Massimiliano De Pierro, and G. Franceschinis. Efficient Analysis Algorithms for Parametric Fault Trees. In *Proceedings of the Workshop on Techniques, Methodologies and Tools for Performance Evaluation of Complex Systems*, pages 91–105, Turin, Italy, September 2005.

[11] A. Bobbio, G. Franceschinis, R. Gaeta, and G. Portinale. Parametric fault tree for the dependability analysis of redundant systems and its high-level Petri net semantics. *IEEE Transactions on Software Engineering*, 29(3):270–287, March 2003.

[12] A. Bobbio, G. Franceschinis, L. Portinale, and R. Gaeta. Exploiting Petri Nets to Support Fault-Tree Based Dependability Analysis. In *8-th International Conference on Petri Nets and Performance Models - PNPM99*, pages 146–155. IEEE Computer Society, 1999.

[13] A. Bobbio, L. Portinale, M. Minichino, and E. Ciancamerla. Improving the Analysis of Dependable Systems by Mapping Fault Trees into Bayesian Networks. *Reliability Engineering and System Safety*, 71:249–260, 2001.

[14] M. Bouissou and J. L. Bon. A new formalism that combines advantages of fault-trees and markov models: Boolean logic driven markov processes. *Reliability Engineering and System Safety*, 82(2):149–163, November 2003.

[15] M. Bouissou, F. Bruyere, and A. Rauzy. BDD based Fault-Tree Processing: A Comparison of Variable Ordering Heuristics. In *Proceedings of European Safety and Reliability Association Conference*, volume 3, pages 2045–2052, 1997.

[16] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient implementation of a BDD package. In *Proceedings 27-th ACM/IEEE Design Automation Conference*, pages 40–45, 1990.

[17] R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35:677–691, 1986.

[18] R. E. Bryant. Symbolic Boolean manipulation with Ordered Binary Decision Diagrams. *ACM Computing Surveys*, 24:293–318, 1992.

[19] K. Buchacker. Analyzing Safety-critical Systems Using Extended Fault Trees. In W. Erhard, K. E. Grosspietsch, W. Koch, E. Maehle, and E. Zehendner, editors, *ARCS 99: Architektur von Rechensystemen 1999 (Workshops)*, pages 117–125, Universitat Jena, Institut fur Informatik, 1999.

[20] K. Buchacker. Combining fault trees and Petri nets to model safety-critical systems. *High Performance Computing*, 1999.

[21] J. A. Carrasco and V. Suñé. An algorithm to find minimal cuts of coherent fault-trees with event-classes, using a decision tree. *IEEE Trans. on Reliability*, 48:31–41, 1999.

[22] G. Chiola. *GreatSPN* 1.5 Software architecture. In G. Balbo and G. Serazzi, editors, *Computer Performance Evaluation*, pages 121–136. Elsevier Science Publishers, 1992.

[23] G. Chiola, G. Bruno, and T. Demaria. Introducing a Color Formalism into Generalized Stochastic Petri Nets. In *Proceedings 9-th European Workshop on Application and Theory of Petri Nets*, Venezia, 1988.

[24] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic Well-Formed Coloured Nets and Multiprocessor Modelling Applications. In K. Jensen and G. Rozenberg, editors, *High-Level Petri Nets. Theory and Application*. Springer Verlag, 1991.

[25] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic Well-Formed Colored Nets and Symmetric Modeling Applications. *IEEE Transactions on Computers*, 42:1343–1360, 1993.

[26] G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo. GreatSPN 1.7: Graphical Editor and Analyzer for Timed and Stochastic Petri Nets. *Performance Evaluation, special issue on Performance Modeling Tools*, 24(1&2):47–68, November 1995.

[27] G. Clark, T. Courtney, D. Daly, D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. Webster. The Möbius Modeling Tool. In *Proceedings of the 9th International Workshop on Petri Nets and Performance Models*, pages 241–250, Aachen, Germany, September 2001.

[28] D. Codetta-Raiteri. BDD based analysis of Parametric Fault Tress. In *Proceedings of the Annual Reliability and Maintainability Symposium*, pages 442–449, Newport Beach, CA USA, January 2006.

[29] D. Codetta-Raiteri. Parametric Dynamic Fault Tree and its Solution through Modularization. In *Supplemental Volume of the International Conference on Dependable Systems and Networks*, pages 157–159, Florence, Italy, June 2004.

[30] D. Codetta-Raiteri. The Conversion of Dynamic Fault Trees to Stochastic Petri Nets, as a case of Graph Transformation. *Electronic Notes on Theoretical Computer Science*, 127(2):45–60, March 2005.

[31] D. Codetta-Raiteri. Development of a Dynamic Fault Tree solver based on Colored Petri Nets and graphically interfaced with DrawNET. Technical Report TR-INF-2003-10-06-UNIPMN, Dipartimento di Informatica, Università del Piemonte Orientale, October 2003.

[32] D. Codetta-Raiteri, G. Franceschinis, M. Iacono, and V. Vittorini. Repairable Fault Tree for the automatic evaluation of repair policies. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks*, pages 659–668, Florence, Italy, June 2004.

[33] S. Contini and A. Poucet. Advances on fault tree and event tree techniques. In A.G. Colombo and A. Saiz de Bustamante, editors, *System Reliability Assessment*, pages 77–102. Kluwer Academic P.G., 1990.

[34] T. Courtney, D. Daly, S. Derisavi, S. Gaonkar, M. Griffith, V. Lam, and W. H. Sanders. The Möbius Modeling Environment: Recent Developments. In *Proceedings of the 1st International Conference on Quantitative Evaluation of Systems (QEST)*, Twente, The Netherlands, September 2004.

[35] T. Courtney, D. Daly, S. Derisavi, V. Lam, and W. H. Sanders. The Möbius Modeling Environment. In *Tools of the 2003 Illinois International Multiconference on Measurement, Modelling, and Evaluation of Computer-Communication Systems*, volume research report no. 781/2003, pages 34–37, Universitat Dortmund Fachbereich Informatik, 2003.

[36] K. Czarnecki and S. Helsen. Classification of Model Transformation Approaches. In *Proceedings of the Workshop on Generative Techniques in the Context of Model-Driven Architectures*, 2003.

[37] D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J. Doyle, W. Sanders, and P. G. Webster. The Möbius Framework and its Implementation. *IEEE Transactions on Software Engineering*, 28(10):956–969, 2002.

[38] S. A. Doyle and J. B. Dugan. Dependability Assessment Using Binary Decision Diagrams (BDDs). In *FTCS '95: Proceedings of the Twenty-Fifth International Symposium on Fault-Tolerant Computing*, page 249, Washington, DC, USA, 1995. IEEE Computer Society.

[39] J. B. Dugan and T. S. Assaf. Dynamic Fault Tree Analysis of a Reconfigurable Software System. In *The 19th International System Safety Conference*, Huntsville, Alabama, September 2001.

[40] J. B. Dugan, S. J. Bavuso, and M. A. Boyd. Dynamic Fault-Tree Models for Fault-Tolerant Computer Systems. *IEEE Transactions on Reliability*, 41:363–377, 1992.

[41] J. B. Dugan, S. J. Bavuso, and M. A. Boyd. Fault-trees and Markov models for reliability analysis of fault-tolerant digital systems. *Reliability Engineering and System Safety*, 39:291–307, 1993.

[42] J. B. Dugan, K. J. Sullivan, and D. Coppit. Developing a Low-Cost High-Quality Software Tool for Dynamic Fault-Tree Analysis. *IEEE Transactions on Reliability*, 49:49–59, 2000.

[43] Y. Dutuit, O. Lemaire, and A. Rauzy. New Insight on Measures of Importance of Components and Systems in Fault Tree Analysis. In S. Kondo and K. Furuta, editors, *Proceedings of the International Conference on Probabilistic Safety Assessment and Management, PSAM'5*, pages 729–734. Universal Academy Press, 2000.

[44] Y. Dutuit and A. Rauzy. A Linear-Time Algorithm to Find Modules of Fault Trees. *IEEE Transactions on Reliability*, 45:422–425, 1996.

[45] Y. Dutuit and A. Rauzy. Exact and Truncated Computations of Prime Implicants of Coherent and non-Coherent Fault Trees within Aralia. *Reliability Engineering and System Safety*, 58:127–144, 1997.

[46] Y. Dutuit and A. Rauzy. A guided tour of minimal cutsets handling by means of binary decision diagrams. In *Proceedings of Probabilistic Safety Assessment conference, PSA'99*, volume 2, pages 55–62. American Nuclear Society, 1999.

[47] Y. Dutuit and A. Rauzy. New algorithms to compute importance factors CPr, MIF, CIF, DIF, RAW and RRW. In *Proceedings of the European Safety and Reliability Association Conference, ESREL99*, volume 2, pages 1015–1020, 1999.

[48] Y. Dutuit and A. Rauzy. Efficient Algorithms to Assess Components and Gates Importances in Fault Tree Analysis. *Reliability Engineering and System Safety*, 72(2):213–222, 2000.

[49] H. Ehrig and J. Padberg. Graph Grammars and Petri Net Transformation. In G. Rozenberg J. Desel, W. Reisig, editor, *Lectures on Concurrency and Petri Nets*, number 3098 in LNCS, pages 496–536. Springer, 2004.

[50] G. Engels and R. Heckel. Graph Transformation and Visual Modeling Techniques. *Bulletin of the European Association for Theoretical Computer Science, EATCS*, 71, June 2000.

[51] G. Franceschinis, R. Gaeta, and G. Portinale. Dependability Assessment of an Industrial Programmable Logic Controller via Parametric Fault-Tree and High Level Petri Net. In *Proc. 9th Int. Workshop on Petri Nets and Performance Models*, pages 29–38, Aachen, Germany, Sept. 2001.

[52] G. Franceschinis, M. Gribaudo, M. Iacono, N. Mazzocca, and V. Vittorini. Towards an Object Based Multi-Formalism Multi-Solution Modeling Approach. In *Proc. of the Second International Workshop on Modelling of Objects, Components, and Agents (MOCA'02)*, pages 47–66, Aarhus, Denmark, August 2002.

[53] G. Franceschinis, M. Gribaudo, M. Iacono, V. Vittorini, and C. Bertoncello. DrawNet++: a flexible framework for building dependability models. In *In Proc. of the Int. Conf. on Dependable Systems and Networks*, Washington DC, USA, June 2002.

[54] M. Fujita, H. Fujisawa, and Y. Matsugana. Variable ordering algorithm for ordered binary decision diagrams and their evaluation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(1):6–12, January 1993.

[55] K. B. Fussel. How to hand-calculate system reliability characteristics. *IEEE Transactions on Reliability*, R-24(3), 169-174.

[56] R. Geist and K. Trivedi. Reliability estimation of fault-tolerant systems: tools and techniques. *IEEE Computer*, pages 52–61, July 1990.

[57] A. Goyal, V. F. Nicola, A. N. Tantawi, and K. S. Trivedi. Reliability of systems with limited repair. *IEEE Transactions on Reliability*, R-36:202–207, 1987.

[58] M. Gribaudo, D. Codetta-Raiteri, and G. Franceschinis. Draw-Net, a customizable multi-formalism multi-solution tool for the quantitative evaluation of systems. In *Proceedings of the 2nd International Conference on Quantitative Evaluation of Systems*, pages 257–258, Turin, Italy, September 2005.

[59] M. Gribaudo, M. Iacono, N. Mazzocca, and V. Vittorini. The OsMoSys/DrawNET Xe! Languages System: A Novel Infrastructure for Multi-Formalism Object-Oriented Modelling. In *In Proc. 15th European Simulation Symposium and Exhibition*, Delft, The Netherlands, October 2003.

[60] M. Gribaudo, N. Mazzocca, F. Moscato, and V. Vittorini. Multisolution of Complex Performability Models in the OsMoSys/DrawNet Framework. In *Proc. 2nd Int. Conf. on the Quantitative Evaluation of Systems*, Torino, Italy, Sept. 2005.

[61] R. Gulati and J. B. Dugan. A modular approach for analyzing static and dynamic fault-trees. In *Proceedings IEEE Annual Reliability and Maintainability Symposium*, pages 57–63, 2003.

[62] A. Gupta and A. L. Fisher. Representation and symbolic manipulation of linearly inductive Boolean functions. In *ICCAD '93: Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design*, pages 192–199, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press.

[63] A. Hoyland and M. Rausand. *System Reliability Theory*. John Wiley & Son, 1994.

[64] G. S. Hura and J. W. Atwood. The use of Petri nets to analyze coherent fault-trees. *IEEE Transactions on Reliability*, 37:469–474, 1988.

[65] K. Jensen. Coloured Petri Nets: A High Level Language for System Design and Analysis. *Lecture Notes in Computer Science; Advances in Petri Nets 1990*, 483:342–416, 1991.

[66] Y. Kai. Multistate fault-tree analysis. *Reliability Engineering and System Safety*, 28:1–7, 1990.

[67] I. N. Kovalenko, N. Y. Kuznetsov, and P. A. Pegg. Mathematical theory of Reliability of Time Dependent Systems with Practical Applications. *Wiley Series in Probability and Statistic*, 1997.

[68] J. M. Küster, S. Sendall, and M. Wahler. Comparing two model transformation approaches. In *OCL and Model Driven Engineering, UML 2004 Conference Workshop*, pages 114–127, Lisbon, Portugal, 2004.

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

[70] R. Manian, D. W. Coppit, K. J. Sullivan, and J. B. Dugan. Bridging the Gap Between Systems and Dynamic Fault Tree Models. In *Proceedings Annual Reliability and Maintainability Symposium*, pages 105–111, 1999.

[71] R. Manian, J. B. Dugan, D. Coppit, and K. Sullivan. Combining various solution techniques for dynamic fault tree analysis of computer systems. In *Proceedings Third IEEE International High-Assurance Systems Engineering Symposium*, pages 21–28, Washington, D.C., November 1998.

[72] S. Minato, N. Ishiura, and S. Yajima. Shared binary decision diagrams with attributed edges for efficient boolean function manipulation. In L. J. M. Claesen, editor, *Proceedings of the 27th ACM/IEEE Design Automation Conference (DAC'90)*, pages 52–57, June 1990.

[73] S. Montani, L. Portinale, and A. Bobbio. Dynamic Bayesian Networks for Modeling Advanced Fault Tree Features in Dependability Analysis. In *Advances in Safety and Reliability (ESREL 2005)*, volume 2, pages 1415–1422. Balkema, 2005.

[74] S. Montani, L. Portinale, A. Bobbio, and D. Codetta-Raiteri M. Varesio. A tool for automatically translating Dynamic Fault Trees into Dynamic Bayesian Networks. In *Proceedings of the Annual Reliability and Maintainability Symposium*, pages 434–441, Newport Beach, CA USA, January 2006.

[75] B. Natvig. New light on measures of importance of system components. *Scandinavian Journal of Statistic*, 12:43–52, 1985.

[76] Y. Ou and J. B. Dugan. Sensitivity analysis of modular dynamic fault trees. In *IEEE International Computer Performance and Dependability Symposium*, March 2000.

[77] L. Portinale and A. Bobbio. Bayesian networks for dependability analysis: an application to digital control reliability. In *Proc. 15th Conference on Uncertainty in Artificial Intelligence (UAI 99)*, pages 551–558, July 1999.

[78] A. Rauzy. New Algorithms for Fault Trees Analysis. *Reliability Engineering & System Safety*, 05(59):203–211, 1993.

[79] A. Rauzy. A brief introduction to binary decision diagrams. *Journal Européen des Systèmes Automatisés (RAIRO-APII-JESA)*, 30(8):1033–1051, 1996.

[80] A. Rauzy. Mathematical Foundation of Minimal Cutsets. *IEEE Transactions on Reliability*, 50(4):389–396, December 2001.

[81] A. Reibman. Modeling the effect of reliability on performance. *IEEE Transactions on Reliability*, R-39:314–320, 1990.

[82] R. Sahner and K. S. Trivedi. Reliability modeling using SHARPE. *IEEE Transactions on Reliability*, R-36:186–193, 1987.

[83] R. A. 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.

[84] W. H. Sanders. Integrated frameworks for multi-level and multi-formalism modeling. In *8-th International Conference on Petri Nets and Performance Models - PNPM99*, pages 2–9. IEEE Computer Society, 1999.

[85] W. H. Sanders and J. F. Meyer. Reduced based model construction methods for Stochastic Activity Networks. *IEEE Journal on Selected Areas in Communication, special issue on Computer-Aided Modeling, Analysis and Design of Communication Networks*, 9(1):25–36, 1991.

[86] W. H. Sanders and J. F. Meyer. Stochastic Activity Networks: Formal Definitions and Concepts. *Lectures on Formal Methods and Performance Analysis, Lecture Notes in Computer Science*, 2090:315–343, 2001.

[87] W. G. Schneeweiss. Fault-tree analysis using a binary decision tree. *IEEE Transactions on Reliability*, R-34:452–457, 1985.

[88] W. G. Schneeweiss. Approximate fault-tree analysis with prescribed accuracy. *IEEE Transactions on Reliability*, R-36:250–254, 1987.

[89] W. G. Schneeweiss. *Petri Nets for Reliability Modeling*. LiLoLe Verlag, 1999.

[90] W. G. Schneeweiss. *The Fault Tree Method*. LiLoLe Verlag, 1999.

[91] D. P. Siewiorek. Fault-tolerance in commercial computers. *IEEE Computer*, pages 26–37, July 1990.

[92] D. P. Siewiorek and R. S. Swarz. *Reliable Computer Systems: Design and Evaluation*. Digital Press, 1992.

[93] C. Singh and R. Billinton. *System Reliability Modelling and Evaluation*. Hutchinson, London, 1977.

[94] R. M. Sinnamon and J. D. Andrews. Quantitative fault tree analysis using binary decision diagrams. *Journal Europeen des Systemes Automatises*, 1996.

[95] R. M. Sinnamon and J. D. Andrews. Improved accuracy in quantitative fault tree analysis. *Quality and Reliability Engineering International*, 13:285–292, 1997.

[96] R. M. Sinnamon and J. D. Andrews. Improved efficiency in qualitative fault tree analysis. *Quality and Reliability Engineering International*, 13:293–298, 1997.

[97] K. J. Sullivan, J. B. Dugan, and D. Coppit. The Galileo fault tree analysis tool. In *Proceedings of the 29th Annual International Symposium on Fault-Tolerant Computing*, pages 232–235, Madison, Wisconsin, June 1999.

[98] Z. Tang and J. B. Dugan. Minimal Cut Set/Sequence Generation for Dynamic Fault Trees. In *Procedings of the Annual Reliability and Maintainability Symposium*, pages 207–213, Los Angeles, CA USA, January 2004.

[99] L. Tomek, V. Mainkar, R. M. Geist, and K. S. Trivedi. Reliability modeling of life-critical real-time systems. *Proceedings of the IEEE*, 82:108–121, 1994.

[100] K. Trivedi. *Probability & Statistics with Reliability, Queueing & Computer Science applications*. Wiley, II Edition, 2001.

[101] K. S. Trivedi. Modeling and analysis of fault-tolerant systems. In *Proceedings International Conference on Modelling Techniques and Tools for Performance Analysis*, Paris, 1984.

[102] K. S. Trivedi. Reliability evaluation of fault-tolerant systems. In P.J. Courtois G. Iazeolla and A. Hordijk, editors, *Mathematical Computer Performance and Reliability*, pages 403–424. North Holland, 1984.

[103] K. S. Trivedi, J. K. Muppala, S. P. Woolet, and B. R. Haverkort. Composite performance and dependability analysis. *Performance Evaluation*, 14:197–215, 1992.

[104] V. E. Vesley. A time dependent methodology for fault tree evaluation. *Nuclear Engineering and Design*, 13:337–360, 1970.

[105] N. Viswanadham, V. V. S. Sarma, and M. G. Singh. *Reliability of computers and control systems*. North-Holland, Amsterdam, 1987.

[106] V. Vittorini, G. Franceschinis, M. Gribaudo, M. Iacono, and N. Mazzocca. DrawNet++: Model objects to support performance analysis and simulation of complex systems. In *Proceedings 12th International Conference on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation (TOOLS 2002)*, pages 233–238, London, 2002. Springer Verlag - LNCS, Vol 2324.

[107] V.Vittorini, M. Iacono, N. Mazzocca, and G. Franceschinis. The OsMoSys approach to multi-formalism modeling of systems. *Journal of Software and System Modeling*, 3(1), March 2004.

[108] A. P. Wood. Multistate block diagrams and fault trees. *IEEE Transactions on Reliability*, R-34:236–240, 1985.

[109] http://www.draw-net.com. Draw-Net web page.

[110] H. Xu and J. B. Dugan. Combining Dynamic Fault Trees and Event Trees for Probabilistic Risk Assessment. In *Annual Reliability and Maintainability Symposium 2004 Proceedings*, pages 214–219, Los Angeles, CA USA, January 2004.

[111] X. Zang, H. Sun, and K. Trivedi. A BDD-based algorithm for reliability analysis of phased-mission systems. *IEEE Transactions on Reliability*, 48:50–60, 1999.

Appendix A

Model transformation from DRPFT to SWN

This appendix provides a model transformation system (section 4.5.3) to map DRPFT models to the equivalent SWNs. The source model is a DRPFT, while the target model is a SWN. The compound rules in this model transformation system concern RBs and the several types of event and gate. Such rules are in the form described in section 4.5.3. The model transformation system proposed in this section, requires the definition of two new functions in the DRPFT formalism:

- $conv : \mathcal{E} \rightarrow \mathbb{B}$ is the function returning the $true$ value if an event in the source DRPFT model, has already been mapped in the SWN target model, and returning the $false$ value if an event in the source model has not yet been mapped in the target model.

- $lab : \mathcal{E} \rightarrow \{A, \ldots, Z\}^+$ is the function returning the label assigned to an event, where $\{A, \ldots, Z\}^+$ is the set of all the possible non empty strings we can compose with the alphabet $\{A, \ldots, Z\}$.

The lab function has been defined also in the SWN formalism, together with two new functions concerning colour classes:

- $lab : P \cup T \rightarrow \{A, \ldots, Z\}^+$ is the function returning the label assigned to a place or transition.

- Given the ordered colour class $c \in C$,
  $first(c)$ returns the first element of c;
  $last(c)$ returns the last element of c.

In the compound rules in our model transformation system, labels are used to identify the nodes inside the source model and the target model.

Some of the compound rules dealing with RBs, need the specification of a new attribute for the RBs: before the begin of the model transformation process, for

each RB b in the DRPFT, we set the attribute $RepEv(b)$ to the set of labels of the events belonging to $Cov_E(b) - Cov_{BE}(b)$.

We can classify DRPFTs and SWNs as labelled attributed oriented graphs (section 4.5.1); labels are returned by the function lab, while attributes are returned by other functions defined in the DRPFT formalism (section 6.2) and in the SWN formalism (section 6.4).

Some compound rules may have an higher priority with respect to other ones.

There is a general correspondence between DRPFT elements and the elements of the equivalent SWN obtained through our model transformation system:

- generic event $\Leftrightarrow$ place
- BE, IE or TE $\Leftrightarrow$ place $p : cd(p) = \emptyset$
- RE or BRE $\Leftrightarrow$ place $p : cd(p) \neq \emptyset$
- type $\Leftrightarrow$ colour class
- Cartesian product of types $\Leftrightarrow$ colour domain
- not occurred event $\Leftrightarrow$ empty place
- occurred event $\Leftrightarrow$ marked place
- BE occurrence $\Leftrightarrow$ timed transition firing
- gate $\Leftrightarrow$ set of immediate transitions

The description of all the compound rules used in our model transformation system, follows. In the target graph (SWN) transformation rules, whenever it is not differently indicated, we have that

- the marking of a place is equal to $\emptyset$;
- the colour domain of a place equal to $\emptyset$;
- the priority of an immediate transition is equal to 1;
- the cardinality of an oriented arc touching a place $p : cd(p) = \emptyset$, is 1;
- an oriented arc has no arc expression;
- the cardinality of an inhibitor arc touching a place $p : cd(p) = \emptyset$, is 1;
- an inhibitor arc has no arc expression.

Moreover, we assume that all the colour classes are ordered. The colour domain of a place is indicated as the set of colour classes composing it; actually the colour domain is given by the Cartesian product of the colour classes composing it (section 6.3). Similarly, we indicate the initial marking of a place with non empty colour domain, as the set of colour classes whose Cartesian product provides the colours of the tokens initially contained in the place.

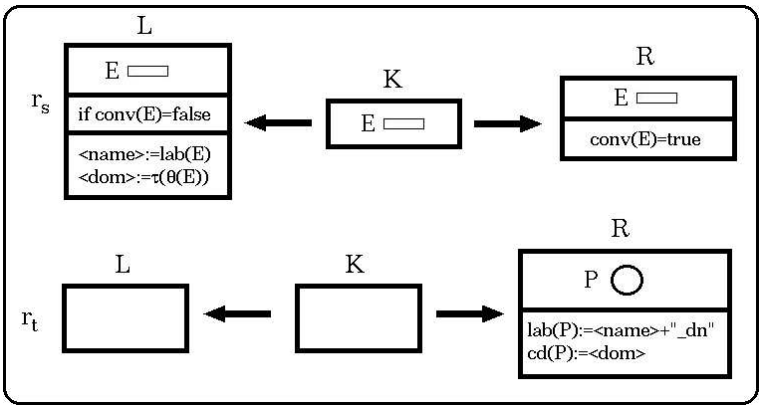

Figure A.1: Compound rule for an IE.

Events conversion

Fig. A.1, Fig. A.2 and Fig A.3 show the compound rules to map the non replicator events (BEs, IEs, TE) of the DRPFT source model in the SWN target model. REs and BREs can be mapped in the target model by means of the compound rules in Fig. A.4 and in Fig. A.5, respectively. The BEs and the BREs representing the failure of spare components must be mapped in the SWN by means of the compound rules in Fig. A.6 (BEs) and in Fig. A.7 (BREs). If the rule in Fig. A.6 can be applied to a BE of the source model, also the rule in Fig. A.3 can be applied to the same BE; this conflict is avoided by assigning an higher priority to the rule in Fig. A.6 with respect to the rule in Fig. A.3. After the application of the rule in Fig. A.6 to a BE, the rule in Fig. A.3 can not be applied to the same BE. In a similar way, given a BRE representing the failure of a set of spare component, both the rule in Fig. A.7 and the rule in Fig. A.5 can be applied to the BRE. For this reason, the rule in Fig. A.7 has an higher priority with respect to the rule in Fig. A.5.

The rules dealing with gates require the previous conversion of the events connected to the gate.

Boolean gates conversion

AND gate conversion

Given a gate of type AND whose output event is an IE, the compound rule in Fig. A.8 must be applied to map the gate in the SWN, together with the application of the rule in Fig. A.9 to every non replicator input event of the gate, and with the application of the rule in Fig. A.10 to every replicator input event of the gate.

We omit the compound rules for the conversion of a gate of type AND with a

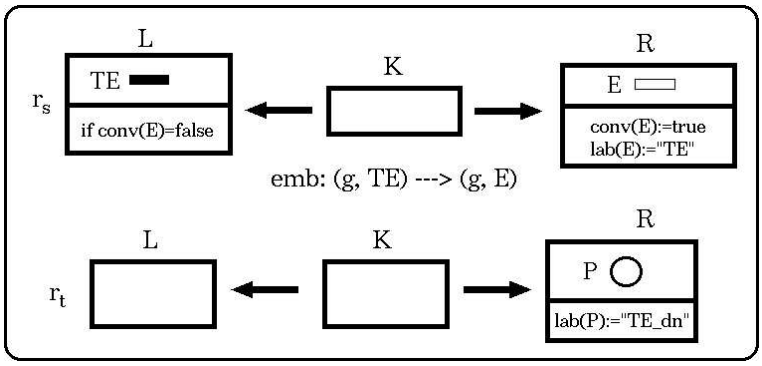

Figure A.2: Compound rule for the TE.

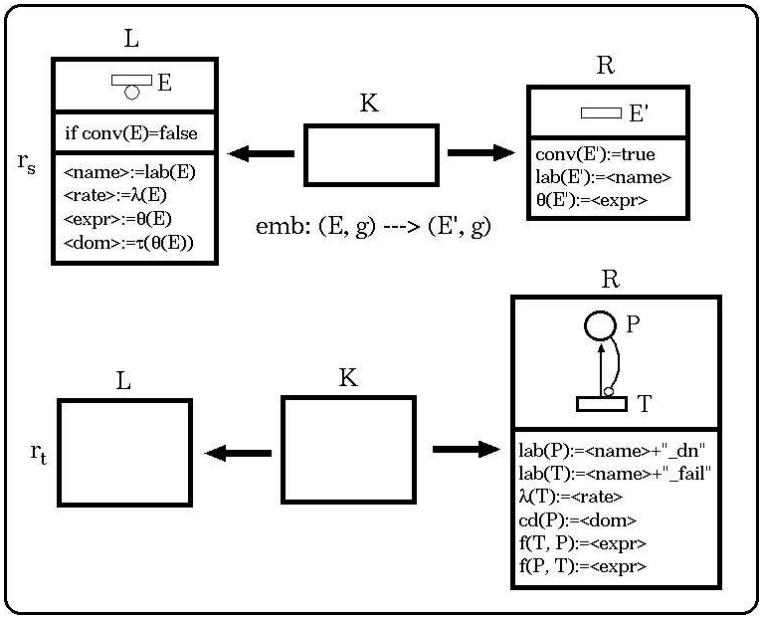

Figure A.3: Compound rule for a BE.

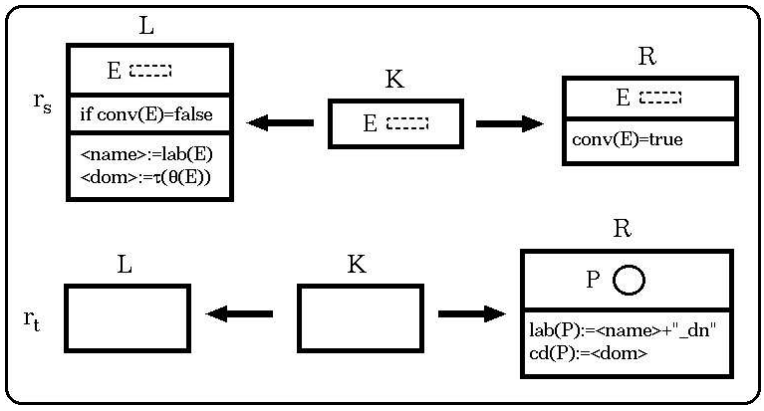

Figure A.4: Compound rule for a RE.

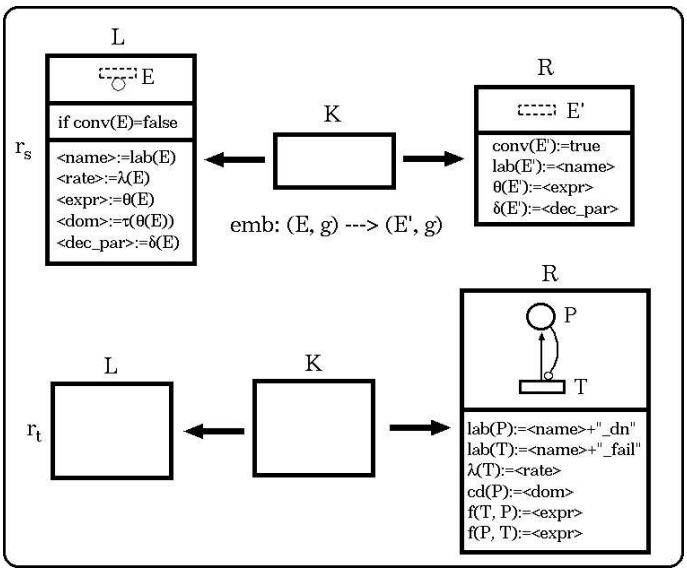

Figure A.5: Compound rule for a BRE.

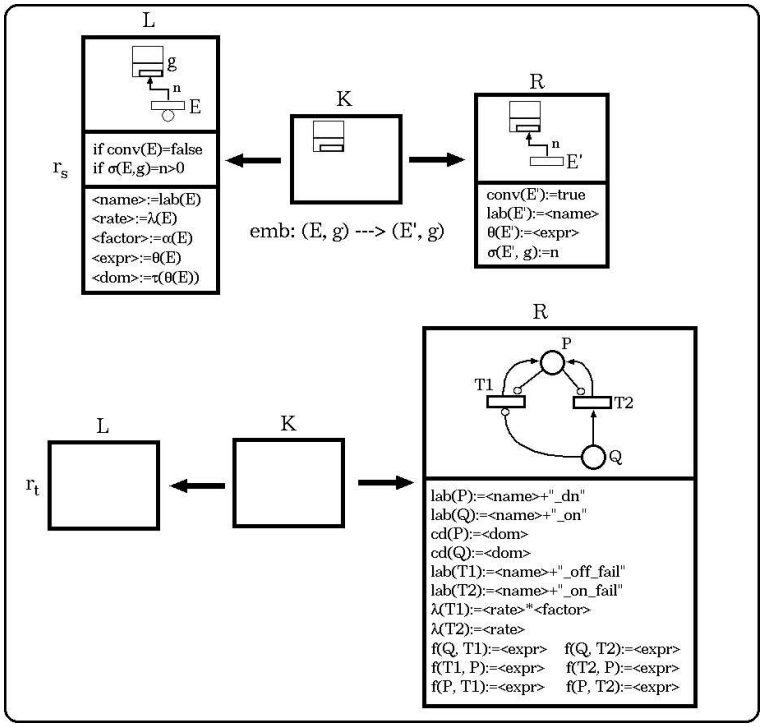

Figure A.6: Compound rule for a non replicator event relative to a spare.

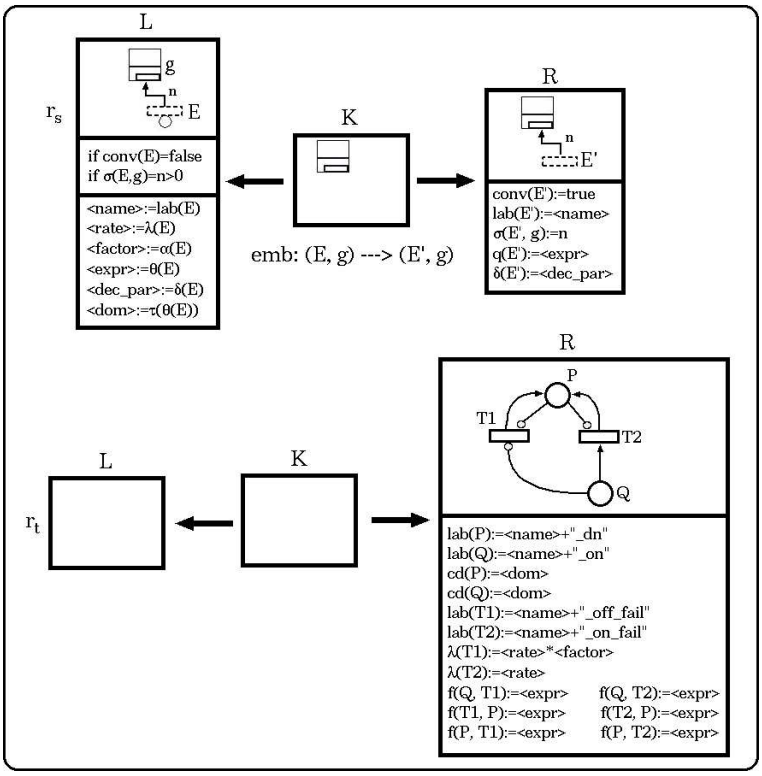

Figure A.7: Compound rule for a replicator event relative to a set of spares.

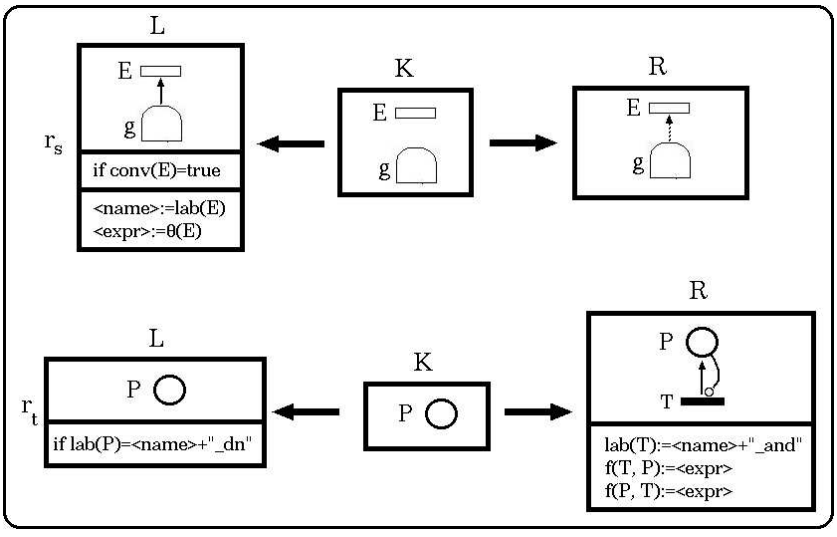

Figure A.8: Compound rule for a gate of type AND with an IE as output event.

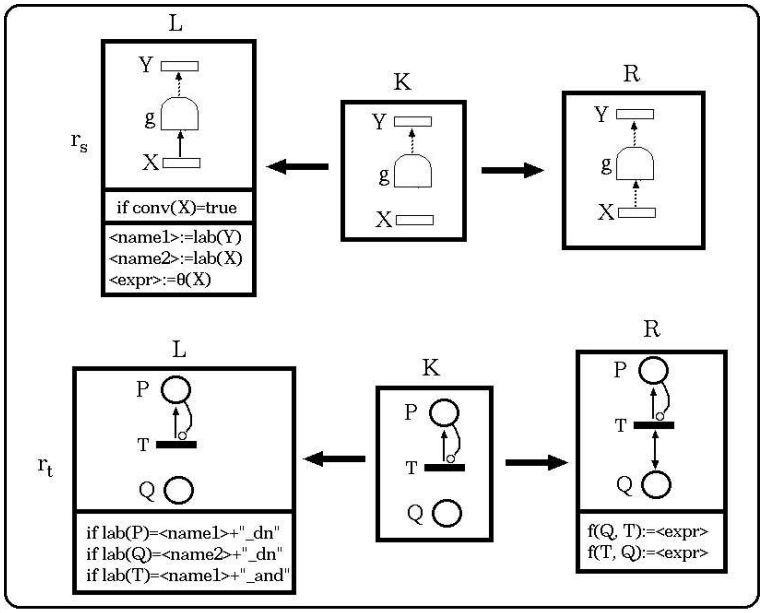

Figure A.9: Compound rule for a gate of type AND with a non replicator event as input event (and an IE as output event).

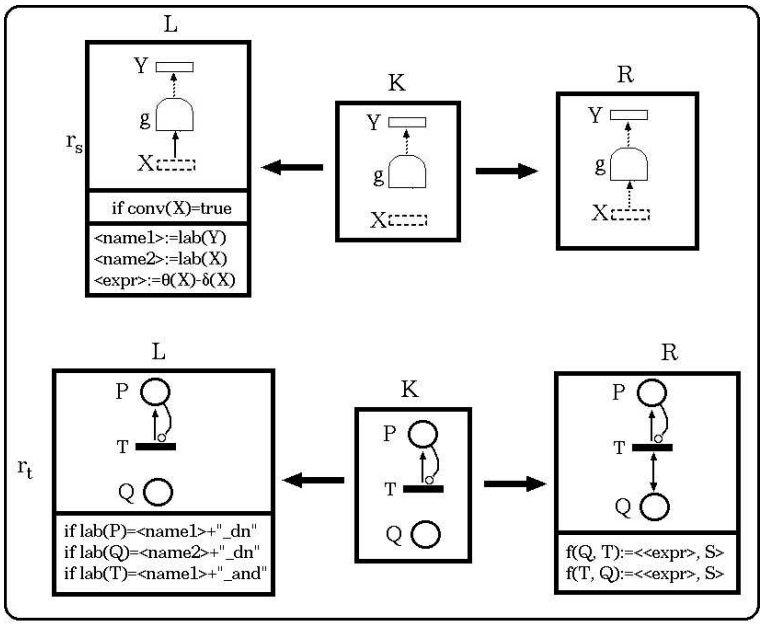

Figure A.10: Compound rule for a gate of type AND with a replicator event as input event (and an IE as output event).

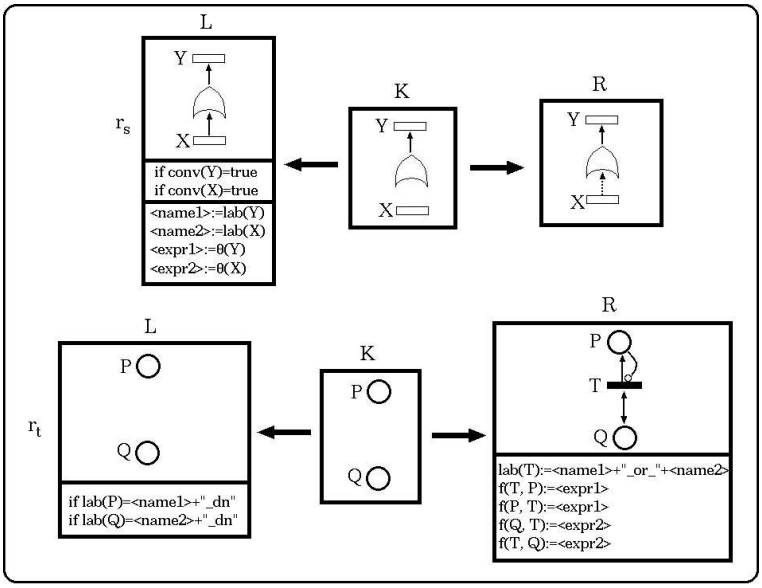

Figure A.11: Compound rule for a gate of type OR with a non replicator event as input event (and an IE as output event).

RE as output event; such rules differ from the rules in Fig. A.8, in Fig. A.9 and in Fig. A.10, only by the presence of a RE as output event of the gate, instead of an IE.

OR gate conversion

Given a gate of type OR having an IE as output event, the gate is converted into SWN by means of the application of the compound rule in Fig. A.11 to every non replicator input event of the gate, and of the compound rule in Fig. A.12 on every replicator input event of the gate.

We omit the compound rules for the conversion of a gate of type OR with a RE as output event; such rules differ from the rules in Fig. A.11 and in Fig. A.12, only by the presence of a RE as output event of the gate, instead of an IE.

Dynamic gates conversion

$FDEP$ gate conversion

The dependence of a non replicator event on some other event, expressed by means of a gate of type $FDEP$, can be mapped in the SWN target model by means of the compound rule in Fig. A.13. If the dependent event is a replicator event, the rule in Fig. A.14 must be used.

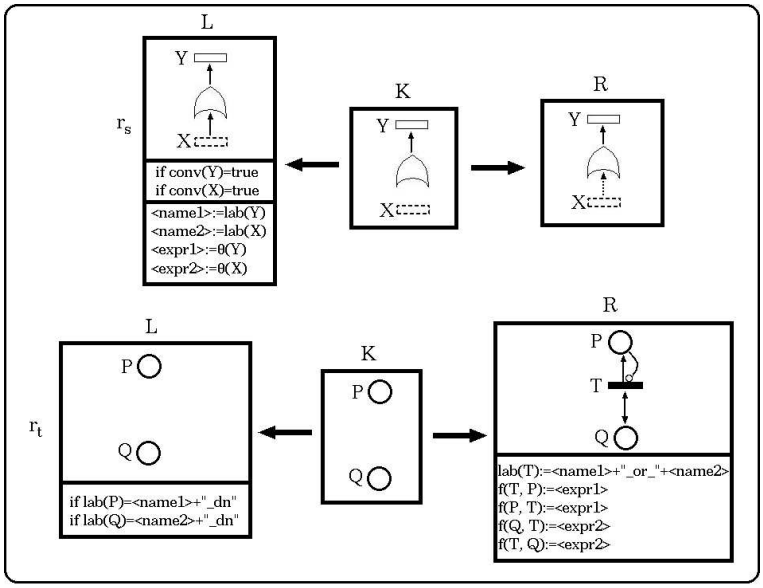

Figure A.12: Compound rule for a gate of type OR with a replicator event as input event (and an IE as output event).

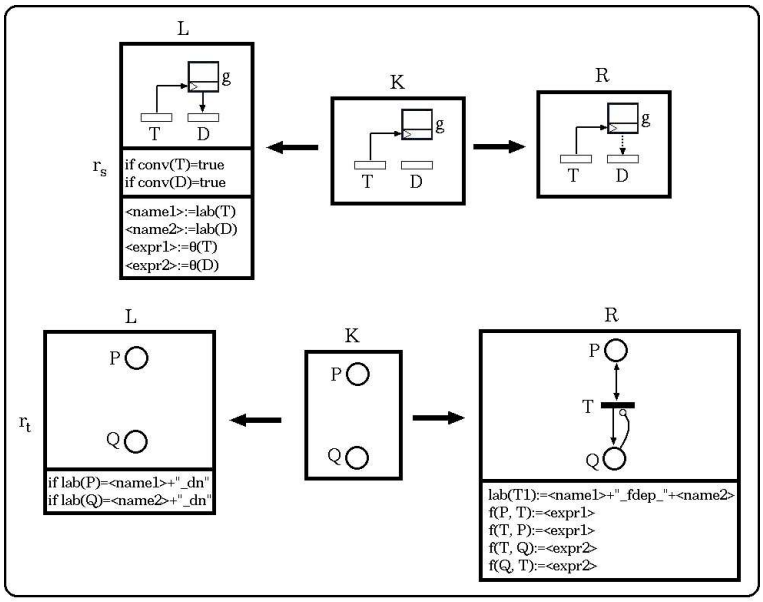

Figure A.13: Compound rule for a gate of type $FDEP$ with a non replicator event as dependent event; the trigger event must be a non replicator event.

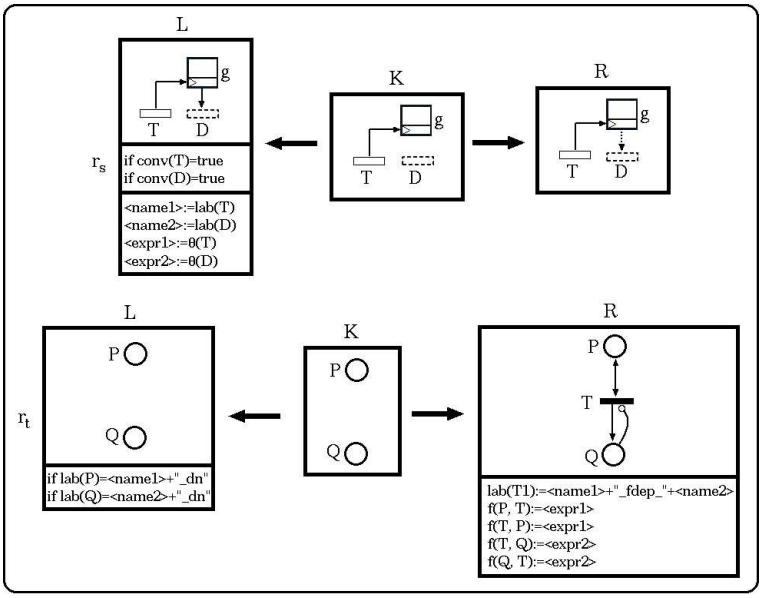

Figure A.14: Compound rule for a gate of type $FDEP$ with a replicator event as dependent event; the trigger event must be a non replicator event.

$PAND$ gate conversion

A gate of type $PAND$ whose output event is an IE and its input events are non replicator events, can be mapped in the SWN by applying the following rules: the compound rule in Fig. A.15 on the output event of the gate, the compound rule in Fig. A.16 on the first and the second of its input events, and the compound rule in Fig. A.17 on the following input events of the gate, if any. If the $PAND$ gate has an IE as output event and a replicator event as unique input event, the compound rule to be applied to the input event, is shown in Fig. A.18.

We omit the compound rules for the conversion of a gate of type $PAND$ with a RE as output event; such rules differ from the rules in Fig. A.15, in Fig. A.16, in Fig. A.17 and in Fig. A.18, only by the presence of a RE as output event of the gate, instead of an IE.

SEQ gate conversion

A gate of type SEQ whose "trigger" and dependent events are all non replicator events, is converted to SWN by means of the application of the compound rule in Fig. A.19 to the trigger event and on the first of the dependent events. If the gate has other dependent events, the rule in Fig. A.20 must be applied to them. If a SEQ gate is connected to a unique replicator event, folding the trigger and the dependent events, the gate must be converted into SWN by means of the rule in

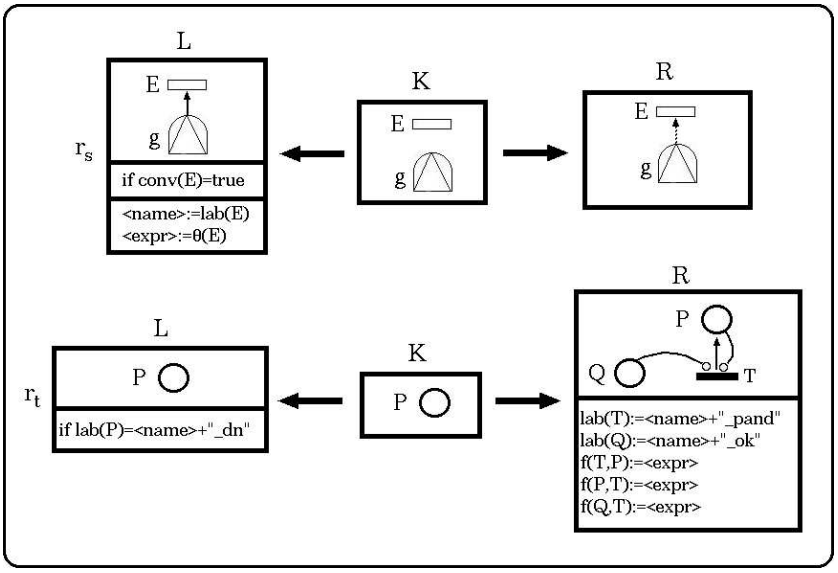

Figure A.15: Compound rule for a gate of type $PAND$ with an IE as output event.

Fig. A.21.

WSP gate conversion

Given a gate of type WSP having an IE as output event, the gate can be mapped into SWN, by applying the following compound rules: the rule in Fig. A.22 to the output event of the gate and to the event relative to the main component, the rule in Fig. A.23 to every non replicator event relative to a spare component, the rule in Fig. A.24 to every replicator event relative to a set of spare components.

We omit the compound rules for the conversion of a gate of type WSP with a RE as output event; such rules differ from the rules in Fig. A.22, in Fig. A.23 and in Fig. A.24, only by the presence of a RE as output event of the gate, instead of an IE.

Repair Boxes conversion

A RB having a non replicator event as trigger event is converted into SWN by applying the following compound rules:

- the rule in Fig. A.25 to trigger event of the RB;
- the rule in Fig. A.26 to every non replicator event in the basic coverage set of the RB;

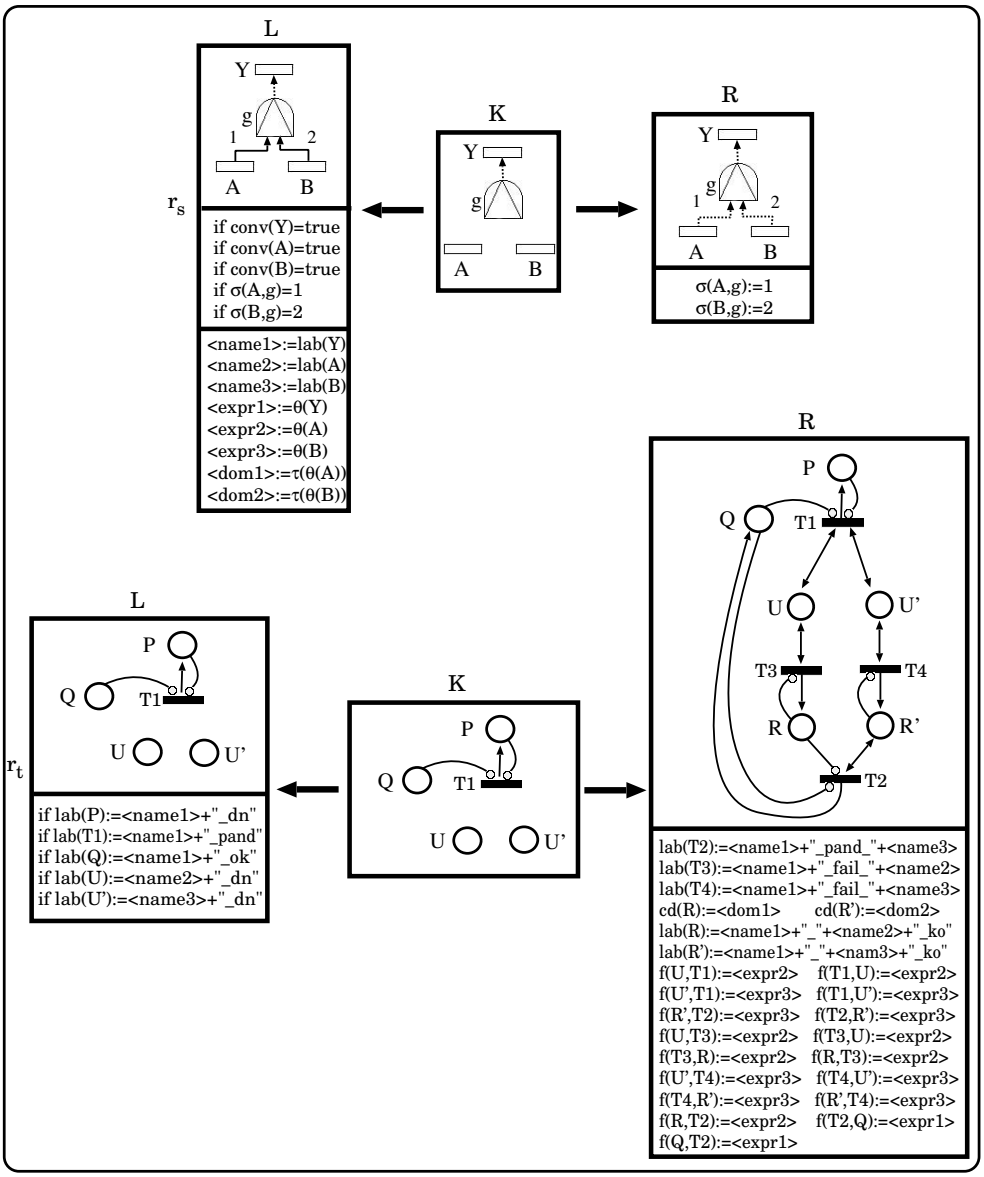

Figure A.16: Compound rule for a gate of type $PAND$ and its input events with order numbers 1 and 2 (with an IE as output event).

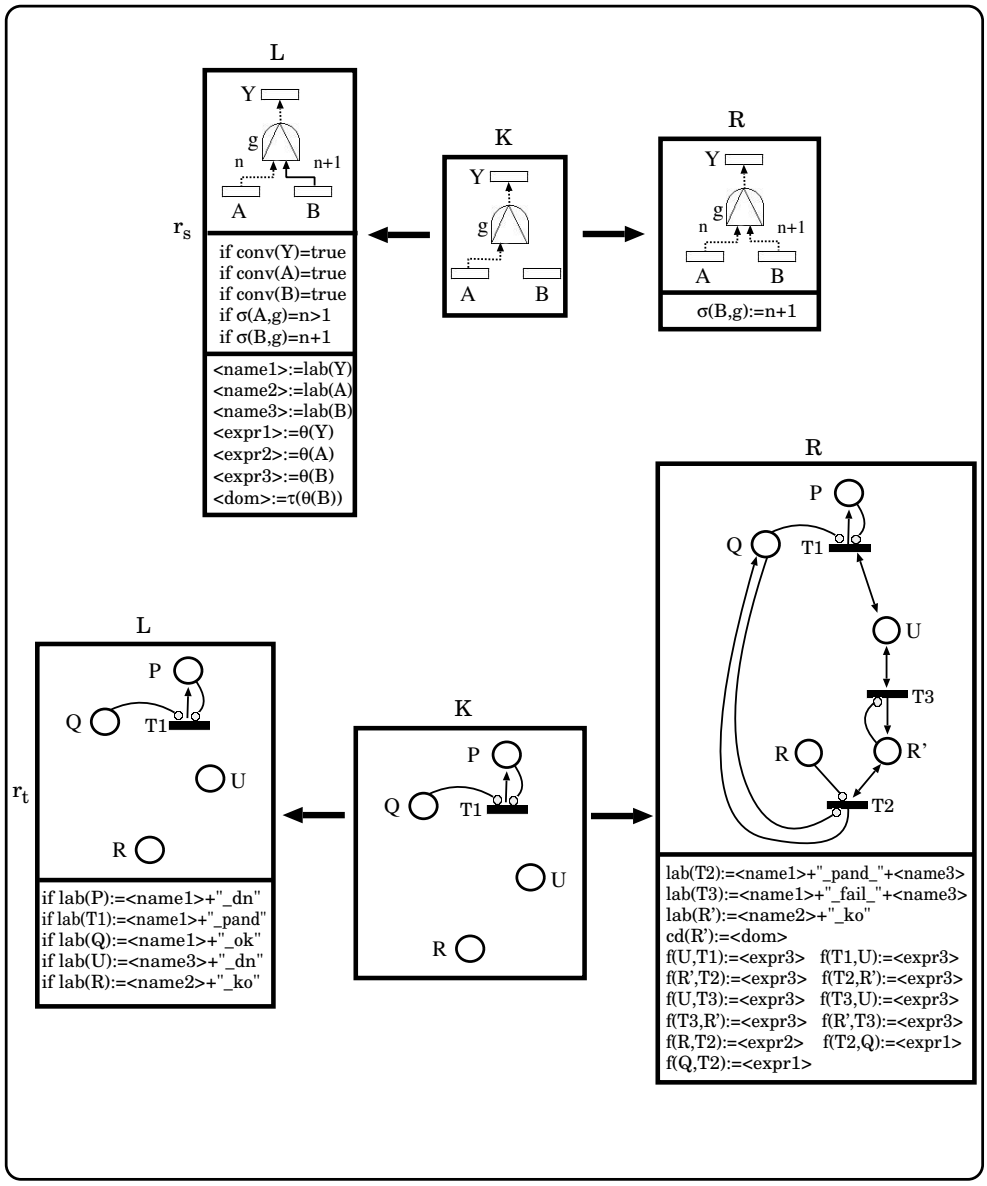

Figure A.17: Compound rule for a gate of type $PAND$ and its input events with order numbers n and $n+1$ (with an IE as output event).

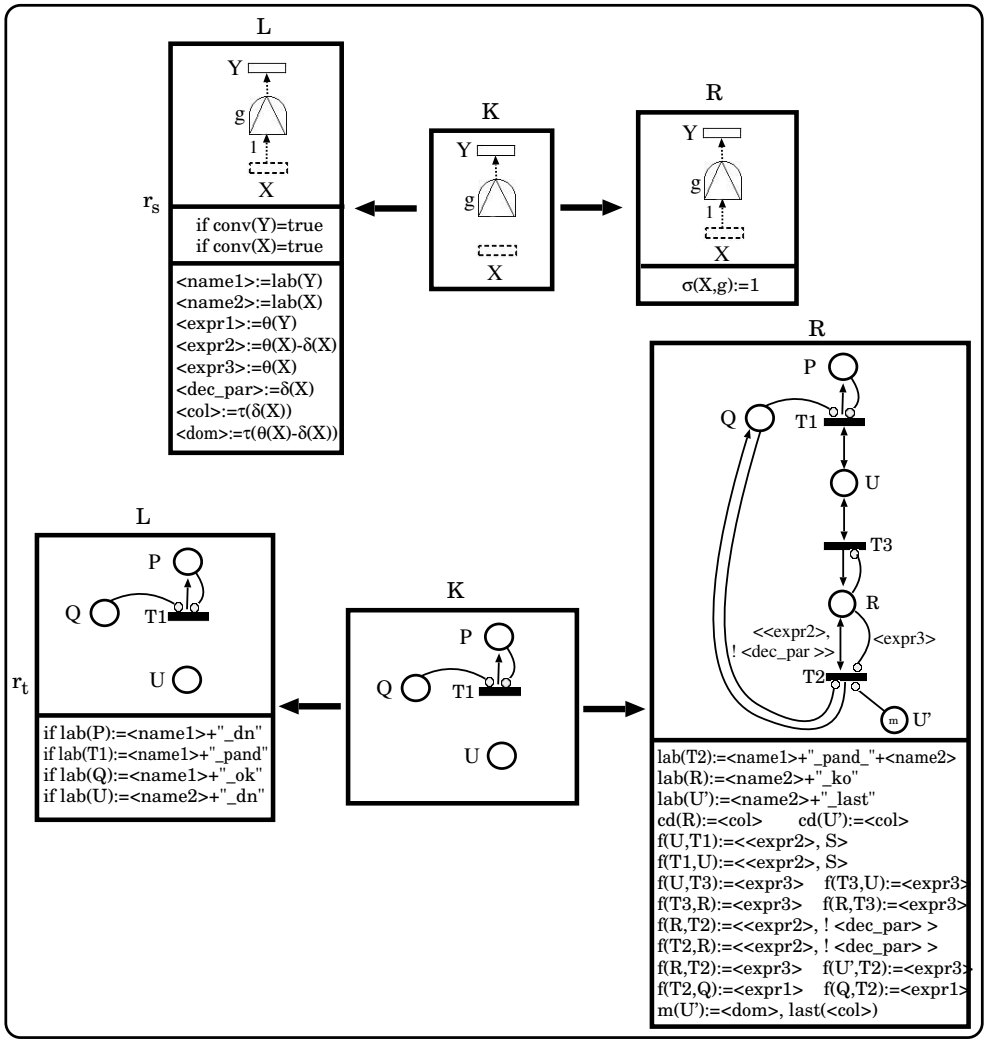

Figure A.18: Compound rule for a gate of type $PAND$ having a replicator event as input event (with an IE as output event).

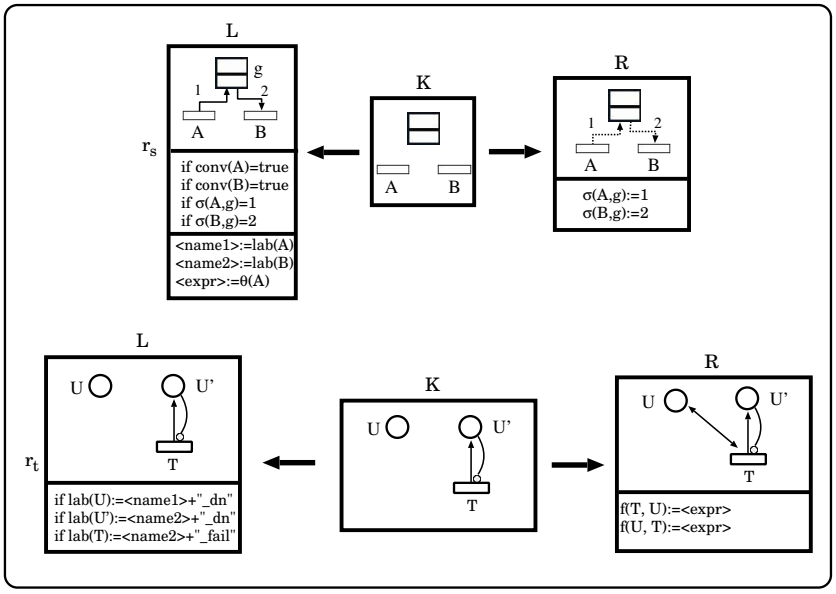

Figure A.19: Compound rule for a gate of type SEQ having a non replicator event as "trigger" event, and a non replicator event as dependent event with order number 2.

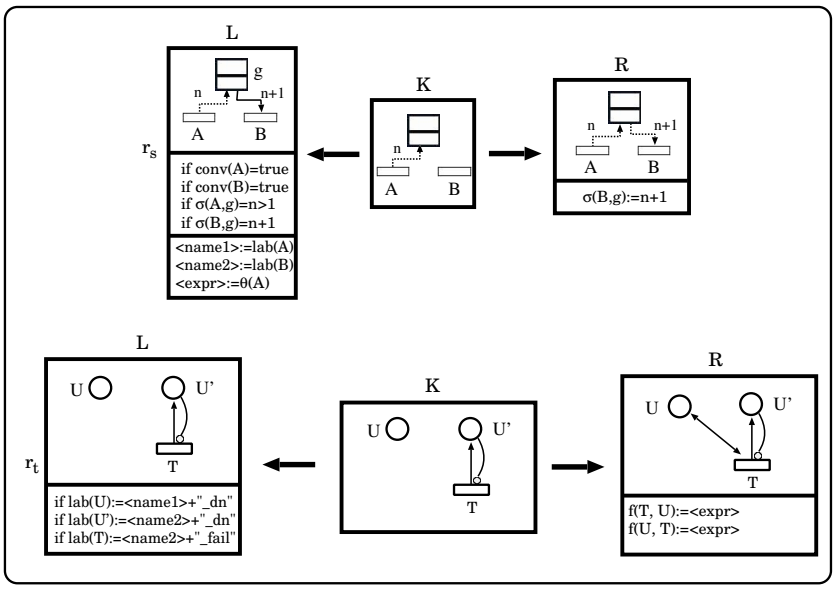

Figure A.20: Compound rule for a gate of type SEQ having two non replicator events as dependent events, with order numbers n and $n+1$ respectively.

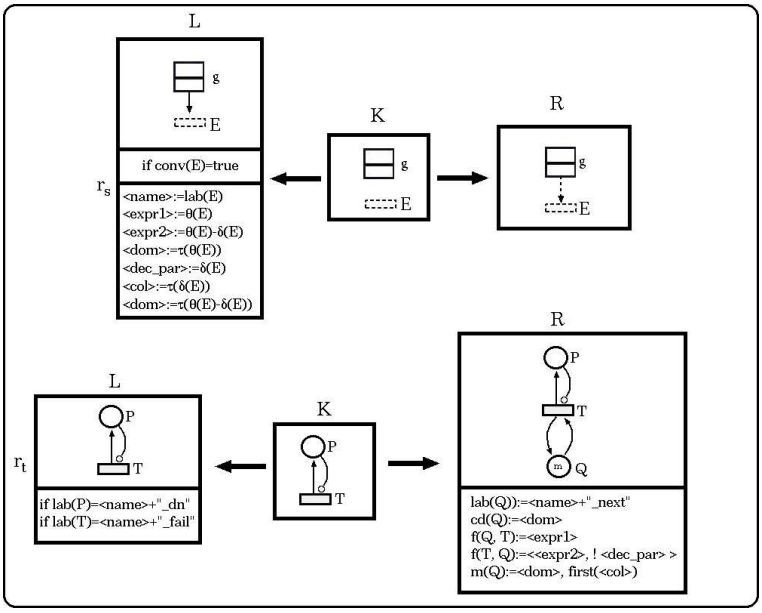

Figure A.21: Compound rule for a gate of type SEQ having a replicator event as dependent event.

- the rule in Fig. A.27 to every replicator event in the basic coverage set of the RB;
- the rule in Fig. A.28 to every non basic non replicator event in the coverage set of the RB;
- the rule in Fig. A.29 to every non basic replicator event in the coverage set of the RB.

We omit the compound rules for the conversion of a RB with a RE as trigger event; such rules differ from the rules in Fig. A.25, in Fig. A.26, in Fig. A.27, in Fig. A.28 and in Fig. A.29, only by the presence of a RE as trigger event of the RB, instead of an IE.

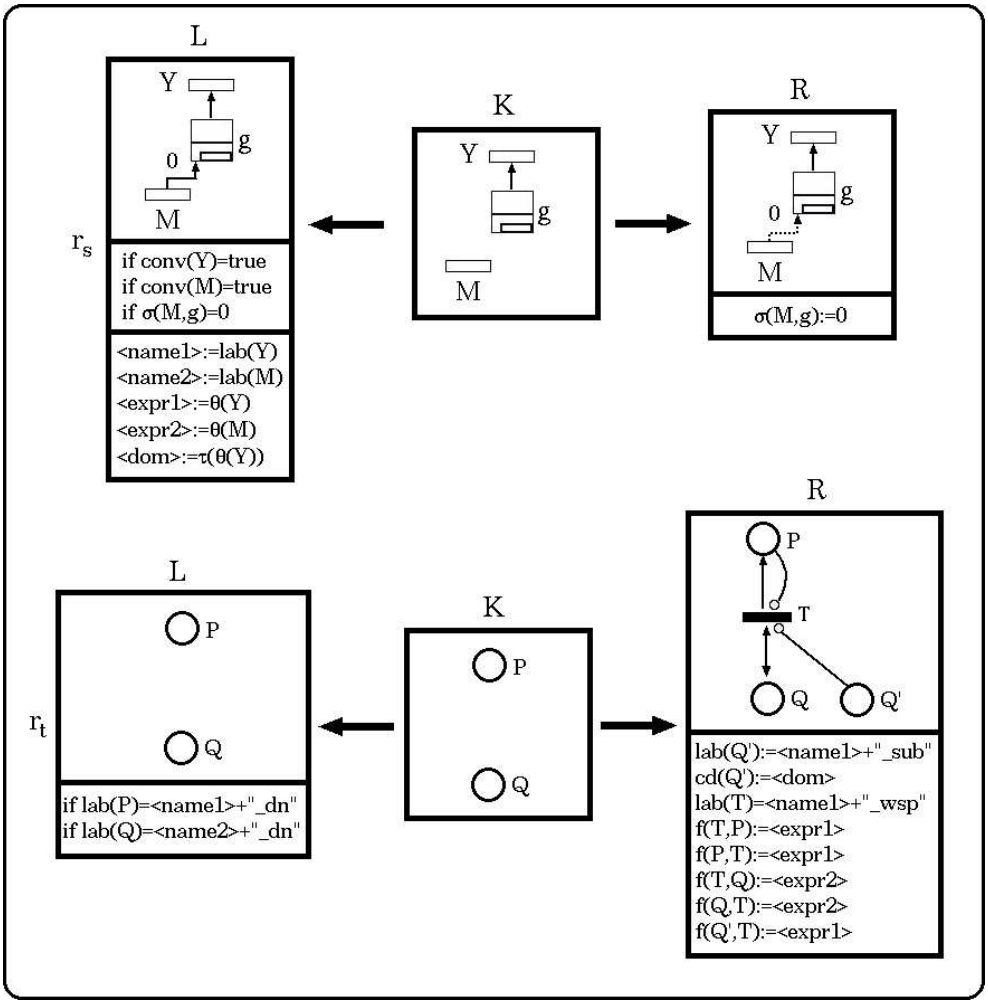

Figure A.22: Compound rule for a gate of type WSP with an IE as output event.

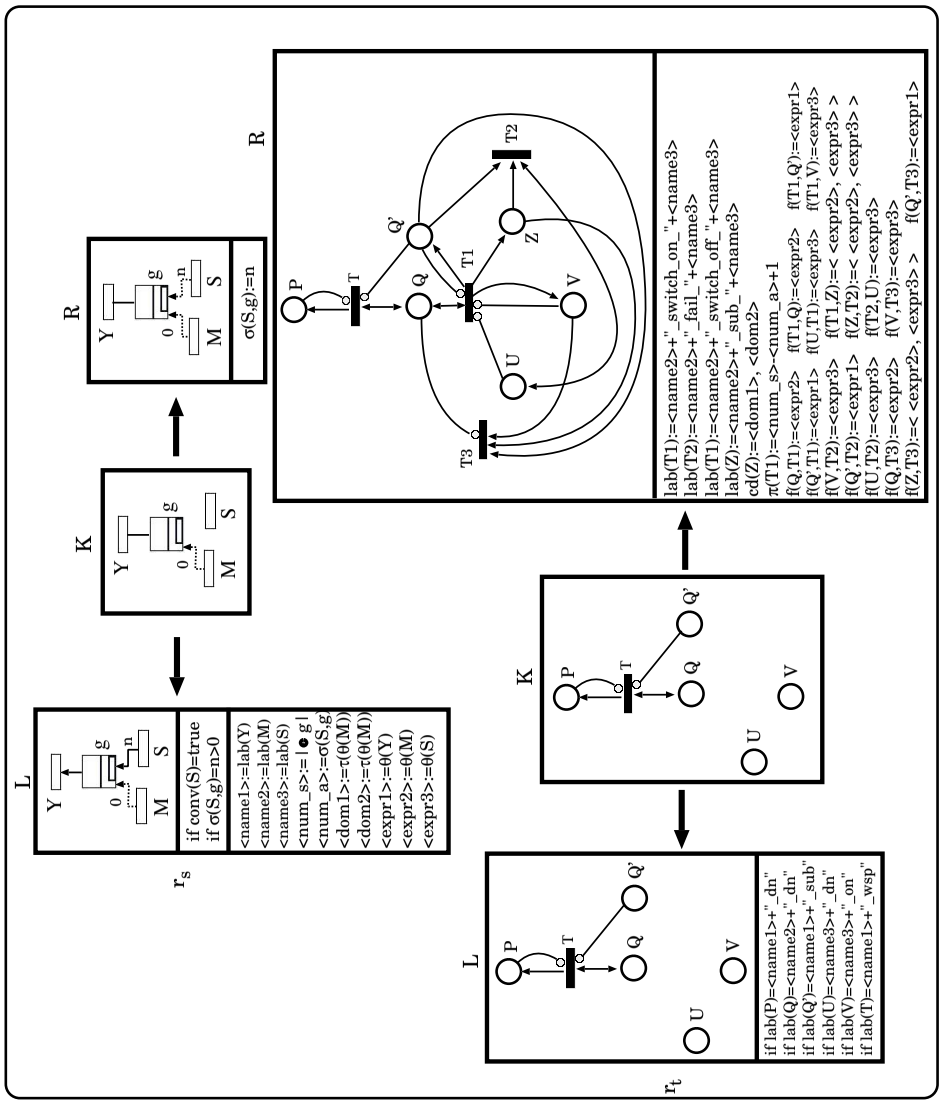

Figure A.23: Compound rule for a gate of type WSP having a non replicator event as input event (and an IE as output event).

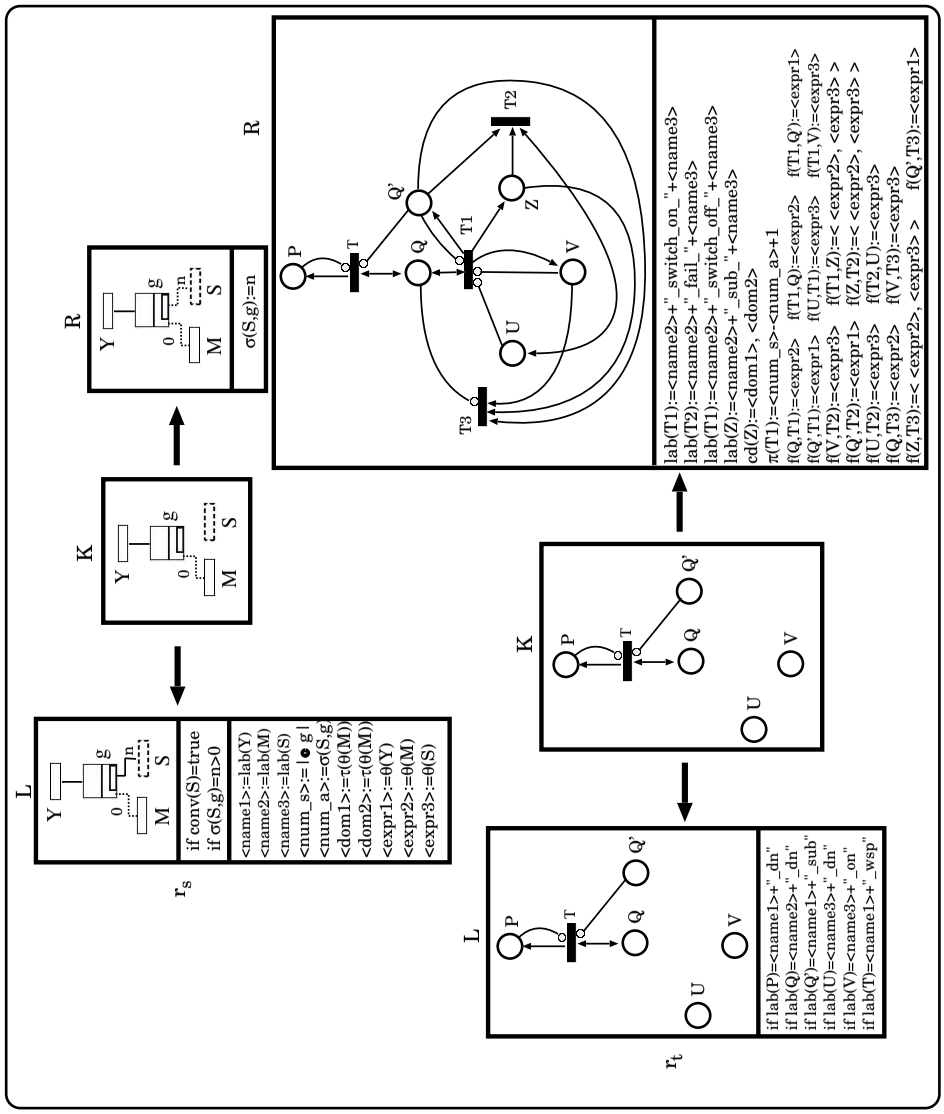

Figure A.24: Compound rule for a gate of type WSP having a replicator event as input event (and an IE as output event).

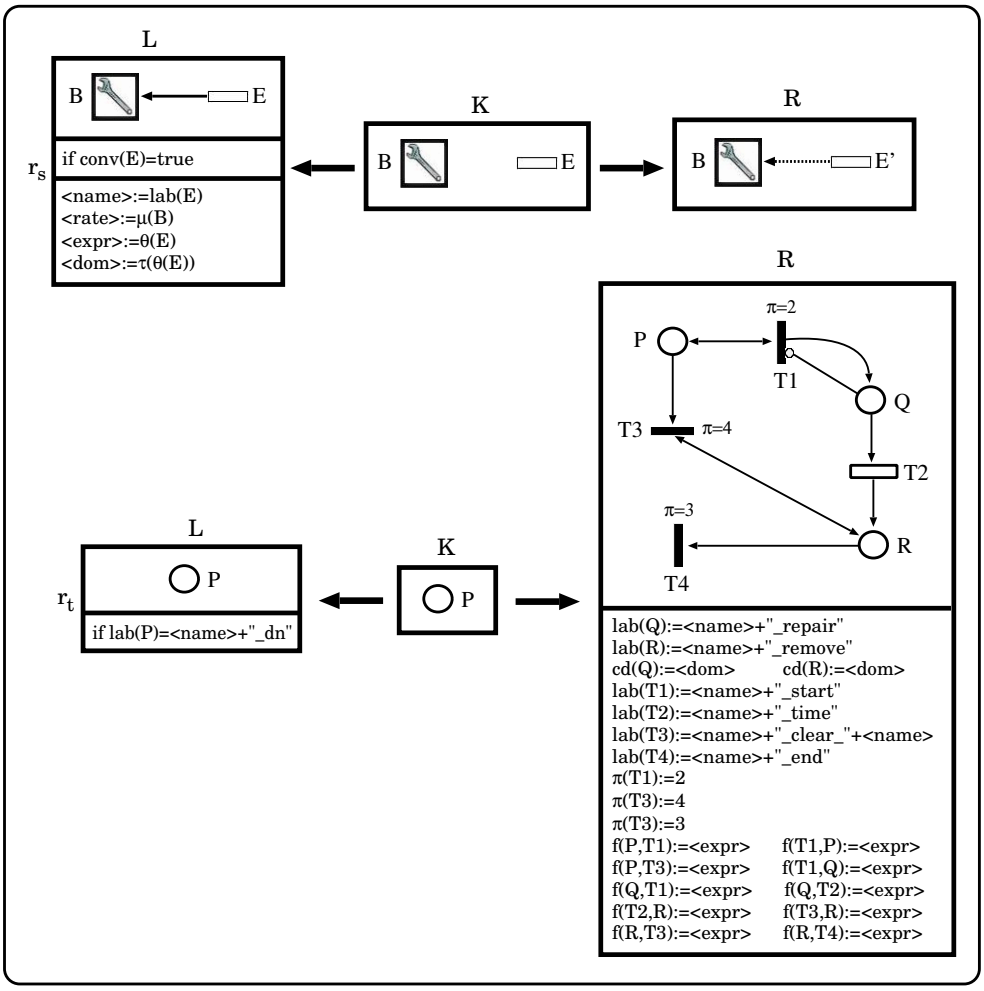

Figure A.25: Compound rule for the trigger event of a RB; the trigger is a non replicator event.

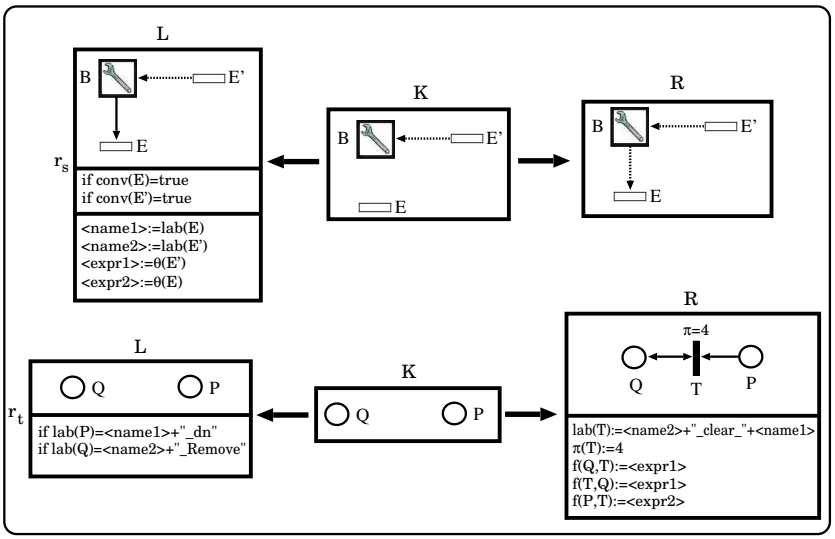

Figure A.26: Compound rule for a non replicator event in the basic coverage set of a RB whose trigger event is a non replicator event.

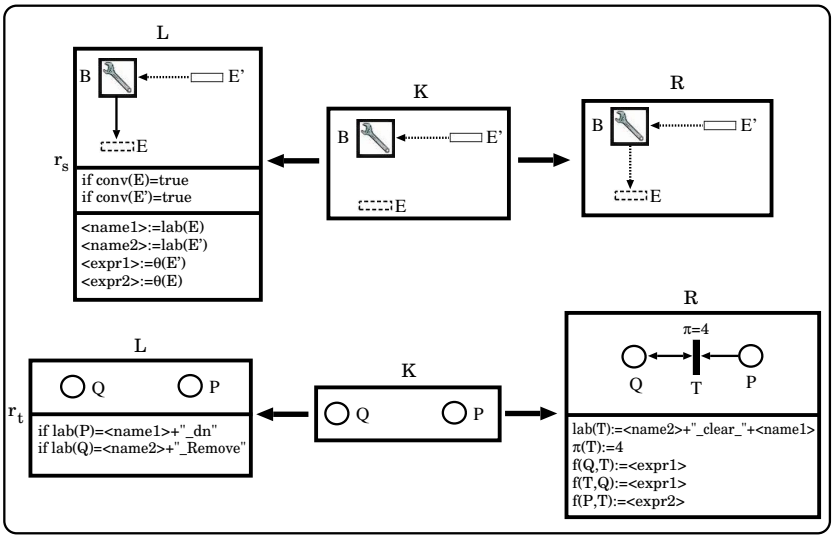

Figure A.27: Compound rule for a replicator event in the basic coverage set of a RB whose trigger event is a non replicator event.

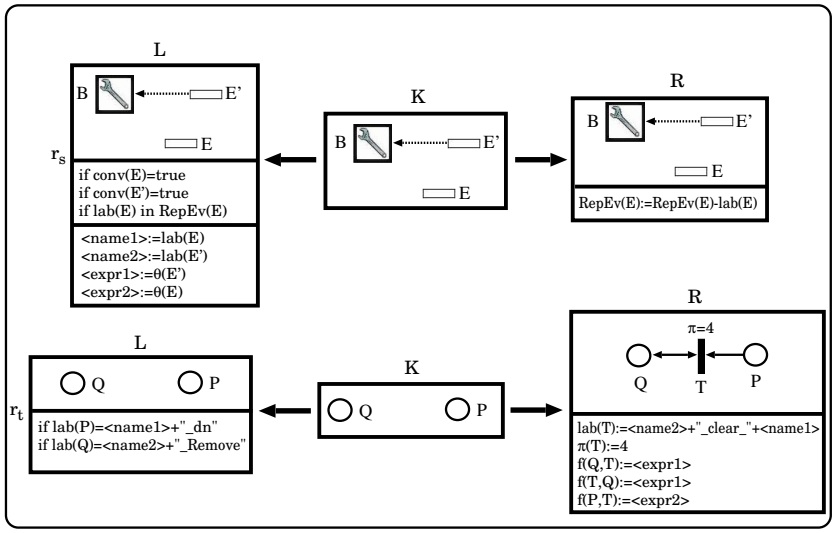

Figure A.28: Compound rule for a non basic non replicator event in the coverage set of a RB whose trigger event is a non replicator event.

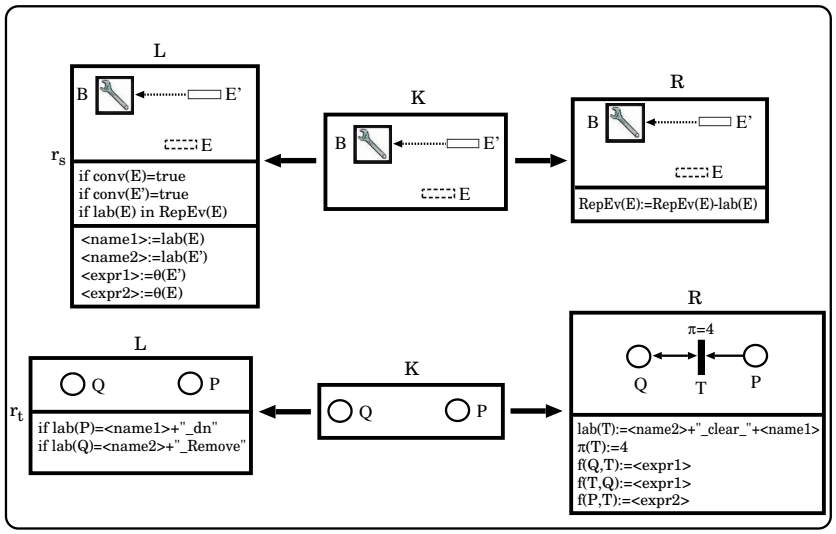

Figure A.29: Compound rule for a non basic replicator event in the coverage set of a RB whose trigger event is a non replicator event.