Parametric NdRFT for the derivation of optimal repair strategies

Marco Beccuti, Giuliana Franceschinis,
Daniele Codetta-Raiteri
DI, Università del Piemonte Orientale
Via T. Michel, 11. Alessandria, Italy
{beccuti, giuliana, raiteri}@mfn.unipmn.it

Serge Haddad
LSV, ENS Cachan, CNRS
61, avenue du Président Wilson. Cachan, France
haddad@lsv.ens-cachan.fr

Abstract

Non deterministic Repairable Fault Trees (NdRFT) are a recently proposed modeling formalism for the study of optimal repair strategies: they are based on the widely adopted Fault Tree formalism, but in addition to the failure modes, NdRFTs allow to define possible repair actions. In a previous paper the formalism has been introduced together with an analysis method and a tool allowing to automatically derive the best repair strategy to be applied in each state. The analysis technique is based on the generation and solution of a Markov Decision Process. In this paper we present an extension, ParNdRFT, that allows to exploit the presence of redundancy to reduce the complexity of the model and of the analysis. It is based on the translation of the ParNdRFT into a Markov Decision Well-Formed Net, i.e. a model specified by means of an High Level Petri Net formalism. The translated model can be efficiently solved thanks to existing algorithms that generate a reduced state space automatically exploiting the model symmetries.

Keywords: Fault Trees, Optimal repair strategy, Markov Decision Process, Symmetries, Well-Formed Nets

1 Introduction

The Fault Trees (FT) [15] are a well-known formalism for the evaluation of dependability of complex systems. They provide an intuitive representation of the system in terms of its faults, modeling how the combinations of failure events relative to the components of the system, can cause the failure of the subsystems or of the whole system.

Recently an extension to FTs called *Non deterministic Repairable Fault Tree* (NdRFT) has been proposed [3]; it is oriented to the study of optimal repair strategies. NdRFT models are based on the widely adopted Fault Tree formalism, but in addition to the failure modes, NdRFTs allow to define possible repair actions. Given the failure modes and the possible repair actions in the system, the optimal repair strategy can be determined, using the analysis method and tool presented in [3, 4]. The analysis technique is based on the generation and solution of a *Markov Decision Process* (MDP) [12]. In this paper we present an extension to NdRFT called Parametric NdRFT (ParNdRFT) which allows to model in a compact way the redundancies in the system, as well as the available repair actions. The goal is the same as in the NdRFT formalism: computing the optimal repair strategy. This means determining what it the set of repair actions to activate in each state of the system in order to minimize the system unavailability.

The advantage of ParNdRFT is not limited to the compact modeling of the system: the presence of redundancy expressed in parametric form is exploited in order to reduce the complexity of the model analysis. Actually the Parametric NdRFT analysis is based on the translation of the model into a Markov Decision Well-Formed Net (MDWN), i.e. a model specified by means of an High Level Petri Nets formalism. From the MDWN model a MDP characterized by a reduced state space can be obtained. This is achieved by means of efficient existing algorithms to generate and analyze the reduced state space automatically exploiting the model symmetries [5]. From such analysis the best repair strategy is obtained. The paper is structured in this way: Sec. 2 presents some related work about modeling repair in FTs; Sec. 3 describes the Parametric NdRFT formalism, while in Sec. 4 we show how a ParNdRFT model can be converted into a MDWN, and provide a sketch of the proof of the translation correctness. An example application is presented in Sec. 5. A comparison of the proposed approach with respect to other possible approaches is provided in Sec. 6. Conclusions and perspectives of this work are presented in Sec. 7.

2 Related work on Fault Trees with repair

The analysis of an FT model returns several dependability measures such as the system *reliability* versus time, and can be supported by several software tools. Some of them

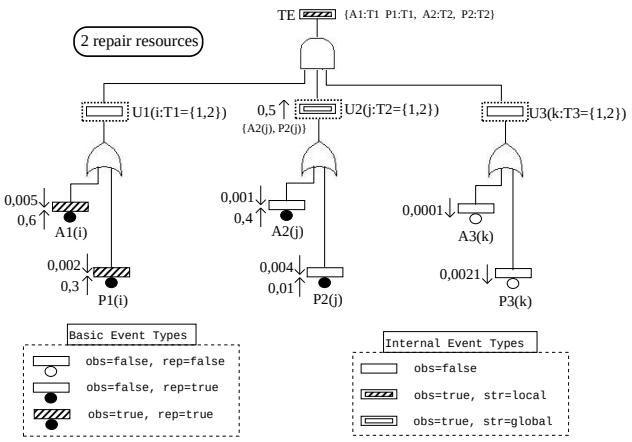


Figure 1. Example of Parametric NdRFT.

allow to model the repair of single components, by associating the repair probability with the basic events (BE). This is the case for instance of *Stars Studio* and *ASTRA* [9].

Other tools allow to model more complicated failure and repair modes for the components of the system, by means of hierarchical modeling: the failure and the repair modes of a component can be modeled by submodels conforming to other formalisms and combined with the main FT model. This is the case of *HIMAP* [10] and *SHARPE* [14]: *HIMAP* allows to edit *Continuous Time Markov Chain* (CTMC) submodels, while *SHARPE* can deal also with *Generalized Stochastic Petri Net* (GSPN) [1].

The *Repairable Fault Tree* (RFT) formalism [8] allows to model the repair of a subsystem according to a *repair policy* specifying several aspects ruling the repair process. In RFT, a new node called *Repair Box* (RB) is used to represent the repair of a subsystem and to set the repair policy. An RFT model requires the (partial) analysis of its state space, carried out by conversion into a GSPN.

In the NdRFT formalism [3] the repair policy (or strategy) is not predefined, but it is the result of the model analysis: given the specification in the model of several repair options, the analysis of the NdRFT model returns the optimal repair strategy which indicates for each state of the system which are the repair options to be activated in order to minimize the global system failure probability. This is done by defining the NdRFT semantics in terms of a MDP, and then solving the optimization problem using the methods available for MDPs.

The formalism presented in this paper is the Parametric NdRFT which extends NdRFT by integrating the parametric form inherited by the *Parametric Fault Tree* (PFT) [6] formalism and used to represent in a compact way the symmetries in the system.

3 Parametric NdRFT formalism

In this section the Parametric NdRFT formalism is introduced: it is very similar to the NdRFT but it allows to parametrize models in the same way PFT allows to do so for the FT; hence in case several copies of the same subtree are present (at any level of the NdRFT hierarchy) only one copy is explicitly defined in a parametric way: one index together with its range are introduced to express the fact that several replicas of a given event/subtree exist. The repair actions definition is also adapted to be parametric.

Definition 1 (ParNdRFT) *A Parametric NdRFT is a 7-tuple*

$$\mathcal{F} = \langle \mathcal{T}, \mathcal{P}, \mathcal{E}, \mathcal{G}, \mathcal{A}, \mathcal{R}, res_0 \rangle$$

where:
$\mathcal{T}$ is a set of finite and not empty sets called $types$ and denoted $T_i, i = 1, \ldots, n$;
$\mathcal{P}$ is a set of typed parameters $x_j : T_i$;
$\mathcal{E}$ is the set of possibly parametric events with associated the following attributes: $params \subseteq \mathcal{P}$, $decl \subseteq params$, $rep, obs \in \{true, false\}$, $fprob, rprob \in [0, 1]$, $str \in \{global, local\}$, $res \in Bag(\mathcal{R})$. Parametric events actually stand for a class of events, and the attributes associated with an event class are identical for all events in the class.
$\mathcal{G}$ is the set of gates; $\mathcal{E} \cap \mathcal{G} = \emptyset$. A gate g has a type[1] denoted $g.type \in \{\texttt{and}, \texttt{or}\}$;
$\mathcal{A}$ is the set of arcs, a subset of $\mathcal{E} \times \mathcal{G} \cup \mathcal{G} \times \mathcal{E}$. For x belonging to $\mathcal{E} \cup \mathcal{G}$, we denote $x^\bullet \equiv \{y \mid (x, y) \in \mathcal{A}\}$ and ${}^\bullet x \equiv \{y \mid (y, x) \in \mathcal{A}\}$. $\mathcal{A}$ satisfies:
1. $\forall g \in \mathcal{G}, |g^\bullet| = 1$ and $\forall e \in \mathcal{E}, |{}^\bullet e| \leq 1$
2. There is exactly one event, denoted $\top$ and called Top Event, s.t. $\top^\bullet = \emptyset$; all other events satisfy $|e^\bullet| \geq 1$
3. The set of events can be partitioned into basic *events (BE) $\underline{\mathcal{E}} \equiv \{e \mid {}^\bullet e = \emptyset\}$ and* internal *events (IE) $\overline{\mathcal{E}} \equiv \{e \mid {}^\bullet e \neq \emptyset\}$*
4. The (directed) graph induced by $\mathcal{A}$ is acyclic.
$\mathcal{R}$ is a set of resource types; $res_0 \in Bag(\mathcal{R})$ is a multiset on $\mathcal{R}$ defining the number of available resources of each resource type. For each repairable event e, $e.res \subseteq res_0$.

Let us describe in more details the attributes associated with the events (or event classes). If $e.params = \emptyset$ then e represents a single (failure) event; if instead $e.params \neq \emptyset$, it denotes the set of parameters of event class e (the class with its parameters as well as the generic event in the class is denoted $e(x_1, \ldots, x_n)$, while $e(v_1, \ldots, v_n)$ with $v_i \in T_i$ denotes one event instance in the class). Attribute $e.decl \subseteq e.params$ is the set of parameters *declared* in e; each parameter can be declared in one and only one event ($e \neq e' \rightarrow e.decl \cap e'.decl = \emptyset$). If $e.decl \neq \emptyset$ then e

[1] Since the proposed optimization method is based on the state space, other gate types could easily be considered, including dynamic ones: in this paper only and/or gates are considered for the sake of space.

is called *replicator*, meaning that several events in class e (differing only for the value of some parameter in $e.decl$) are connected to the gates that have e as input. A well defined ParNdRFT must satisfy some further constraints on the parameters, meant to avoid redundant or inconsistent failure event specification (the same defined for PFTs [6]) Boolean attribute obs is related to the possibility of detecting the corresponding failure (which is a prerequisite to start a repair). $e.fprob$ represents the fault probability of e. The attribute rep, indicating if e is repairable or not, and if $e.rep = true$ it also has a repair probability attribute $e.rprob$ and attribute $e.res$, defining how many resources of each type are needed to perform the repair action.

IE that are observable have an associated repair strategy attribute $str \in \{global, local\}$. If $e.str = global$ then a repair rate attribute $rprob$ and a resource requirement res is also associated with e. The attribute $torep$ instead defines the set of events whose repair is triggered by event e: elements of $torep$ must satisfy that $e' \in e.torep \rightarrow e'.rep = true$ and there is a path from e to e' according to $\mathcal{A}$; they may be parametric, and the parameters associated with a given element e' appearing in $e.torep$ must be a subset of $e.params \cap e'.params$, moreover the syntax $e'(i, T_j)$ denotes $|T_j|$ events in class e' and can be used when e' is an event in the subtree rooted in e, and has one parameter (the second parameter in the example) of type T_j which is declared in e' or in a node on the path from e' to e (hence all events $e'(i, T_j)$ are included in the e subtree).

Let us consider the ParNdRFT submodel of Fig. 1. The set of types is $\mathcal{T} = \{T_1, T_2, T_3\}$: in the figure each of them comprise two elements, but this can change (this is exactly where the parametric nature of the model becomes evident). The set of parameters is defined as $\mathcal{P} = \{i : T_1, j : T_2, k : T_3\}$: even if apparently the three types coincide, it is important to keep them separated to express the fact that they can vary independently. In Sec. 5 we perform a set of experiments for different sizes of T_1,T_2 and T_3. In the model there are nine parametric events (corresponding to nine event classes) plus one Top Event; among them, three are replicator events (where the three parameters are declared), namely $U1$, $U2$ and $U3$. Four out of six basic event classes are repairable, two of them (namely $A1(i)$ and $P1(i)$) are observable and can trigger immediately a local repair action, moreover intermediate event class $U2(j)$ can trigger a global repair action involving $A2(j)$ and $P2(j)$ (as expressed by the string in curly brackets below node $U2$). Finally the TE can trigger a local repair involving the events $A1(i)$, $P1(i)$, $A2(j)$, $P2(j), \forall i \in T_1, j \in T_2$ (this is expressed in the picture by the notation $\{A1(T_1), P1(T_1), A2(T_2), P2(T_2)\}$ appearing next to the TE).

ParNdRFT unfolding. The simplest way of defining ParNdRFT semantics is to give the rules for *unfolding* any ParNdRFT model into an equivalent NdRFT. The unfolding procedure is simple, and consists in substituting each subtree rooted in a replicator event with one copy for each possible element of the Cartesian product of the types of parameters in $e.decl$: in each replica, all parameters in $e.decl$ must be substituted in each parametric event of the subtree with the values associated with that replica. The procedure must start with the innermost subtrees rooted in a replicator event, and proceed with the subtrees containing them until the top event is reached. The unfolding procedure however is not applied in practice because the main benefits of ParNdRFTs, from the point of view of efficient analysis, would be lost. The unfolding of the example in Fig. 1 consists in doubling each subtree $U_h, h = 1, \ldots, 3$ instantiating their parameters so that the AND gate in the final model has six inputs (three pairs of similar subtrees). In the next section we show how a ParNdRFT can be directly translated into a Markov Decision Well-Formed Net (MDWN) and how a MDP can be derived from the MDWN: such MDP can be considerably smaller than the MDP corresponding to the unfolded NdRFT. Despite the MDP reduction, no approximations are introduced: this is due to the homogeneous behavior of all the events in each event class, which induces a symmetric structure in the underlying MDP that can be exploited to *lump equivalent states*.

4 Translation of ParNdRFT into MDWNs

Markov Decision Well-Formed Nets. An MDWN is a high level formalism introduced in [5] to specify Markov Decision Processes (MDP). Its definition is based on Well-Formed Nets, an high level Petri net formalism that combines a powerful modeling language with the possibility to implicitly express in the model the behavioral *symmetries*, which are reflected also at the level of the state space and can be automatically exploited to reduce the state space size, by means of an algorithm for the construction of the so-called Symbolic Reachability Graph (SRG) [7].

The main features of MDWNs are the possibility to specify the general behavior as a composition of several components, that may have similar behavior, and some of which are controllable; moreover each MDP non deterministic or probabilistic transition can be composed by a set of non deterministic or probabilistic steps, each one involving a subset of components. MDWNs allow to exploit the symmetries (due to the presence of similarly behaving components) by deriving from the SRG a MDP of reduced size w.r.t. the original one, on which the same results can be computed more efficiently.

An MDWN model is composed of two parts, both specified using the WN formalism: the WN^{nd} subnet and the

WN^{pr} subnet (describing the non deterministic (ND) and probabilistic (PR) behavior respectively); the two subnets share the color classes definition and the set of places, while transition sets are disjoint. A subset of the color classes may be used to identify (sets of similarly behaving) system components. In both subnets the transitions are partitioned into "run" and "stop" subsets, and each transition has an associated set of components involved in its firing (possibly specified in a parametric way on the transition color). "Run" transition firings represent intermediate steps in a ND/PR transition at the MDP level, while "stop" transitions represent the final step in a ND/PR transition, for all components involved in it. Transitions in WN^{pr} have a "weight" attribute, used to compute the probability of each firing sequence. An MDWN model behavior alternates between ND transition sequences and PR transition sequences, initially starting from a ND state. The PR sequences are determined according to the WN^{pr} structure, and include exactly one stop transition for each component; the ND sequences are determined by the WN^{nd} structure, and include exactly one stop transition for each controllable component plus a stop "global" transition.

The generation of the (reduced) MDP corresponding to a given MDWN consists of (1) a composition step, merging the two sub-nets in a single net, (2) the generation of the (S)RG of the composed net, (3) two reduction steps transforming each PR and ND sequence of the (S)RG into a single (reduced) MDP transition.

Translating a ParNdRFT into a MDWN. The translation of a ParNdRFT into a MDWN has been defined by integrating the translation of a NdRFT into a MDPN [4] and the translation of a PFT in a SWN [6]. The translation is described here in an intuitive way through the introduction of *template submodels* corresponding to the different elements appearing in the ParNdRFT and composition rules to obtain the whole MDWN model from the submodels.

Color classes and components. A preliminary step in defining a MDWN is the color classes setup: each parameter type T_h in the ParNdRFT corresponds to a basic color class C_h in the MDWN. Each place and transition in MDWN models have a color domain: the template submodels described in next sections correspond to the translation of each (class)event in the ParNdRFT, hence the color domain of the places representing the state of a given event class e can be derived from the attribute $e.param$ (Cartesian product of the color classes corresponding to its parameter types). The functions $\langle x_1, \ldots, x_n \rangle$ appearing on the arcs can be interpreted as parameter tuples: the meaning is that each transition withdraw or generate "colored" tokens (i.e. tokens carrying some information), and the color represents the identity of a specific event within a given event class.

MDWN definition includes a set of components, and among these a subset of controllable components. There is one component for each event in the ParNdRFT. Controllable components are all basic events that can undergo a local repair and all global repair trigger events. Components corresponding to parametric events can be identified by composing the event class name and the a set of parameter values (hence we can call them parametric components).

Probabilistic subnet. Fig. 2 shows how each basic event class e can be translated in a WN^{pr} submodel according to its rep attribute: each non repairable event class is translated into subnet **A** while each repairable event class is translated into subnet **B**. Observe that all the places in the subnets are annotated with a color domain $C_1 \times \ldots \times C_n$ derived from the definition of $e.params$. Places UP_e, $DOWN_e$ and $UnderRepair_e$ model the state of each (basic) event in the class e. "Run" and "stop" transitions have different icons, so that they can be easily recognized. For instance $WorkR_e$ is a run transition for component $e(x_1, \ldots, x_n)$, while $WorkS_e$ is a stop transition for the same component.

At each probabilistic step an Up component can either remain Up (stop transition $Work_e$) or go $Down$ (sequence $FailR_e$, $FailS_e$). A $Down$ component can either remain $Down$ (stop transition $FailS_e$) or start its repair (run transition $Repair$, either followed by stop transition $ContRep_e$, meaning that the repair has not completed in the current time unit, or by the sequence $EndRepR_e$, $EndRepS_e$ if the repair completes). Observe that all these transitions have an associated set of *variables*, corresponding to the set of parameters of the events; this is reflected also in the tuple $\langle x_1, \ldots, x_n \rangle$ appearing on the arcs. The firing of these transitions (representing a state change of one event) requires the instantiation of the parameters to a specific value within their color class (i.e. within their ParNdRFT type). Place $Assigned_e$ is set by the WN^{nd} when a decision to repair a given $e(v_1, \ldots, v_n)$ within class e is taken (represented by the presence of a token of color $\langle v_1, \ldots, v_n \rangle$ in the marking $m(Assigned_e)$ of such place). Places AV_RES_i represent the resources, and they become available as the repair ends. The $rprob$ and $fprob$ attributes associated with the events are used to properly weight the transitions representing failure and end/continuation of repair actions.

The conversion rule for an AND or OR gate g and its output event e is shown in Fig. 3. We emphasize that the WN models in Fig. 3 are templates that must be instantiated according to the set and type of input events of each gate. Subnets **C** and **E** simply model the propagation of the faults from the input events of the gate to its output event. IE that are not observable or have local repair strategy are translated into these simple subnets. Those with a global repair strategy have an additional subnet (common to both gate types) shown on the right: this subnet represents the corresponding global repair process. Observe that the annotations appearing on the arcs of the AND gates may differ

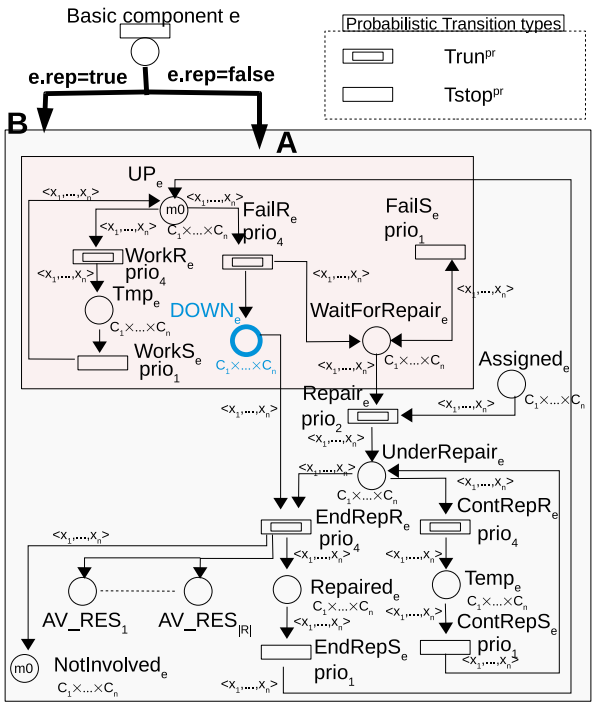


Figure 2. Translation of the BEs (WN^{pr} submodel).

depending whether the input parametric event e' is a replicator or not. If the input event is a replicator, then all the occurrences of the parameters in $e'.decl$ must be substituted by the *synchronization function* S_i which is a constant function which evaluates to the color class C_i: this allows to model the "and" semantics of the gate taking into account the set of subtrees represented by each replicator (thinking in terms of the unfolding may help understanding).

Algorithm 1 is used to generate the WN^{pr} submodel: it first instantiates an appropriate template submodel for each event (and corresponding gate, for IE) according to its type and attributes; the instantiation is performed by function $WN(event, template)$ and requires to appropriately rename the places and transitions and to assign proper color domain and arc functions depending on the involved event parameters; then it merges all submodels into a unique net by means of method *Compose()* that performs a composition by superposition of places with equal label, moreover it generates the color class definition for the whole net (which coincides with the ParNdRFT parameters type definition).

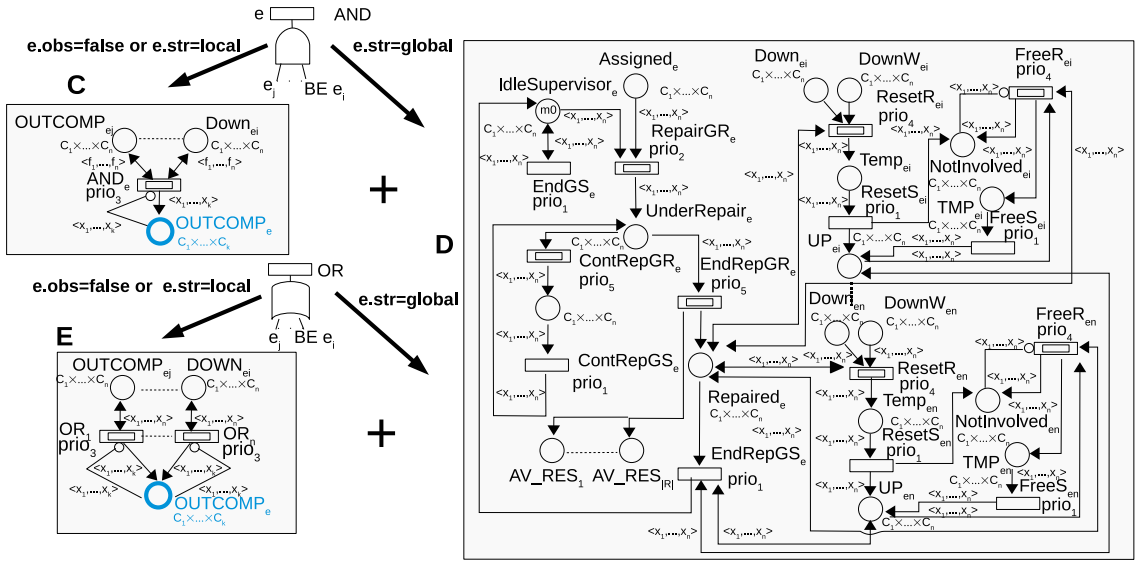
Non deterministic subnet. The WN^{nd} subnet is built according to the templates depicted in Fig. 4. The basic idea is that the WN^{nd} submodel represents the decision whether a repair action must be started for any down BE and for any down IE with global repair strategy. Firing of stop transition

Algorithm 1: Algorithm generating WN^{pr}

```
Class WN^pr GenerateWN^pr(Class ParNdRFT F )
Input: F is a ParNdRFT model
Output: A WN^pr model
set WNet= ∅;
set Events= insert_events(F);
while Events ≠ ∅ do
    e =Events.extract();
    if (e ∈ Ē) g =• e;
    then switch e do
        case (e ∈ E − E_R) WNet.insert(WN(e, A));
        case (e ∈ E_R) WNet.insert(WN(e, B));
        case (e ∈ Ē ∧ g.type = AND)
            if (e.obs = false ∨ e.str = local) then
            WNet.insert(WN(e, C));
            else WNet.insert(WN(e, C + D));
        end
        case (e ∈ Ē ∧ g.type = OR)
            if (e.obs = false ∨ e.str = local) then
            WNet.insert(WN(e, E));
            else WNet.insert(WN(e, E + D));
        end
    end
end
WNet.Compose();
return WNet.extract();
```

$NoAssign_e$ means that a no repair decision has been taken for event e, while firing of stop transition $Assign_e$ corresponds to the opposite decision: observe that the second decision can be taken only if the needed resources are available (input places AV_RES_i) and the BE is not involved in some global repair (input place $NotInvolved_e$). Start of a local repair action triggered by observable BEs is represented by subnet G. Start of local repair actions triggered by observable internal events are represented by subnet L, start of global repair actions are represented by subnet I. Subnet H instead is needed for technical reasons: it is used to clear the state of the IE which must be recomputed at the end of each probabilistic step (after all fail/repair steps have been taken for basic events). Notice that all these transitions are parameterized as the corresponding event: each transition $NoAssign_e$ or $Assign_e$ must be fired for all possible instances of their parameters (appearing in the tuples labeling the arcs). Algorithm 2 generates the WN^{nd} submodel of the MDWN. It operates similarly to Algorithm 1 by first instantiating the appropriate template (using function $WN(event, template)$) for each event that can trigger a repair (these are the controllable components of the MDWN) plus the template used to clear the IE status so that they can be set again in the next probabilistic step. Finally the subnets are composed by places superposition.

An example of MDWN corresponding to the subtree rooted by U_1 in Fig. 1 is shown in Fig. 5. The WN^{pr} is obtained by the instantiation of two templates B corresponding to the BEs $A1$ and $P1$ and one template E corresponding to the OR gate of IE $U1$. Instead, the WN^{nd} is obtained by the instantiation of two templates G modeling the repair decisions for $A1$ and $P1$ and one template H used to to clear the state of $U1$.



AND gate: $\forall\langle f_1,\ldots,f_n\rangle \Rightarrow f_i = x_i$ *iff* $x_i \in e.parm \setminus e.decl$ *otherwise* $f_i = S_i$

Figure 3. Translation of the AND/OR gates and the corresponding output event (WN^{pr} submodel).

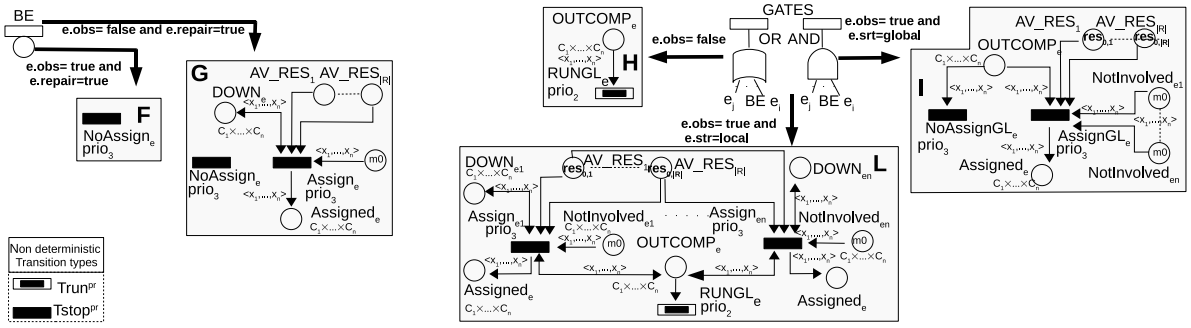


Figure 4. Translation of the BEs, the AND/OR gates and the corresponding output event (WN^{nd} submodel).

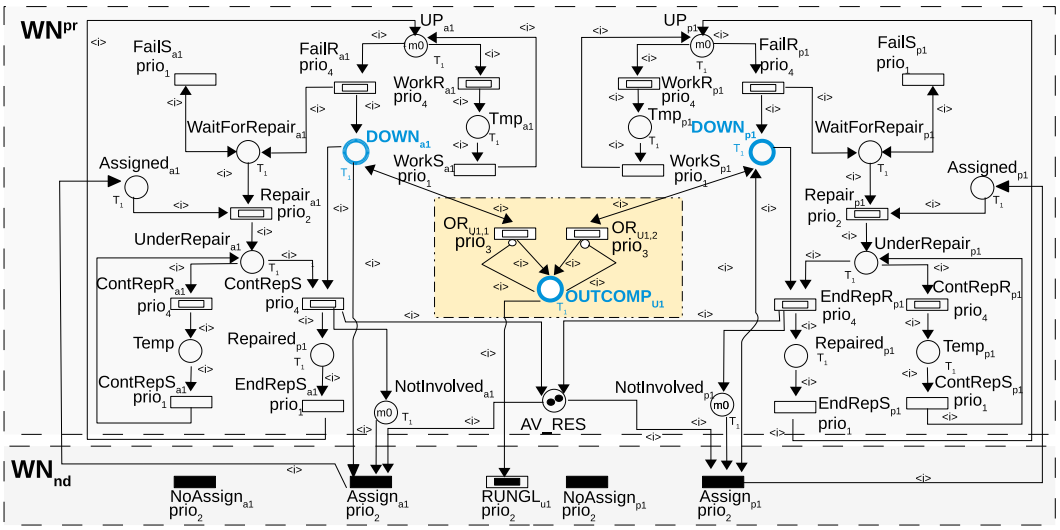


Figure 5. An example of MDWN considering only the subtree $U1(i)$ in Fig. 1.

Algorithm 2: Algorithm generating WN^{nd}

```
Class WN^nd GenerateWN^nd(Class ParNdRFT F)
Input: F is a ParNdRFT model
Output: A WN^nd model
set WNet=∅;
set Events=insert_events(F);
while Events ≠ ∅ do
    e=Events.extract();
    switch e do
        case (e ∈ E_R ∧ (!e.obs)) WNet.insert(WN(e, F));
        case (e ∈ E_R ∧ e.obs) WNet.insert(WN(e, G));
        case (e ∈ Ē ∧ (!e.obs)) WNet.insert(WN(e, H));
        case (e ∈ Ē ∧ e.obs)
            if (e.obs and e.str = global) then
            WNet.insert(WN(e, I));
            else WNet.insert(WN(e, L));
        end
    end
end
WNet.insert(WN(NULL, STOPGL));
WNet.Compose();
return WNet.extract();
```

The reward function. In order to analyze the MDWN model the reward function to be optimized must be specified. One simple example of such function, suitable if the objective is to have a low probability of finding the whole system down is: $r = -1$ if $m(OUTCOMP_{TE}) = 1$ otherwise 0 associated with the model state.

This means that a negative reward (corresponding to a penalty) is associated with each state where the TE failure is true. All other states and all actions have reward of 0. Hence, every time unit spent in a state with a TE failure produces a penalty of -1. The optimization problem consists in finding the strategy that maximizes the reward (i.e. that minimizes the penalty).

More complex reward structures can be devised to take into account the cost of repair actions, as well as the penalties when the system is in a degraded state (some subsystem down while the global fault tolerant system is still up, causing the system to work with degraded performance).

Correctness of translation. Let us discuss the correctness of the proposed translation. In [4] it has been proven that the MDP semantics of a NdRFT corresponds to the MDP derived from the MDPN obtained by automatic translation. A similar argument could be used to prove the correctness of the translation proposed in this paper, however a simpler argument can be used in this case: in Sec. 3 the semantics of a ParNdRFT has been defined in terms of its unfolding into a NdRFT (that we denote *unfold*$_{FT}$*(ParNdRFT)*). A simple proof of correctness consists in showing that *unfold*$_{WN}$(*transl(ParNdRFT))* = *transl(unfold*$_{FT}$*(ParNdRFT))*, where *unfold*$_{WN}$ is the classical unfolding function for high level Petri nets (in particular WNs). The correctness then follows directly from that of the NdRFT translation.

Let us give a sketch of proof for the above statement: observe that the ParNdRFT unfolding corresponds to replicating all event class nodes (and the associated gates, for IE) in the model as many times as the possible instantiations of their parameters. The arcs must then connect the nodes with corresponding parameter values. In case of a replicator event e, several instances of the same event class may be input to the same gate (namely those which differ only for the value of some parameter in $e.decl$).

Considering the MDWN obtained from the ParNdRFT translation, first of all observe that the net structure translating a BE is isomorphic to the corresponding net structure translating a single event in *transl(NdRFT)*, the only difference being the color annotations. The places corresponding to the possible states of each event e have color domain $T_1 \times \ldots \times T_n$ where T_i is the type of the i-th parameter x_i of e; hence in the MDWN unfolding these places are replicated as many times as the corresponding event class in the ParNdRFT unfolding. Similarly for transitions: in fact, the functions appearing on arcs are $f = \langle x_1, \ldots, x_n \rangle$ so that the possible color instances of the transition are the same as those of the connected places, moreover function f corresponds to an identity function so that the unfolded arcs will connect places and transition replicas obtained with the same instantiation of parameters. The same explanation applies to the translation of all the submodels representing the repair processes as well as the resource assignment actions (the latter are in the non deterministic part of the MDWN).

The translation of gates is slightly more complex: first of all, the output event of a gate may have different parameters than its input events (it is a possibly empty subset of the union of the parameters of all input events). The arc functions appearing on the arcs of the gate subnets can be either simple projections $\langle x_1, \ldots, x_n \rangle$, or they may include also a synchronization function $\langle x_1, \ldots, S_i, \ldots, x_n \rangle$: the latter case happens only in the translation of the AND gate (see Fig. 3.C). The synchronization function is used in the translation of the AND gate when it has input replicator events: in this case in fact the presence of the synchronization causes the unfolding to include several instances of the input place corresponding to the replicator event, in input to the same AND gate transition instance: this is consistent with several instances of a given replicator event class being input to the same gate in the *unfold*$_{FT}$*(ParNdRFT)*. The use of such function in the case of the OR gate is not necessary because in the OR translation there is one separate transition for each input event of the gate: the convergence of several event instances towards the same gate is reflected in the different color domain of the place corresponding to the output event of the gate (place $OUTCOMP_e$ in Fig. 3.E) that has fewer parameters than the places corresponding to input replicator events (i.e. the parameters introduced by the replicator are not present in the color domain of $OUTCOMP_e$ and several instances of $OUTCOMP_{e_j}$ or $DOWN_{e_i}$ differing only in the value of the parameters not

shared with $OUTCOMP_e$ are connected through a transition to the latter place.). This completes the proof sketch.

Let us remark that the ParNdRFT analysis method proposed in this paper does not apply the unfolding of the MDWN, but rather the MDWN properties are exploited to build a lumped RG (the SRG) and, as a consequence, a smaller (lumped) MDP. The equivalence (in terms of computed optimal strategy) between the lumped MDP generated from the SRG of the MDWN and the MDP of the MDPN obtained by unfolding the MDWN has been proved in [5].

5 An example of application

Fig. 1 shows an example of Parametric NdRFT which extends the NdRFT model presented in [3] by maintaining the graph structure and introducing parameters. The model in [3] represented a system composed by three subsystems corresponding to the events $U1, U2, U3$, while the parametric version represents six subsystems, grouped in the replicator events $U1(i), U2(j), U3(k)$ (surrounded by a dashed rectangle), each representing two subsystems sharing the same repair options. According to the gates in the model, all the subsystems include two basic components (A and P) and fail if at least one of the components fails. The system fails (TE) if all the subsystems fail. The failure probability ($\downarrow$) and the repair probability ($\uparrow$) of each BE are shown in Fig. 1.

The following repair processes can be activated in the Parametric NdRFT in Fig. 1: 1) a global repair process in case of failure of subsystem $U2(j)$ involving the corresponding components $A2(j)$ and $P2(j)$; 2) a local repair process in case of the system failure (TE) involving the components $A1(i)$, $P1(i)$, $A2(j)$ and $P2(j)$, $\forall i \in T_1, j \in T_2$.

In case of global repair, one repair resource is used to repair the subsystem; in case of local repair, one resource has to be dedicated to the repair of each BE. In our case example, two repair resources are available (Fig. 1), so that only two repair processes (one global and one local repair or two local repairs) can be performed in parallel.

We will discuss some experiments performed on this example. In particular we will discuss the state space explosion problem showing how the SRG can mitigate it; moreover we will show how the TE probability decreases when increasing the redundancy and how the repair strategy may change in different situation.

Tab. 1 shows some experiments and allows to compare the complexity of the parametric model (SRG) versus the unfolded one (RG). The first column shows the model size (function of $|T_1|, |T_2|$ and $|T_3|$), the second, the third, the fourth are related to the RG approach, while the others to the SRG one. In particular the second column shows the RG size (number of states), the third column the MDP size and the fourth one the global computation time (RG + MDP). Instead the fifth column shows the SRG size, the six column the MDP size and the last one the global computation time. The computation has been performed with an INTEL Centrino DUO 2.7, 2Gb RAM.

$\|T_i\|$	RG	MDP_{RG}	Time	SRG	MDP_{SRG}	Time
1,2,3	St.	St.	RG+MDP	St.	St.	RG+MDP
Same repair policy of Fig. 1						
1,1,1	3,1E+3	389	<1s	3,1E+3	389	<1s
2,1,1	3,5E+4	937	12s	1,5E+4	579	<1s
2,2,1	4,5E+5	7.754	32m	2,2E+5	3.143	119s
2,2,2	2,9E+6	32.558	15h	7,8E+5	16.222	23m
2,2,3	8,3E+7	—	—	1E+7	52.271	13h
Without the repair processes triggered by TE						
2,2,1	3,8E+5	483	30m	1,9E+5	161	816s
2,2,2	1,7E+6	2.567	2h	6,5E+5	341	16m
2,2,3	5,9E+7	—	—	7,4E+6	401	10h
Without global repair processes triggered by $U2$						
2,2,1	2,5E+5	633	16m	1,2E+5	211	678s
2,2,2	7,5E+5	3.005	1h	2,8E+5	392	16m
2,2,3	2,1E+7	—	—	2,6E+6	575	4h
2,3,2	1,6E+8	—	—	1,2E+7	1.167	7h

Table 1. Experimental results

These results show that state space grows very fast when redundancy is increased, so that the model becomes quickly intractable. Moreover, the state space growth depends on the repair policies applied in the model as shown in Tab. 1. For instance if we remove the global repair process triggered by $U2$ we observe that the state space size decreases sensibly, allowing to solve the case $2, 3, 2$.

Comparing the two approaches, it is easy to observe how the parametric model needs less memory and time w.r.t the unfolded one. For instance, for $2, 2, 2$ the memory reduction factor is $3,89$, and the time reduction factor is ≈ 7, while for $2, 2, 3$ and $2, 3, 2$ the RG cannot be computed (its size is inferred from the SRG).

Since the non repairable components ($A3$ and $P3$) cannot induce directly the failure of the global system, we can compute the average reward and the optimal strategy of the underlying MDP at infinite horizon. Observe that defining the optimal strategy for this model is not trivial: for instance when all the basic events are down then the optimal strategy suggests us to repair $P1_i$ with a local repair action, while $A2_j, P2_j$ with a global repair action. This is justified by the fact that the global repair action of $A2_j, P2_j$ needs only one resource. Instead when $A1_*, P1_*, P2_*$ and $P3_*$ are down, it suggests to repair $P1_i$ and $P2_j$ with a local repair action. The choice to repair locally $P2_j$ is justified by the fact that in this case the probability to repair the component $(1 - P2_j.rprob)$ in one time unit is greater than that associated with the global repair action $(1 - U2_j.rprob)$.

Moreover we have computed the TE probability in steady state, solving the DTMC obtained from the MDP fixing the actions according to the optimal strategy. For instance the TE probability for cases (1,1,1), (1,2,1) and (2,2,2) are resp. 0.0151943, 0.0045737 and 0.0009536.

6 Comparing several approaches about Fault Trees with repair

We discuss here the main differences among the formalisms cited in Sec. 2 and the ParNdRFT.

Several software tools, such as *Stars Studio* and *ASTRA* [9], extend the FT formalism by allowing to model the repair of single components (BEs). Usually the user can associate with a BE, besides the failure probability, also the repair probability of the corresponding component in some form (e.g. repair time or rate). So, the behaviour of the component can be modeled with a Markov chain (MC) composed by two states: working and failed.

In *SHARPE* [14] the probability of a BE failure can be set equal to some measure computed on another kind of model designed by the user, for instance a CTMC or a GSPN. In this way, the failure and repair mode of a component may be more complex than a simple transition from the working state to the failure state and vice-versa. The *HIMAP* tool [10] allows to deal with FTs with repair, according to two approaches [2]: 1) the modeler can design a FT model and the tool automatically converts it into the equivalent MC; then, the modeler can edit the MC in order to represent and analyze the presence of repair. 2) the modeler can design a FT model where some BEs are declared as repairable. The tool allows to convert the FT *modules* [2] including the repair in MC, and to analyze them in this form.

All the approaches described so far consider the repair limited to single components. The RFT formalism [8] instead, allows to model the repair of subsystems. The repair of a subsystem is a more complex process than the repair of a single component, so in RFT, the repair is characterized by several parameters collected in a *repair policy*. They can be the event triggering the repair process, the mean time to detect the failure of the subsystem, the set of repairable components in the subsystem, the mean time to repair a single component, the number of repair facilities, the order of repair of the components, etc. The repair of a subsystem establishes several dependencies among the events in the RFT. This determines the need to analyze the model by generating its state space. The state space based analysis is typically more expensive in computational terms than the combinatorial analysis used for standards FTs, and usually performed by conversion into *Binary Decision Diagram* (BDD) [13]. According to the approach proposed in [2], the state space based analysis (performed by conversion into GSPN) can be limited to the parts of the RFT (*modules*) that contain dependencies. The rest of the model has to be solved resorting to the combinatorial analysis.

In the RFT formalism, the repair policy is not the result of the model analysis: it is pre-defined by the modeler and is associated with the RB node before that the model is analyzed. In the NdRFT formalism [3] instead, the repair policy (or strategy) is the result of the model analysis. The NdRFT formalism allows to express several possible start repair options based on: 1) the concept of "observability" of events (repair actions can only be triggered by observable failures), 2) the notion of local versus global repair action, 3) the notion of repair supervisor component, in case of global repair, 4) the notion of resource requirements for each type of repair action. Very few restrictions are imposed on the scope of repair actions (so that the repair of each basic component can start based on observations made on different failure events). Given this information, the analysis of the NdRFT models returns the optimal repair strategy. This is done by generating a MDP from the NdRFT, through an intermediate translation of the NdRFT model into a *Markov Decision Petri Net* (MDPN) [5]: this allows to reuse the efficient algorithms devised to derive an MDP from an MDPN.

The ParNdRFT can be useful when the system has a high level of redundancy of the critical components or subsystems: the parts of the model characterized by the same structure can be 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. As shown in Sec. 5 this has a relevant impact on the complexity of the analysis since symmetries can reduce dramatically the state space size and then the MDP derived from it.

The possibility of decomposing the analysis of a (Par)NdRFT by exploiting modules have not yet been pursued. Actually the possibility to specify several repair options and in particular to share repair resources among different repair actions, introduces strong dependencies among the events that cause state changes in the model. Therefore its not so frequent that independent subtrees (modules) are present in the model. Anyway a subtree sharing no events with other subtrees can be a module in a (Parametric) NdRFT model, in (at least) two particular situations: 1) the subtree contains no repairable components; 2) the subtree includes a global repair process and no other repair options. In these cases, such modules can be solved in isolation with the proper technique: combinatorial or state space analysis respectively. Then, they can be replaced by a BE with a properly computed failure and repair probability: the MDP can then be generated from the simplified (Par)NdRFT model.

7 Conclusion and future work

We have presented an extension to FT that allows to model failure modes of complex systems as well as their repair processes in parametric form. With respect to other existing approaches, this formalism allows to consider repair of whole subsystems rather than repair of BE, with a

variety of repair start options and taking into account repair resources requirements and resource limitations. Moreover it allows to face repair strategy optimization problems rather than evaluating a strategy provided by the modeler. This is done by defining the ParNdRFT semantics in terms of an MDWN and then deriving an MDP from the SRG of the MDWN and by using the techniques available for MDPs.

The originality of this formalism with respect to the NdRFT is that it allows to exploit the presence of redundancy to reduce the complexity of the model and of the analysis. It is based on the translation of the parametric NdRFT into a MDWN, so that SRG technique can be used to produce a reduced MDP w.r.t. the original one, on which the analysis may be performed more efficiently.

A foreseeable future work is extending the ParNdRFT, so that the modeler can directly define more complex reward functions, for instance considering the cost of repair actions, or the penalties due to the fact that the system is in a degraded state (the system is up, but some subsystem is down, e.g. corresponding to a system with degraded performance): this requires to extend the formalism to specify the function to be optimized, and the translation to derive the corresponding MDP reward function. Another possible extension of ParNdRFT could be to consider *dynamic gates* [11], which allow to express functional and temporal dependencies among component failures, as well as repair resources preemption.

Acknowledgements: the activity of M. Beccuti, D. Codetta-Raiteri and G. Franceschinis has been partially supported by the EU-Project CRUTIAL IST-2004-27513. The activity of all the authors has been partially supported by the bilateral project Galileo.

References

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

[2] A. Anand and A. K. Somani. Hierarchical Analysis of Fault Trees with Dependencies, Using Decomposition. In *Annual Reliability and Maintainability Symposium*, pages 69–75, 1998.

[3] M. Beccuti, D. Codetta-Raiteri, G. Franceschinis, and S. Haddad. Non deterministic repairable fault trees for computing optimal repair strategy. In *Int. Conf. on Performance Evaluation, Methodologies and Tools*, Athens, Greece, October 2008.

[4] M. Beccuti, G. Franceschinis, D. Codetta-Raiteri, and S. Haddad. Non deterministic Repairable Fault Trees for computing optimal repair strategy. Dip. Informatica, Univ. Piemonte Orientale. Tech. Rep. TR-INF-2008-07-05, 2008.

[5] M. Beccuti, G. Franceschinis, and S. Haddad. Markov Decision Petri Net and Markov Decision Well-Formed Net Formalisms. *Lecture Notes in Computer Science*, 4546:43–62, 2007.

[6] 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.

[7] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed coloured nets for symmetric modelling applications. *IEEE Transactions on Computers*, 42(11):1343–1360, nov 1993.

[8] D. Codetta-Raiteri, G. Franceschinis, M. Iacono, and V. Vittorini. Repairable Fault Tree for the automatic evaluation of repair policies. In *Int. Conf. on Dependable Systems and Networks*, pages 659–668, Florence, Italy, June 2004.

[9] S. Contini. *ASTRA - Advanced Software Tool for Reliability Analysis, Theoretical Manual*. European Commission Joint Research Centre (JRC), Ispra, Italy, http://www.jrc.ec.europa.eu, 1999. EUR 18727en.

[10] G. Krishnamurthi, A. Gupta, and A. K. Somani. HIMAP: Architecture, Features, and Hierarchical Model Specification Techniques. *Lecture Notes in Computer Science*, 1469:348–351, 1998.

[11] R. Manian, D.W. Coppit, K.J. Sullivan, and J.B. Dugan. Bridging the Gap Between Systems and Dynamic Fault Tree Models. In *Annual Reliability and Maintainability Symposium*, pages 105–111, 1999.

[12] M. Puterman. *Markov decision processes : Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.

[13] A. Rauzy. New Algorithms for Fault Trees Analysis. *Reliability Engineering and System Safety*, 05(59):203–211, 1993.

[14] 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, 1996.

[15] W.G. Schneeweiss. *The Fault Tree Method*. LiLoLe Verlag, 1999.