A Compositional Semantics for Dynamic Fault Trees in Terms of Interactive Markov Chains*

Hichem Boudali[1], Pepijn Crouzen[2,**], and Mariëlle Stoelinga[1]

[1] Department of Computer Science, University of Twente,
P.O. Box 217, 7500AE Enschede, The Netherlands
[2] Saarland University, Department of Computer Science,
D-66123 Saarbrücken, Germany
{hboudali@cs,p.crouzen@alumnus,marielle@cs}.utwente.nl

Abstract. Dynamic fault trees (DFTs) are a versatile and common formalism to model and analyze the reliability of computer-based systems. This paper presents a formal semantics of DFTs in terms of input/output interactive Markov chains (I/O-IMCs), which extend continuous-time Markov chains with discrete input, output and internal actions. This semantics provides a rigorous basis for the analysis of DFTs. Our semantics is fully compositional, that is, the semantics of a DFT is expressed in terms of the semantics of its elements (i.e. basic events and gates). This enables an efficient analysis of DFTs through compositional aggregation, which helps to alleviate the state-space explosion problem by incrementally building the DFT state space. We have implemented our methodology by developing a tool, and showed, through four case studies, the feasibility of our approach and its effectiveness in reducing the state space to be analyzed.

Fault trees (FTs) [20], also called static FTs, provide a high-level, graphical formalism to model and analyze system failures. An FT is made up of basic events, usually modeling the failure of physical components, and of logical gates, such as AND and OR gates, modeling how the component failures induce the system failure. Dynamic fault trees (DFTs) [11] extend (static) fault trees by allowing the modeling of more complex behaviors and interactions between components: Whereas FTs only take into consideration the combination of failures, DFTs also take into account the order in which they occur. A DFT is typically analyzed by first converting it to a continuous time Markov chain (CTMC) and by then analyzing this CTMC.

This paper formally describes the DFT syntax and semantics, thus providing a rigorous basis for DFT analysis and tool development. The DFT syntax is given in terms of a directed acyclic graph (DAG) and its semantics in terms

* This research has been partially funded by the Netherlands Organisation for Scientific Research (NWO) under FOCUS/BRICKS grant number 642.000.505 (MOQS); the EU under grant number IST-004527 (ARTIST2); and by the DFG/NWO bilateral cooperation programme under project number DN 62-600 (VOSS2).

** The majority of this work was done while the author was at the University of Twente.

K.S. Namjoshi et al. (Eds.): ATVA 2007, LNCS 4762, pp. 441–456, 2007.

of an input/output interactive Markov chain (I/O-IMC). Our semantics is fully compositional. That is, we present the semantics of each DFT element (i.e. gate or basic event) as an I/O-IMC; the semantics of a DFT is then obtained by parallel composing the I/O-IMC semantics of all its elements. Compositionality is a fundamental and highly desirable property of a semantics: it enables compositional reasoning, i.e. analyzing complex systems by breaking them down into their constituting parts. In our case, it enables compositional aggregation to combat the state-space explosion problem existing in DFTs. Moreover, a compositional semantics is comprehensible, since one can focus on one construct at a time, and readily extensible. As elaborated in [4], we can easily add new DFT gates or concepts such as repair policies.

Earlier work on formalizing DFTs can be found in [10], where a semantics is described using the Z specification language. This work revealed a number of ambiguities in the DFT framework. Most notably, in some instances of DFTs non-determinism has arisen. But, since non-determinism was not intended in the original formulation of DFTs and every DFT had to be mapped into a CTMC, this non-determinism was resolved by transformation into a deterministic or probabilistic choice (see [4] for further details on non-determinism in DFTs). The semantics in [10] is, however, not compositional and hence is, in our opinion, not easy to understand. Formalizing the DFT syntax and semantics in a compositional way turned out to be a non-trivial task.

Interactive Markov chains (IMCs) [14] are an extension of CTMCs with discrete actions and have proven to be a powerful formalism for a variety of applications. IMCs come with efficient algorithms for aggregating equivalent (i.e. weakly bisimilar) states and operators for parallel composing IMCs and for hiding (i.e. making internal) certain discrete actions. For our purposes, we needed IMCs that distinguish between input and output actions. Hence, we introduce I/O-IMCs, combining IMCs with features from the I/O automaton model in [17]. We also present a notion of weak bisimilarity for I/O-IMCs. To aggregate as many states as possible, our weak bisimulation disregards Markovian transitions from a state s into its own equivalence class. Thus, we do not only generalize the usual IMC weak bisimulation to I/O-IMCs, but also extend it along the lines of [8]. Furthermore, we show that weak bisimulation is a congruence w.r.t. parallel composition and hiding.

The conversion into a CTMC and resolution of a DFT has been first introduced by Dugan et al. in the so called DIFTree methodology [12]. This method suffers from the well-known state-space explosion problem. In fact, the size of the CTMC grows exponentially with the number of basic events in the DFT. Recently, there have been attempts in dealing with the state-space explosion problem by avoiding the CTMC generation [6,1]. In [6], the authors propose a Bayesian network approach and provide an approximate DFT solution. In [1], the authors present a method to identify submodules in a DFT where the CTMC generation is not needed.

We use a compositional aggregation approach to build the I/O-IMC of the whole DFT: We start with the interpretation of a single DFT element as an

I/O-IMC. Then we repeatedly take the parallel composition with the interpretation of another element, while aggregating equivalent states. We keep repeating these two steps until we are left with a single aggregated I/O-IMC. This compositional aggregation approach is crucial in alleviating the state-space explosion problem. To summarize, this paper makes the following contributions:

1. We derive, based on IMCs and I/O automata, the I/O-IMC formalism and introduce a notion of weak bisimilarity for I/O-IMCs, which we show to be a congruence w.r.t. parallel composition and hiding.
2. We formally define the DFT syntax and semantics in terms of respectively a DAG description and I/O-IMCs.
3. We report on a tool and show the feasibility of our approach on four case studies.

The remainder of the paper is organized as follows: Section 1 introduces DFTs and Section 2 treats the formalism of I/O-IMCs. In Section 3, we present the syntax and the semantics of DFTs, and in Section 4 we illustrate the compositional aggregation technique. Finally, Section 5 provides some case studies and presents the prototype tool, and we conclude the paper in Section 6.

1 Dynamic Fault Trees

An FT is a tree (or rather, a DAG) in which the leaves are called *basic events* (BEs) and the other elements are *gates*. BEs model the failure of physical components and are depicted by circles. The failure of a BE is governed by an exponential distribution. That is, the probability that the BE fails within t time units equals $1 - e^{-\lambda t}$, where λ is the *failure rate* of the BE. The non-leaf elements are gates, modeling how the component failures induce a system failure. Static fault trees have three type of (static) gates: the AND gate, the OR gate and the K/M (or called VOTING) gate, depicted in Figure 1.a, 1.b, and 1.c, respectively. These gates fail if respectively all, at least one, or at least K (called the threshold) out of M of their inputs fail.

Dynamic fault trees [11] extend (static) FTs with three novel types of gates[1]: The priority AND gate (PAND); the spare gate (SPARE), modeling the management and allocation of spare components; and the functional dependency gate (FDEP). These gates (depicted in Figure 1.d, 1.e, and 1.f) are described below.

PAND Gate. The priority AND (PAND) gate models a failure sequence dependency. It fails if all of its inputs fail from left to right order in the gate's depiction. If the inputs fail in a different order, the gate does not fail.

SPARE Gate. The SPARE gate has one primary input and zero (which is a degenerated case) or more alternate inputs called *spares*. All inputs are BEs.

[1] A fourth gate called 'Sequence Enforcing' (SEQ) gate has also been defined in [11], but it turns out that this gate is expressible in terms of the cold spare gate.

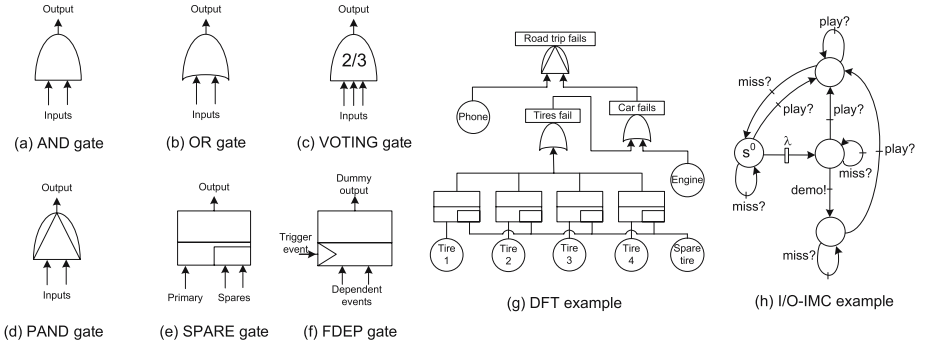

Fig. 1. DFT gates, DFT example, and I/O-IMC example

The primary input of a SPARE gate is initially powered on and the alternate inputs are in standby mode. When the primary fails, it is replaced by the first available alternate input (which then switches from the standby mode to the active mode). In turn, when this alternate input fails, it is replaced by the next available alternate input, etc.

In standby (or dormant) mode, the BE failure rate λ is reduced by a *dormancy factor* $\alpha \in [0, 1]$. Thus, the BE failure rate in standby mode is $\mu = \alpha\lambda$. In active mode, the failure rate switches back to λ. Two special cases arise if $\alpha = 0$ or $\alpha = 1$. If $\alpha = 0$, the spare is called a *cold spare* and can by definition not fail before the primary. When $\alpha = 1$, the spare is called a *hot spare* and its failure rate is the same whether in standby or in active mode. If $0 < \alpha < 1$, the spare is called a *warm spare*. The SPARE gate fails when the primary and all its spares have failed.

Multiple spare gates can share a pool of spares. When the primary unit of any of the spare gates fails, it is replaced by the first available (i.e. not failed or not already taken by another spare gate) spare unit; which becomes, in turn, the active unit for that spare gate.

FDEP Gate. The functional dependency gate consists of a trigger event (i.e. a failure) and a set of dependent events (or components). When the trigger event occurs, it causes all the dependent components to become inaccessible or unusable (the dependent components can of course also still fail by themselves). Dependent events need to be BE's. All dependent events and the trigger event are considered to be inputs to the FDEP gate. The FDEP gate's output is a 'dummy' output (i.e. it is not taken into account during the calculation of the system failure probability).

Example 1. Figure 1.g shows a DFT modeling a road trip. Looking at the top PAND gate, we see that the road trip fails (i.e. we are stuck on the road) if the car fails after the mobile phone has failed; if the car fails first, then we can call the road services to tow the car and continue our journey. The car fails if either the engine fails or the tire subsystem fails, as modeled by the OR gate

labeled 'car fails'. The car is equipped with a spare tire, which can be used to replace any of the primary tires. When a second tire fails, the tire subsystem fails, causing in turn a car failure. Thus, we model the tire subsystem by four spare gates, each having a primary tire (BEs 'Tire 1', 'Tire 2','Tire 3', and 'Tire 4') and all sharing a spare tire (BE 'Spare tire'). The spare tire is a cold spare, i.e. it is initially in standby mode with failure rate 0.

2 Input/Output Interactive Markov Chains

The formalism. This section introduces the formalism of input/output interactive Markov chains (I/O-IMCs), which are based on IMCs [14]. IMCs combine continuous-time Markov chains with discrete actions (also called signals). State changes in IMCs can occur either because a discrete action is taken, or after a delay, which is governed by an exponential distribution. Thus, IMCs have two types of transitions: discrete transitions (denoted $\xrightarrow{a}$ and $\dashrightarrow$ in figures) labeled with a discrete action a and Markovian transitions (denoted $\xrightarrow{\lambda}{}^{M}$ and $\dashrightarrow\!\!\square$ in figures) labeled with the rate λ of an exponential distribution.

I/O-IMCs are a variant of IMCs that partition the set of discrete actions into input actions, output actions and internal actions (inspired by the I/O variant of automata introduced in [17]). Input actions, being under the control of the environment of the I/O-IMC, are delayable, while output actions must be taken immediately and cannot be delayed. This partition is natural in the DFT context where elements have input and output signals and where – rather than being a handshake – communication is always initiated by the failing (or activating) component. Moreover, in contrast with IMCs where all observable actions are delayable, in I/O-IMCs only input actions are delayable. Internal actions are not visible to the environment and are also immediate.

Definition 1. *An* input/output interactive Markov chain $\mathcal{P}$ *is a tuple* $\langle S, s^0, A, \rightarrow, \rightarrow^M \rangle$*, where*

- S *is a set of states,*
- $s^0 \in S$ *is the initial state.*
- A *is a set of discrete actions (or signals), where* $A = (A^I, A^O, A^{int})$ *is partitioned into a set of input actions* A^I*, output actions* A^O *and internal actions* A^{int}*. We write* $A^V = A^I \cup A^O$ *for the set of visible actions of* $\mathcal{P}$*. We suffix input actions with a question mark (e.g. a?), output actions with an exclamation mark (e.g. a!) and internal actions with a semi-colon (e.g. a;).*
- $\rightarrow \subseteq S \times A \times S$ *is a set of interactive transitions. We write* $s \xrightarrow{a} s'$ *for* $(s, a, s') \in \rightarrow$*. We require that I/O-IMCs are input-enabled:* $\forall s \in S, a? \in A^I, \exists s' \in S \cdot s \xrightarrow{a?} s'$*.*
- $\rightarrow^M \subseteq S \times \mathbb{R}_{>0} \times S$ *is a set of Markovian transitions. We write* $s \xrightarrow{\lambda}{}^{M} s'$ *for* $(s, \lambda, s') \in \rightarrow^M$*.*

We denote the components of $\mathcal{P}$ *by* $S_\mathcal{P}$*,* $s^0_\mathcal{P}$*,* $A_\mathcal{P}$*,* $\rightarrow_\mathcal{P}$*,* $\rightarrow^M_\mathcal{P}$ *and omit the subscript* $\mathcal{P}$ *whenever clear from the context. The* action signature *of an I/O-IMC is the partitioning* (A^I, A^O, A^{int}) *of* A*. We denote the class of all I/O-IMCs by* IOIMC*.*

Example 2. Figure 1.h shows an example I/O-IMC modeling a video game. If a user presses the play button (input action *play*?), the game starts and continues until the user makes a mistake (*miss*?), which brings the game back to the initial state s^0. If after some delay d, no *play*? signal has been received, then the system runs a game demonstration (output *demo*!) until a *play*? signal is received. The delay d (in hours) is exponentially distributed with rate 12. This means that, on average, a demo is played after $\frac{1}{12}$ hours (= 5 minutes).

Note that the system is input enabled, i.e. each state enables the input actions *play*? and *miss*?.

I/O-IMCs can be built from smaller I/O-IMCs through parallel composition. If two I/O-IMCs $\mathcal{P}$ and $\mathcal{Q}$ are composable, then their composition $\mathcal{P}\|\mathcal{Q}$ is the I/O-IMC representing their joint behavior. As in the I/O automaton framework, the components $\mathcal{P}$ and $\mathcal{Q}$ synchronize on shared actions and evolve independently on actions that are internal or not shared. The hiding operator *hide* B *in* $\mathcal{P}$ makes all actions in a set B of visible actions internal.

Definition 2. *Let $\mathcal{P}$ and $\mathcal{Q}$ be I/O-IMCs.*

1. *$\mathcal{P}$ and $\mathcal{Q}$ are* composable *if $A^O_\mathcal{P} \cap A^O_\mathcal{Q} = A^{int}_\mathcal{P} \cap A_\mathcal{Q} = A_\mathcal{P} \cap A^{int}_\mathcal{Q} = \emptyset$.*
2. *If $\mathcal{P}$ and $\mathcal{Q}$ are composable I/O-IMCs, their* composition *$\mathcal{P}\|\mathcal{Q}$ is the I/O-IMC $(S_\mathcal{P} \times S_\mathcal{Q}, (s^0_\mathcal{P}, s^0_\mathcal{Q}), ((A^I_P \cup A^I_Q) \setminus (A^O_P \cup A^O_Q), (A^O_P \cup A^O_Q), (A^{int}_P \cup A^{int}_Q))$, $\rightarrow_{\mathcal{P}\|\mathcal{Q}}, \rightarrow^M_{\mathcal{P}\|\mathcal{Q}})$, where*

$$\begin{aligned}
\rightarrow_{\mathcal{P}\|\mathcal{Q}} &= \{(s,t) \xrightarrow{a}_{\mathcal{P}\|\mathcal{Q}} (s',t) \mid s \xrightarrow{a}_\mathcal{P} s' \wedge a \in A_\mathcal{P} \setminus A_\mathcal{Q}\} \\
&\quad \cup \{(s,t) \xrightarrow{a}_{\mathcal{P}\|\mathcal{Q}} (s,t') \mid t \xrightarrow{a}_\mathcal{Q} t' \wedge a \in A_\mathcal{Q} \setminus A_\mathcal{P}\} \\
&\quad \cup \{(s,t) \xrightarrow{a}_{\mathcal{P}\|\mathcal{Q}} (s',t') \mid s \xrightarrow{a}_\mathcal{P} s' \wedge t \xrightarrow{a}_\mathcal{Q} t' \wedge a \in A_\mathcal{P} \cap A_\mathcal{Q}\} \\
\rightarrow^M_{\mathcal{P}\|\mathcal{Q}} &= \{(s,t) \xrightarrow{\lambda}{}^M (s',t) \mid s \xrightarrow{\lambda}{}^M_\mathcal{P} s'\} \cup \{(s,t) \xrightarrow{\lambda}{}^M (s,t') \mid t \xrightarrow{\lambda}{}^M_\mathcal{Q} t'\}
\end{aligned}$$

   Given a set $\{\mathcal{P}_1, \mathcal{P}_2 \ldots \mathcal{P}_n\}$ of I/O-IMCs, we write $\|\mathcal{P}_i$ for $\mathcal{P}_1\|\mathcal{P}_2\| \cdots \mathcal{P}_n$.
3. *Let $B \subseteq A^V$ be a set of visible actions. We define the I/O-IMC hide B in $\mathcal{P}$ by $(S_\mathcal{P}, s^0_\mathcal{P}, (A^I_\mathcal{P} \setminus B, A^O_\mathcal{P} \setminus B, A^{int}_\mathcal{P} \cup B), \rightarrow_\mathcal{P}, \rightarrow^M_\mathcal{P})$.*

Bisimilarity. Bisimilation relations are equivalences on the state-space that identify states with the same step-wise behavior. Our notion of weak bisimilarity is based on bisimulation for IMCs [14]. The key differences are we distinguish between input and output transitions, and ignore Markovian self-loops (as in [8]); i.e. Markovian transition from a state in an equivalence class to a state in the same equivalence class.

Let $\mathcal{P}$ be an I/O-IMC and let $s, s' \in S$ be states in $\mathcal{P}$. We write $s \xRightarrow{\varepsilon} s'$ if there exists a sequence (possibly of length zero, i.e. $s = s'$) $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \ldots \xrightarrow{a_n} s_n = s'$ of transitions with $a_i \in A^{int}$. For $a \in A^V$ we write $s \xRightarrow{a} s'$ if there exists states s_1, s_2 such that $s \xRightarrow{\varepsilon} s_1 \xrightarrow{a} s_2 \xRightarrow{\varepsilon} s'$. Also, $\gamma_M(s, C) = \sum\{|\lambda \mid s \xrightarrow{\lambda}{}^M s' \wedge s' \in C|\}$, with $\{|\ldots|\}$ denoting a multiset of transition rates, is the sum of the rates of all Markovian transitions from s into C. We write $C^{int} = \{s \mid \exists s' \in C \cdot s \xRightarrow{\varepsilon} s'\}$ for the set of all states which can reach some element in C via internal transitions.

Finally, we say that s is *stable* if s has no outgoing immediate (i.e. internal or output) transition.

Definition 3 (Weak bisimulation). *Let P be an I/O-IMC. A* weak bisimulation *for P is an equivalence relation R on S such that for all $(s,t) \in R$, $a \in A \cup \{\varepsilon\}$*

1. *$s \stackrel{a}{\Longrightarrow} s'$ implies that there is a weak transition $t \stackrel{a}{\Longrightarrow} t'$ with $(s',t') \in R$.*
2. *$s \stackrel{\varepsilon}{\Longrightarrow} s'$ and s' stable imply that there is a t' such that $t \stackrel{\varepsilon}{\Longrightarrow} t'$ and t' stable and $\gamma_M(s', C^{int}) = \gamma_M(t', C^{int})$, for all equivalence classes C of R, except for $C = [s']_R$, the equivalence class of s'.*

States s and t in P are weakly bisimilar*, notation $s \approx t$, if there exists a weak bisimulation R with $(s,t) \in R$.*

Our notion of weak bisimilarity satisfies the usual properties: $\approx$ is the largest weak bisimulation and it is a congruence with respect to parallel composition and hiding. To compute $\approx$, one can use an algorithm similar to the one in [14], which runs in time $O(n^3)$, where n is the number of states in the I/O-IMC. We refer the reader to [5] for more details.

As most bisimulation relations, $\approx$ can be used to aggregate (also referred to as lump, minimize or reduce) an I/O-IMC $\mathcal{P}$: By grouping together equivalent (i.e. weakly bisimilar) states in $\mathcal{P}$, we obtain an equivalent I/O-IMC that is (usually) smaller. The mentioned properties enable an efficient aggregation algorithm that works in a compositional way, cf. Section 4.

3 Formalizing DFTs

3.1 DFT Syntax

To formalize the syntax of a DFT, we first define the set $\mathcal{E}$, characterizing each DFT element by its type, number of inputs and possibly some other parameters. We use the following notation. Given a set X, we denote by $\mathcal{P}(X)$ the power set over X and by X^* the set of all sequences over X. For a sequence $x \in X^*$, we denote by $|x|$ the length of the sequence (also called list), and by $(x)_i$ the i^{th} element in x.

Definition 4. *The set $\mathcal{E}$ of DFT elements consists of the following tuples. Here, $k, n \in \mathbb{N}$ are natural numbers with $1 \leq k \leq n$ and $\lambda, \mu \in \mathbb{R}^{\geq 0}$ are rates.*

- *(OR, n), (AND, n), $(PAND, n)$ represent respectively OR, AND and PAND gates with n inputs.*
- *(VOT, n, k) represent a voting gate with n inputs and threshold k.*
- *$(SPARE, n)$ represent a SPARE gate with one primary and $n-1$ spares. By convention, the first input to the SPARE gate is the primary component.*
- *$(FDEP, n)$ represents an FDEP gate with 1 trigger input event and $n-1$ dependent input events. By convention, the first input to the FDEP gate is the trigger event.*

– $(BE, 0, \lambda, \mu)$, *represents BE, which has no inputs (i.e.* $n = 0$*), an active failure rate* λ *and a dormant failure rate* μ.

Given a tuple $e \in \mathcal{E}$*, we write* $type(e)$ *for the first item in* e*, and* $arity(e)$ *for the second.*

A DFT is a directed acyclic graph, where each vertex v is labeled with a DFT element $l(v) \in \mathcal{E}$. An edge from v to w means that the output of $l(v)$ is an input to $l(w)$. Since the order of inputs to a gate matters (e.g. for a PAND gate), the inputs to v are given as a list $preds(v)$, rather than as a set.

Definition 5. *A dynamic fault tree is a quadruple* $\mathcal{D} = (V, preds, l)$*, where*

– V *is a set of vertices,*
– $l : V \to \mathcal{E}$ *is a labeling function, that assigns to each vertex a DFT element.*
– $preds : V \to V^*$ *is a function that assigns to each vertex a list of inputs.*

The set of *edges* E is the set $\{(v, w) \in V^2 | \exists i \,.\, v = (preds(w))_i\}$ of all pairs (v, w) such that v appears as a predecessor of w. We write $type(v)$ for $type(l(v))$ and $arity(v)$ for $arity(l(v))$. For $\mathcal{D}$ to be a well-formed DFT, the following restrictions have to be met.

– The set (V, E) is a directed acyclic graph.
– All inputs to a DFT element must be connected to some node in $\mathcal{D}$, i.e. for all $v \in V$, we have $arity(v) = |preds(v)|$.
– Since we do not include the dummy output of an FDEP gate in $\mathcal{D}$, FDEP gates have no outgoing edges: if $(v, w) \in E$, then $type(v) \neq FDEP$.
– There is a unique top element in $\mathcal{D}$, i.e. a non-FDEP element whose output is not connected. That is, there exists a unique $v \in V$, $type(v) \neq FDEP$ such that for all $w \in V \,.\, (v, w) \notin V$. This unique v is denoted by $T_{\mathcal{D}}$; or by T if $\mathcal{D}$ is clear from the context.
– The first input of a SPARE gate can not be an input to another SPARE gate (i.e. primary components can not be shared): If $v = (preds(w))_1 = (preds(w'))_1$ with $type(w) = type(w') = SPARE$, then $w = w'$.
– Inputs (primary and spare components) of a SPARE gate must be BEs: if $type(w) = SPARE$, then $type((preds(w))_i) = BE$, for all $1 \leq i \leq |preds(w)|$.
– The dependent inputs (i.e. inputs number 2 and higher) of an FDEP gate must be BEs: if $type(w) = FDEP$, then $type((preds(w))_i) = BE$, for all $2 \leq i \leq |preds(w))|$.
– An output can not be twice or more the input of the same gate: for all $1 \leq i, j \leq |preds(w)|$ with $(preds(w))_i = (preds(w))_j$, we have $i = j$.

3.2 DFT Element Semantics

This section provides the I/O-IMC semantics $[\![e]\!]_{\mathsf{ELT}}$ for each DFT element $e \in \mathcal{E}$. The I/O-IMC is parametric in its input and output signals. (These parameters are instantiated in Section 3.3, so that output signals in the semantics of a child element correspond to input signals in the semantics of its parents.) Thus,

formally, $[\![e]\!]_{\mathsf{ELT}}$ is a function that, depending on the type of e, takes as arguments a number of actions and returns an I/O-IMC. Each of these I/O-IMCs has an initial operational state, some intermediate operational states, a *firing* (or failed) state, and an absorbing *fired* state. The firing and fired states are drawn as gray circles and double circles respectively. For the sake of clarity, all self-loops $(s, a?, s)$ labeled by input actions are omitted from the figures.

Basic Events I/O-IMC Model. As pointed out in Section 1, a BE has a different failing behavior depending on its dormancy factor. Figure 2 shows the (parameterized) I/O-IMCs associated to a cold, warm, and hot BE[2], i.e. it shows the functions $[\![(BE, 0, \lambda, \mu)]\!]_{\mathsf{ELT}} : A^2 \rightarrow \mathsf{IOIMC}$ taking as arguments an activation signal $a?$ and a firing signal $f!$.

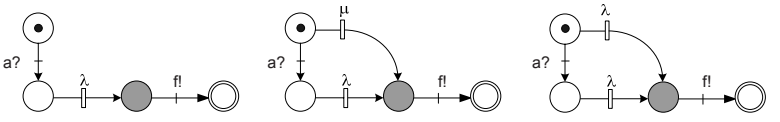

Fig. 2. The I/O-IMCs $[\![(BE, 0, \lambda, 0)]\!]_{\mathsf{ELT}}(a, f)$, $[\![(BE, 0, \lambda, \mu)]\!]_{\mathsf{ELT}}(a, f)$, and $[\![(BE, 0, \lambda, \lambda)]\!]_{\mathsf{ELT}}(a, f)$, modeling the semantics of a cold, warm and hot BE

AND Gate I/O-IMC Model. Figure 3.a shows the semantics of the $(AND, 2)$ gate, i.e. the function $[\![(AND, 2)]\!]_{\mathsf{ELT}} : A^3 \rightarrow \mathsf{IOIMC}$, taking as arguments the output and two inputs signals of the AND gate.[3] This I/O-IMC models that the AND gate fires (action f_1) after it receives firing signals from both its inputs (actions f_2 and f_3). Note that the AND gate does not have an activation signal as this element does not exhibit a dormant or active behavior as such. The semantics of the OR and VOTING gates are similar.

PAND Gate I/O-IMC Model. Figure 3.b shows the semantics $[\![(PAND, 2]\!]_{\mathsf{ELT}}$: $A^3 \rightarrow \mathsf{IOIMC}$ of a PAND gate with two inputs. The PAND gate fires after all its inputs fire from left to right order. If the inputs fire in the wrong order, the PAND gate moves to an operational absorbing state (denoted with an **X** in Figure 3.b).

FDEP Gate I/O-IMC Model. An FDEP gate does not have semantics itself, but instead is used in combination with the semantics of its dependent BEs. To model a functional dependency, we define the *firing auxiliary* function $FA : A^2 \times \mathcal{P}(A) \rightarrow \mathsf{IOIMC}$. This (parametric) I/O-IMC ensures that a dependent BE fires either when the BE fails by itself, or when its failure is triggered by the FDEP gate trigger: Figure 3.c shows the FA to be applied in combination with a BE that is functionally dependent on n triggers. Signal f_2 corresponds to the failure of the dependent event by itself; signals $f_3, f_4, \ldots, f_{n+2}$ correspond to the

[2] The hot BE I/O-IMC can be reduced to: $\odot \xrightarrow{\lambda} \bigcirc \xrightarrow{f!} \bigcirc$.

[3] The semantics $[\![(AND, n)]\!]_{\mathsf{ELT}} : A^{n+1} \rightarrow \mathsf{IOIMC}$ of the AND gate with n inputs can be constructed in a similar fashion, cf [5].

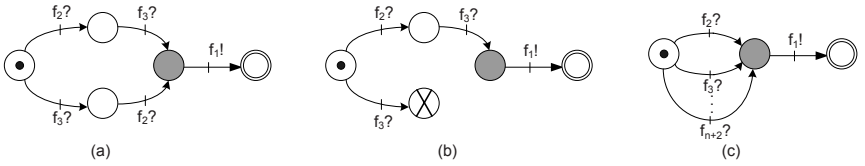

Fig. 3. The semantics (a) $[\![(AND, 2)]\!]_{\mathsf{ELT}}(f_1, f_2, f_3)$, (b) $[\![(PAND, 2)]\!]_{\mathsf{ELT}}(f_1, f_2, f_3)$, and (c) $FA(f_1, f_2, \{f_3, f_4 \ldots, f_{n+2}\})$

failures of any of the triggers; and f_1 corresponds to the failure of the dependent event when also considering its functional dependency upon the triggers. Hence, f_1 is emitted as soon as any signal from $\{f_2, f_3, \ldots, f_{n+2}\}$ occurs. Thus, the *FA* takes as arguments two firing signals and a set of firing signals (corresponding to all triggers of the dependent BE).

SPARE Gate I/O-IMC Model. Figure 4 shows the I/O-IMC of a spare gate sharing a spare with another spare gate. The SPARE gate behaves, to a certain extent, similarly to the AND gate. That is, for the spare gate to fail, both its primary has to fail and its spare has to be unavailable (fail or be taken by the other spare gate). The state reached after the primary fails is of particular interest (i.e. the state reached from the initial state after transition $f_2?$ is taken). In this state, a non-deterministic situation arises where the spare can be activated by either of the spare gates (signals $a_{1,1}!$ and $a_{1,2}?$). We could of course also get signal $f_3?$ (i.e. failure of the spare) immediately after signal $f_2?$. The signals $a_{1,1}$ and $a_{1,2}$ are signals between the two spare gates notifying each other about the activation (and thus the acquisition) of the shared spare. These signals are also sent to the spare to activate it. The semantics of a spare gate having n spares is a function $A^2 \times (A^2 \times \mathcal{P}(A))^n \rightarrow \mathsf{IOIMC}$ that takes as arguments the firing signal of the spare gate, the firing signal of its primary and n spare-tuples containing, for each spare, its firing signal, its activation signal (by the spare gate) and a set of activation signals of the other spare gates sharing that spare. Figure 4.b shows the semantics $[\![(SPARE, 2)]\!]_{\mathsf{ELT}}(f_1, f_2, (f_3, a_{1,1}, \{a_{1,2}\}))$ of the spare gate in Figure 4.a. Generalizing the SPARE gate I/O-IMC model to handle the case

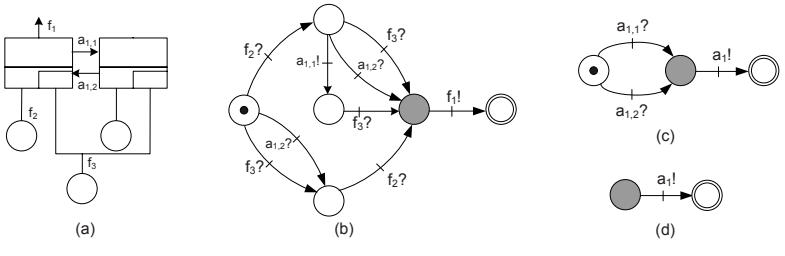

Fig. 4. (a) A DFT, (b) semantics $[\![(SPARE, 2)]\!]_{\mathsf{ELT}}(f_1, f_2, (f_3, a_{1,1}, \{a_{1,2}\}))$ of (left) SPARE gate, (c) $AA(a_1, \{a_{1,1}, a_{1,2}\})$, (d) $AA(a_1, \emptyset)$

where multiple spare gates share multiple spares turned out to be a non-trivial task [4,5].

3.3 DFT Semantics

This section shows how to get the semantics of a DFT from the semantics of its elements. First, we define the node semantics $[\![v]\!]$ of a DFT node $v \in V$ by instantiating the parameters of $[\![l(v)]\!]_{\mathsf{ELT}}$ appropriately, using the following main actions: The firing signal f_X of element $X \in \mathcal{E}$ denotes the failure of X and the activation signal a_X denotes the activation of a BE X, i.e. the switching from dormant to active mode. When used as a spare, a BE is activated by its SPARE gates; and $a_{S,G}$ denotes the activation of spare S by SPARE gate G. Otherwise, the BE is activated from the start.

Given a node v in a DFT $\mathcal{D}$, we define the node semantics $[\![v]\!]$ as follows.

OR, AND, VOT, and PAND. If v is labeled as an OR, AND, VOT, or PAND gate, then $[\![v]\!]$ is obtained from $[\![l(v)]\!]_{\mathsf{ELT}}$ by instantiating its parameters in such a way that the input signals of v connect to output signals of v's children (i.e. nodes $w \in \mathit{preds}(v)$). Thus, for $\mathit{type}(v) = \mathit{OR}$, AND, VOT, PAND, with $\mathit{preds}(v) = w_1 w_2 \dots w_n$, we have

$$[\![v]\!] = [\![l(v)]\!]_{\mathsf{ELT}}(f_v, f_{w_1}, f_{w_2}, \dots f_{w_n})$$

BE. If v is labeled as a basic event, two steps need to be carried out. First, we have to check if the BE is a dependent event of some FDEP gate. If so, we use the firing auxiliary so that a failure $f_v!$ is emitted whenever either the BE fails (via f_v^*) or any of the triggers fails (via $f_t \in T_v$). As an intermediate step, let

$$[\![v]\!]_1 \;=\; [\![l(v)]\!]_{\mathsf{ELT}}(a_v, f_v^*) \parallel \mathit{FA}(f_v, f_v^*, T_v)$$

Here, $T_v = \{f_t \mid \exists w \in V \,.\, (v,w) \in E \wedge l(w) = \mathit{FDEP} \wedge t = (\mathit{preds}(w))_1\}$ is the set of trigger signals of FDEP gates on which $l(v)$ is dependent.[4]

Second, we need to activate the BE if it is used as a spare. This is done by composing $[\![v]\!]_1$ in parallel with an activation auxiliary (see Figure 4.c and Figure 4.d), where the latter outputs the activation signal a_v of $l(v)$. Thus we have

$$[\![v]\!] \;=\; [\![v]\!]_1 \parallel \mathit{AA}(a_v, \mathit{Atv}_v)$$

Here, $\mathit{Atv}_v = \{a_{v,w} \mid v \in \mathit{preds}(w) \wedge \mathit{type}(w) = \mathit{SPARE}\}$ is the set of activation signals emitted by all SPARE gates sharing $l(v)$.

SPARE. If v is labeled as a SPARE gate, with $\mathit{preds}(v) = w_1 w_2 \dots w_n$, then w_1 is its primary BE and $w_2, \dots, w_n$ are its $n-1$ spare BEs. As with the other gates, $[\![v]\!]$ is obtained from $[\![l(v)]\!]_{\mathsf{ELT}}$ by instantiating its parameters in such a way that the input signals of v connect to output signals of v's primary and

[4] If v is not a dependent event, the firing auxiliary can be omitted and we have $[\![v]\!]_1 = [\![l(v)]\!]_{\mathsf{ELT}}(a_v, f_v)$.

spare BEs. In addition, we need to find all the other SPARE gates that share any of v's spare BEs. Following the SPARE gate semantics in Section 3.2, we have

$$[\![v]\!] = [\![l(v)]\!]_{\mathsf{ELT}}(f_v, f_{w_1}, (f_{w_2}, a_{w_2,v}, P_{w_2}), \ldots, (f_{w_n}, a_{w_n,v}, P_{w_n}))$$

where $P_{w_i} = \{a_{w_i,g} \mid (w_i, g) \in E \wedge g \neq v \wedge type(g) = SPARE\}$ is the set of activation signals emitted by all other SPARE gates sharing spare $l(w_i)$.

We do not define node semantics for nodes labeled by an FDEP gate, since these are already incorporated in the semantics of their dependent BEs. Now, the semantics of a DFT is obtained by parallel composing the semantics of all (non-FDEP) nodes.

Definition 6. *The semantics of a DFT $\mathcal{D} = (V, preds, l)$ is the I/O-IMC* $[\![\mathcal{D}]\!] = \|_{v \in V \mid type(v) \neq FDEP} [\![v]\!]$.

To compute the reliability of $\mathcal{D}$, we are only interested in the failure of the top node T. Hence, we hide all signals except f_T, i.e. we compute $M_{\mathcal{D}} = hide\ A_{\mathcal{D}} \setminus f_T\ in\ [\![\mathcal{D}]\!]$; recall that $A_{\mathcal{D}}$ denotes the set of all actions in $\mathcal{D}$. The compositional aggregation technique described in the following section is an efficient way to derive $M_{\mathcal{D}}$.

Example 3. Figure 5 shows the I/O-IMC semantics of a DFT consisting of a SPARE gate A having a primary B and a spare C. The I/O-IMC of the DFT is obtained by parallel composing the seven I/O-IMCs shown on the figure:

$[\![A]\!] = [\![(SPARE, 2)]\!]_{\mathsf{ELT}}(f_A, f_B, (f_C, a_{C,A}, \emptyset))$
$[\![B]\!]_1 = [\![(BE, 0, \lambda, 0)]\!]_{\mathsf{ELT}}(a_B, f_B*) \| FA(f_B, f_B*, \emptyset)$ $[\![B]\!] = [\![B]\!]_1 \| AA(a_B, \emptyset)$
$[\![C]\!]_1 = [\![(BE, 0, \lambda, \mu)]\!]_{\mathsf{ELT}}(a_C, f_C*) \| FA(f_C, f_C*, \emptyset)$ $[\![C]\!] = [\![C]\!]_1 \| AA(a_C, \{a_{C,A}\})$

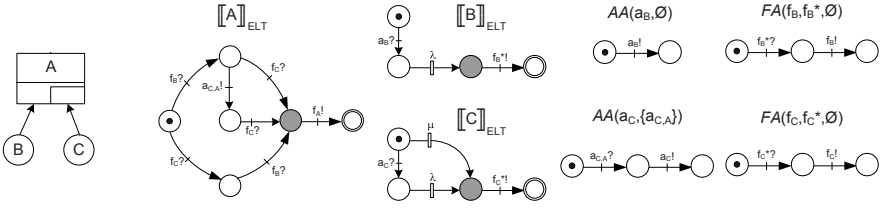

Fig. 5. A DFT and the seven I/O-IMCs that model its behavior

4 Compositional Aggregation Approach

Our compositional semantics allows one to build the I/O-IMC associated to a DFT in a component-wise fashion, leading to a significant state-space reduction. This kind of compositional aggregation approach has been previously successfully used, most notably in [16]. The compositional aggregation approach is to be contrasted with a more classical approach of model generation, such as the one

used by DIFTree, where the CTMC model of a dynamic system is generated at once and as a whole and then possibly aggregated at the end. We propose the following conversion algorithm to transform a DFT into an I/O-IMC.

1. Translate each DFT element to its corresponding (aggregated) I/O-IMC.
2. Pick two I/O-IMCs and parallel compose them (Definition 2).
3. Hide (Definition 2) output signals that will not be subsequently used (i.e. synchronized on).
4. Aggregate, using weak bisimulation (Definition 3), the I/O-IMC obtained in step 3.
5. Go to step 2 if more than one I/O-IMC is left, otherwise stop.

The choice of I/O-IMCs we make in step 2 is important as this has an impact on the size of the generated state space during the intermediate steps. In the case studies (see Section 5) we have used intuitive heuristics based on the level of interaction between models to decide the composition order. In the absence of simultaneous failures [4] in the DFT model, the algorithm results in an aggregated CTMC. However, in cases with simultaneous failures the result can be a continuous-time Markov decision process, which can be analyzed by computing bounds on the performance measure of interest [2,15].

Note that originally non-determinism was not intended to be present in DFT models. Our algorithm also yields a well-specified check for DFTs: By seeing whether the I/O-IMC translation yields a CTMC or a CTMDP, one can decide if any unintended non-determinism is present in a DFT.

5 Tool Support and Case Studies

We have developed a tool named *CORAL* [3] (COmpositional Reliability and Availability anaLysis) that takes as input a DFT specified in the Galileo DFT format and computes, if there is no non-determinism in the resulting I/O-IMC, the unreliability of the DFT for given mission times. CORAL is integrated with the CADP tool set [9], which provides tool support for IMCs [13].

The tool consists of three parts:

1. The *dft2bcg* tool which uses as input the DFT file in Galileo format. This tool translates the elements of the DFT into their I/O-IMC counterparts.
2. The *composer* tool which uses as input the I/O-IMC models created by the dft2bcg tool and a composition order. The composer tool applies compositional aggregation to the I/O-IMCs according to the composition order to generate a single I/O-IMC representing the DFT's behaviour. The composition order must be supplied by the user (see Section 4).
3. The *dft_eval* tool with as its input the I/O-IMC generated by the composer tool and a number of mission-times. The dft_eval tool calculates the unreliability of the system modeled by the original DFT for the given mission-times.

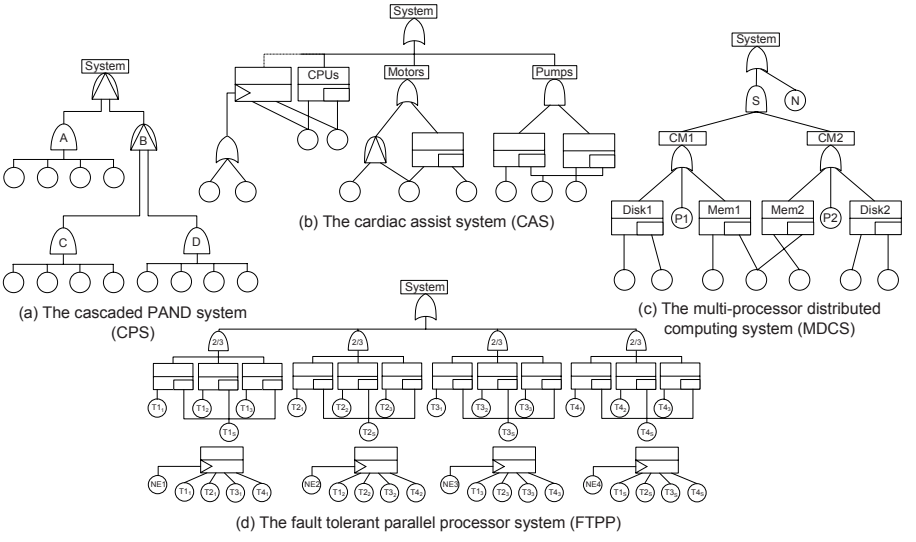

Fig. 6. The DFT representations of the case studies

The composer tool uses the CADP tool set to compose, abstract (i.e. hide signals) and aggregate I/O-IMCs. In particular we have used a version of the bcg_min tool, which was adapted to aggregate I/O-IMCs using weak bisimulation (see Definition 3).

To compare the compositional aggregation (Comp-Aggr) approach with the traditional DIFtree method, we have conducted four case studies (none having non-determinism). Figure 6 shows the cascaded PAND system [6,7] (CPS), the cardiac assist system [7] (CAS), the multi-processor distributed computing system [18] (MDCS) and the fault-tolerant parallel processor [11] (FTPP).

The results of the case studies are given in Figure 7. The size of the largest model (with regard to the number of states) appearing during analysis is given for each experiment.

Case study	Analysis method	Maximum number of states	Maximum number of transitions	Unreliability (Mission-time = 1)
CPS	DIFTree	4113	24608	0.00135668
CPS	Comp-Aggr	132	426	0.00135668
CAS	DIFtree	8	10	0.657900
CAS	Comp-Aggr	36	119	0.657900
MDCS	DIFtree	253	1383	$2.00025 \cdot 10^{-9}$
MDCS	Comp-Aggr	157	756	$2.00025 \cdot 10^{-9}$
FTPP	DIFtree	32757	426826	$2.56114 \cdot 10^{-11}$
FTPP	Comp-Aggr	1325	14153	$2.56114 \cdot 10^{-11}$

Fig. 7. The results of the case studies

From these experiments one can conclude that the compositional aggregation approach to analyzing DFTs is very promising and we expect it to combat the state-space explosion effectively in many cases. The relative performance of the DIFtree and compositional aggregation approaches vary greatly for different DFTs. The DIFtree method seems to perform better for DFTs with few basic events and (possibly) many interconnections (e.g. each of the three modules in the CAS). The compositional aggregation approach seems to perform better in DFTs with symmetries (such as in the CPS and FTPP examples) and DFTs with a large number of elements and few connections (and highly modular). More research is needed to further investigate which method is best to apply under which circumstances.

6 Conclusions and Future Work

In this paper, we have formalized the semantics of DFTs using I/O-IMCs (a variant of IMCs that we have defined) and showed how a compositional aggregation approach is used to analyze DFTs. Being a first step, we have restricted our attention to basic events with exponential failure distributions, but the same approach could be taken for other probability distributions using phase-type distributions or a different underlying formalism, e.g. the one in [8]. Future work includes experimenting with other case studies and improving on the heuristic used in the order of the I/O-IMCs composition (for instance by adapting and implementing the heuristics proposed in [19]).

Acknowledgment. We thank Hong Xu from the University of Virginia for running various experiments with the Galileo tool.

References

1. Amari, S., Dill, G., Howald, E.: A new approach to solve dynamic fault trees. In: Annual Reliability and Maintainability Symposium, pp. 374–379 (January 2003)
2. Baier, C., Hermanns, H., Katoen, J.P., Haverkort, B.R.: Efficient computation of time-bounded reachability probabilities in uniform continuous-time markov decision processes. Theor. Comput. Sci. 345(1), 2–26 (2005)
3. Boudali, H., Crouzen, P., Stoelinga, M.I.A.: CORAL - a tool for compositional reliability and availability analysis. In: ARTIST workshop. Presented at the 19th international conference on Computer Aided Verification
4. Boudali, H., Crouzen, P., Stoelinga, M.I.A.: Dynamic fault tree analysis using input/output interactive markov chains. In: Proc. of Dependable Systems and Networks conference, UK, pp. 708–717. IEEE Computer Society, Los Alamitos (2007)
5. Boudali, H., Crouzen, P., Stoelinga, M.I.A.: Compositional analysis of dynamic fault trees. Technical report, University of Twente (to appear)
6. Boudali, H., Dugan, J.B.: A discrete-time Bayesian network reliability modeling and analysis framework. Reliability Engineering and System Safety 87(3), 337–349 (2005)

7. Boudali, H., Dugan, J.B.: A new Bayesian network approach to solve dynamic fault trees. In: Proc. of Reliability and Maintainability Symposium, pp. 451–456. IEEE, Los Alamitos (2005)
8. Bravetti, M., Gorrieri, R.: The theory of interactive generalized semi-markov processes. Theoretical Computer Science 282(1), 5–32 (2002)
9. Construction and Analysis of Distributed Processes (CADP) software tool. `http://www.inrialpes.fr/vasy/cadp/`
10. Coppit, D., Sullivan, K.J., Dugan, J.B.: Formal semantics of models for computational engineering: A case study on dynamic fault trees. In: Proc. of the Inter. Symp. on Software Reliability Engineering, pp. 270–282. IEEE, Los Alamitos (2000)
11. Dugan, J.B., Bavuso, S.J., Boyd, M.A.: Dynamic fault-tree models for fault-tolerant computer systems. IEEE Trans. on Reliability 41(3), 363–377 (1992)
12. Dugan, J.B., Venkataraman, B., Gulati, R.: DIFTree: a software package for the analysis of dynamic fault tree models. In: Reliability and Maintainability Symposium, pp. 64–70 (January 1997)
13. Garavel, H., Hermanns, H.: On combining functional verification and performance evaluation using cadp. In: Eriksson, L.-H., Lindsay, P.A. (eds.) FME 2002. LNCS, vol. 2391, pp. 410–429. Springer, Heidelberg (2002)
14. Hermanns, H.: Interactive Markov Chains. LNCS, vol. 2428. Springer, Heidelberg (2002)
15. Hermanns, H., Johr, S.: Uniformity by construction in the analysis of nondeterministic stochastic systems. In: Proc. of Dependable Systems and Networks conference, UK, pp. 718–728. IEEE Computer Society, Los Alamitos (2007)
16. Hermanns, H., Katoen, J.P.: Automated compositional Markov chain generation for a plain-old telephone system. Sci. of Comp. Programming 36(1), 97–127 (2000)
17. Lynch, N.A., Tuttle, M.R.: An introduction to input/output automata. CWI Quarterly 2(3), 219–246 (1989)
18. Malhotra, M., Trivedi, K.S.: Dependability modeling using petri-nets. IEEE Transactions on Reliability 44(3), 428–440 (1995)
19. Tai, K.-C., Koppol, P.V.: An incremental approach to reachability analysis of distributed programs. In: Software Specifications & Design workshop, IEEE Computer Society Press, Los Alamitos (1993)
20. Veseley, W.E., Goldberg, F.F., Roberts, N.H., Haasl, D.F.: Fault tree handbook, NUREG-0492. Technical report, NASA (1981)