AltaRica 3.0 project: compile Guarded Transition Systems into Fault Trees

T. Prosvirnova & A. Rauzy
LIX
Ecole Polytechnique, Palaiseau, France

ABSTRACT: The goal of this communication is to present an algorithm to compile Guarded Transition Systems into Boolean equations (Fault Trees). This work is done as a part of AltaRica 3.0 project, which aims to design a new version of AltaRica and to develop a complete set of authoring and assessment tools for this new version of the language. AltaRica 3.0 improves significantly the expressive power of AltaRica Data-Flow without decreasing the efficiency of its assessment algorithms. Its underlying mathematical model – Guarded Transition Systems(GTS) – makes it possible to design acausal components and to handle looped systems. GTS is a states/transitions formalism that generalizes classical safety formalisms, such as Reliability Block Diagrams and Markov chains. The compilation of GTS into Fault Trees is of interest for several reasons. First, some regulation authorities still require Fault Trees to support the certification process. Second, the automated generation of Fault Trees from higher level representations makes easier their maintenance through the life cycle of systems. Finally, assessment tools for Boolean models are much more efficient than those for states/transitions models.

1 INTRODUCTION

Fault trees are widely used to perform Safety Analyses and some regulation authorities require them to support the certification process. Efficient algorithms and tools are available to assess this kind of models. However, despite of their qualities, these formalisms share a major drawback: models are far from the specifications of the systems under study. As a consequence, models are hard to design and to maintain throughout the life cycle of systems. A small change in the specifications may require revisiting completely safety models, which is both resource consuming and error prone.

The high-level modeling language AltaRica Data-Flow (Rauzy 2002, Boiteau et al. 2006) has been created to tackle this problem. AltaRica Data-Flow models are made of hierarchies of reusable components. Graphical representations are associated with components, making models visually very close to Process and Instrumentation Diagrams. AltaRica Data-Flow is at the core of several Integrated Modeling and Simulation Environments. Successful industrial experiments using AltaRica Data-Flow have been reported (Bernard et al. 2007, Bieber et al. 2008). In a word, AltaRica Data-Flow has reached an industrial maturity.

However, more than ten years of experience showed that both the language and the assessment tools can be improved. AltaRica 3.0 is an entirely new version of the language. Its underlying mathematical model – Guarded Transition Systems (Rauzy 2008, Prosvirnova and Rauzy 2012) – makes it possible to design acausal components and to handle looped systems. The development of a complete set of freeware authoring and assessment tools is planed, so to make them available to a wide audience.

Guarded Transition Systems (GTS) is a states/transitions formalism, that generalizes classical safety formalisms, such as Reliability Block Diagrams, Markov chains and Petri Nets. From a GTS model, it is possible to generate corresponding Fault Trees (FT), i.e. to transform a states/transitions model into a set of Boolean formulae. It may seem inefficient at a first glance to use a states/transitions formalism to end up with a Fault Tree. However in practice, it is of great interest. It is easier and less time consuming to automatically generate FT from high-level models rather than create them from scratch. High-level models improves greatly the design, the sharing and the maintenance of models. Moreover, to assess the generated Fault Trees a calculation engine XFTA (Rauzy 2012) can be used. XFTA relies on a powerful and original algorithm to extract Minimal Cutsets and implements calculation of usual reliability indicators, such as top-event

probabilities, importance factors, sensitivity analyses and Safety Integrity Levels.

The algorithm of compilation to Fault Trees for AltaRica Data-Flow, described in Rauzy (2002), can be extended to a general case of GTS. The objective of this article is to present the extended algorithm. It is organized as follows. Section 2 gives an overview of the AltaRica 3.0 project. Section 3 introduces the formalism of Guarded Transition Systems (GTS). Section 4 presents the algorithm of compilation of GTS to Fault Trees. Section 5 presents related works. Finally Section 6 concludes the article and outlines directions for future works.

2 ALTARICA 3.0 PROJECT

The objective of the AltaRica 3.0 project is to propose a set of modeling and assessment tools to perform preliminary safety analyses. Figure 1 presents the overview of the project.

AltaRica 3.0 is in the heart of the project. It significantly increases the expressive power of AltaRica Data-Flow without decreasing the efficiency of assessment algorithms. Models are compiled into a low level formalism: Guarded Transition Systems (GTS). GTS is a states/transitions formalism generalizing classical safety formalisms, such as Reliability Block Diagrams and Markov Chains. It is a pivot formalism for Safety Analyses: other safety models can be compiled into GTS to take benefits from assessment tools. The assessment tools for GTS already include a Fault Tree compiler to perform Fault Tree Analysis (FTA), a Markov Chain generator, a stochastic and a stepwise simulators. Other tools are under specification or implementation: a model-checker and a reliability allocation module. These tools will be distributed under a free license in order to make them available to a wide audience, especially in the academic community. They enable users to perform virtual experiments on systems, to compute reliability indicators and, also, to perform cross check calculations.

3 GUARDED TRANSITION SYSTEMS

3.1 *Definition*

A Guarded Transition Systems is formally a quintuple $\langle V, E, T, A, \iota \rangle$, where:

- $V = S \uplus F$ is a set of variables, divided into disjoint sets S of state variables and F of flow variables.
- E is a set of symbols, called events.
- T is a set of transitions.
- A is an assertion (i.e. an instruction built over V).
- ι is the initial (or default) assignment of variables of V.

GTS is thus a states/transitions formalism where states are implicit, i.e. given by variables assignments σ. A transition is a triple $\langle e, G, P \rangle$, also denoted $e : G \rightarrow P$, where $e \in E$ is an event, G is a guard, i.e. a Boolean formula built over V, and P is an instruction built over V, also called an action or a post-condition. A transition $e : G \rightarrow P$ is said *fireable* in a given state σ if its guard G is satisfied in this state.

Instructions

Both assertions and actions of transitions are described by means of instructions. There are basically four types of instructions:

- The *empty instruction* noted $skip$.
- The *assignment* $v := E$, where v is a variable and E is an expression built over variables from V.
- The *conditional assignment* $if\ C\ then\ I$, where C is a Boolean expression and I is an instruction.
- The *block* $\{I_1, \ldots, I_n\}$, where I_1, ..., I_n are instructions.

State variables can be presented as the left member of an assignment only in the action of a transition. Flow variables can be presented as the left member of an assignment only in the assertion.

Example

Consider a system composed of two pumps connected in series (see Figure 2). This system is represented by GTS as follows:

```
domain PumpState {WORKING, FAILED};
class TwoPumps
  PumpState p1.state (init = WORKING);
  PumpState p2.state (init = WORKING);
  Boolean p1.input(reset = false), p1.output (reset = false);
  Boolean p2.input(reset = false), p2.output (reset = false);
  Boolean input(reset = false), output (reset = false);
  event p1.failure, p2.failure;
  transition
    p1.failure:  (p1.state==WORKING)  -> p1.state := FAILED;
    p2.failure:  (p2.state==WORKING)  -> p2.state := FAILED;
  assertion
    input := true;
    p1.input := input;
    p1.output := if p1.state==WORKING then p1.input else false;
    p2.input := p1.output;
    p2.output := if p2.state==WORKING then p2.input else false;
    output := p2.output;
end
```

The states of the pumps are represented by state variables *p1.state* and *p2.state*, which can be WORKING or FAILED. Events *p1.failure* and *p2.failure* represent failures of the pumps, and transitions describe how changes the state of the system. In the transition, labeled by the event *p1.failure*, *p1.state==WORKING* is the guard and *p1.state := FAILED;* is the action. Assertion expresses how flow variables *p1.output, p2.output, p1.input, p2.input, output* are calculated according to the state variables of the system.

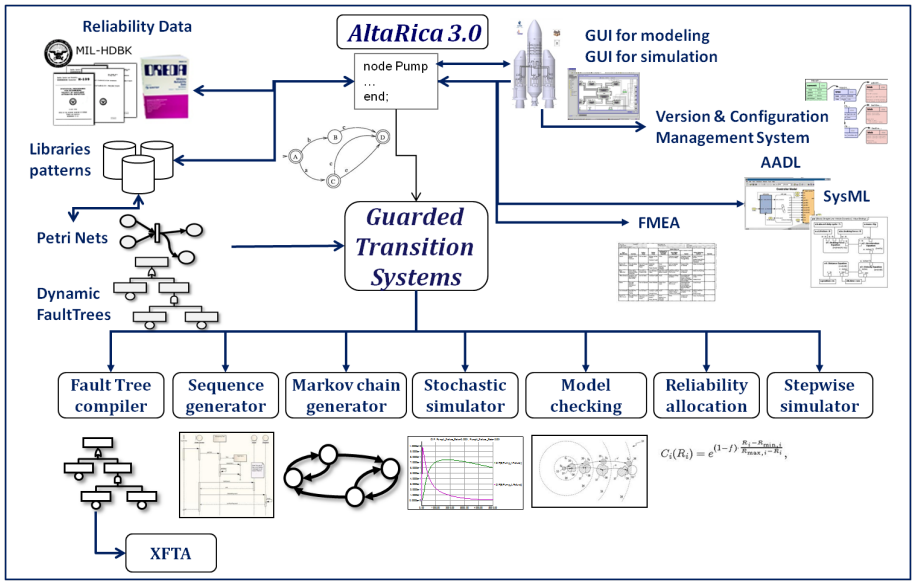

Figure 1: Overview of the AltaRica 3.0 project

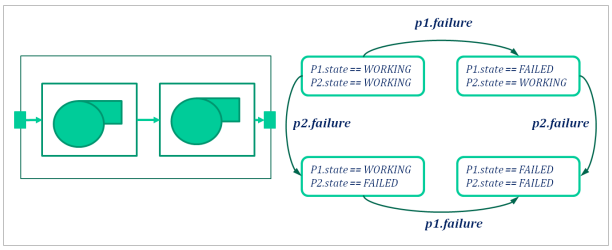

Figure 2: Pumps connected in series and the corresponding reachability graph

3.2 *Reachability graph*

Guarded Transition Systems are implicit representations of labeled Kripke structures, i.e. of graphs whose nodes are labeled by variable assignments and whose edges are labeled by events. The reachability graph corresponding to the GTS given Section 3.1 is illustrated Figure 2. For the sake of the clarity, flow variables are not indicated on this picture, their value is determined from the value of state variables.

To describe how this graph is constructed, we should first explain the way instructions are interpreted.

3.2.1 *Instructions*

Instructions are interpreted in a slightly different way depending they are used in the actions or in the assertion. Let σ be the variable assignment before the firing of the transition $e : G \rightarrow P$. Applying the instruction P to the variable assignment σ consists in calculating a new variable assignment τ. The right hand side of assignments and conditional expressions are evaluated in the context of σ. Thus, the result does not depend on the order in which instructions of a block are applied. In other words, instructions of a block are applied in parallel. Let denote by $Update(P, \sigma)$ the variable assignment τ resulting from the application of the instruction P to σ.

Let A be the assertion and τ the variable assignment obtained after the application of the action of a transition. Applying A consists in calculating a new variable assignment (of flow variables) π as follows. We start by setting all state variables in π to their values in τ: $\forall v \in S \pi(v) = \tau(v)$. Let D be a set of unevaluated flow variables, we start with $D = F$. Then,

- If A is an empty instruction, then π is left unchanged.
- If A is an assignment $v := E$, then if $\pi(E)$ can be evaluated in π, i.e. all variables of E have a value in π, then $\pi(v)$ is set to $\pi(E)$ and v is removed from D. An error is raised if the value of v has been already modified and is different from the calculated one.
- If A is a conditional assignment $if\ C\ then\ I$ and $\pi(C)$ can be evaluated in π and is true, then the instruction I is applied to π. Otherwise, π is left unchanged.
- If A is a block of instructions $\{I_1, \ldots, I_n\}$ then instructions $I_1, \ldots, I_n$ are repeatedly applied to π until there is no more possibility to assign a flow variable.

If after applying A to π there are unevaluated variables in D, then all these variables are set to their default values $\forall v \in D \pi(v) = reset(v)$ and A is applied to π in order to verify that all assignments are satisfied. If that is not true an error is raised. Let denote by $Propagate(A, \sigma)$ the variable assignment resulting from the application of the instruction A to σ.

3.2.2 *Calculation of the Reachability Graph*

Assume that σ is the variable assignment just before the firing of a transition. Then, the firing of the transition transforms σ into the assignment $Fire(e : G \rightarrow$

$P, A, \sigma)$ defined as follows:

$$Fire(e : G \to P, A, \sigma) = Propagate(A, Update(P, \sigma))$$

The so-called reachability graph $\Gamma = (\Sigma, \Theta)$ is the smallest Kripke structure verifying:

1. $\sigma_0 = Propagate(A, \iota, \iota) \in \Sigma$. σ_0 is the initial state of the Kripke structure.
2. If $\sigma \in \Sigma$ and $\exists t = \langle e, G, P \rangle \in T$, such that the guard G is verified in σ then the state $\tau = Fire(P, A, \iota, \sigma) \in \Sigma$ and the transition $(\sigma, e, \tau) \in \Theta$,

In special cases, the calculation of $\Gamma = (\Sigma, \Theta)$ may raise errors. A well designed GTS avoids this problem. Currently assessment tools detect modeling errors at the execution of the model.

3.3 *Stochastic models*

A probabilistic time structure can be put on top of a Guarded Transition System so to get timed/stochastic models. The idea is to associate to each event:

- A delay which can be deterministic or stochastic and may depend on the state. When a transition labeled with the event becomes fireable at time t, a delay d is calculated and the transition is actually fired at time $t + d$ if it stays fireable from t to $t + d$.
- a weight, called expectation, used to determine the probability that the transition is fired in case of several transitions are fireable at the same date.

Let define an oracle $o : \mathbb{N} \to [0; 1] \subset \mathbb{R}$, an infinite sequence of real numbers comprised between 0 and 1 (included). The only operation available on an oracle is to consume its first element. This operation returns the first element and the remaining of the sequence (which is itself an oracle).

Formally, a Stochastic Guarded Transition System is a tuple $\langle V = S \cup F, E, T, A, \iota, delay, expectation \rangle$, where

- $\langle V = S \cup F, E, T, A, \iota \rangle$ is a GTS;
- delay is a function from events and oracles to non-negative real numbers $delay : E \times O \to \mathbb{R}_+$;
- expectation is function from events to positive real numbers $expectation : E \to \mathbb{R}_+$.

For the sake of simplicity we made $delay$ depend only on the event and the oracle. It is however possible that $delay$ depends on the current state and the elapsed time since the beginning of the mission. When several transitions are scheduled to be fired at the same date, i.e. $\exists e_1, e_2, \ldots, e_k$ such that $d_n(e_1) = 0, d_n(e_2) = 0, \ldots, d_n(e_k) = 0$, one is picked at random by using the oracle and according to their expectations. The probability $p(e_k : G_k \to P_k)$ to fire the transition $e_k : G_k \to P_k$, is defined as follows.

$$p(e_k : G_k \to P_k) = \frac{expectation(e_k)}{\sum_{e_i : d_n(e_i) = 0} expectation(e_i)} \quad (1)$$

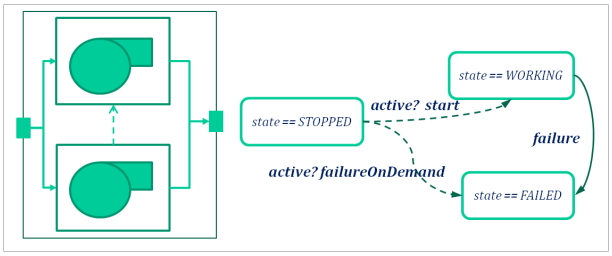

Figure 3: Spare pumps

The semantics of Stochastic Guarded Transition Systems is defined in terms of extended Reachability Graph, where states are defined by couples (σ, d) with σ being a variable assignment and d being a vector of dates, when each transition $t \in T$ should be fired $(d : T \to \mathbb{R}_+)$. The semantics of Stochastic Guarded Transition Systems can also be defined in terms of execution runs.

Example

Consider now a system, composed of two pumps in cold redundancy (see Figure 3). When the primary pump fails, the spare one starts working. Moreover, the spare pump may fail to start with a probability γ.

The corresponding GTS is as follows:

```
domain PumpState {STOPPED, WORKING, FAILED};
class SparePumps
  PumpState p1.state (init = WORKING);
  PumpState p2.state (init = STOPPED);
  Boolean p1.input(reset = false), p1.output (reset = false);
  Boolean p2.input(reset = false), p2.output (reset = false);
  Boolean p1.active(reset = false), p2.active (reset = false);
  Boolean input(reset = false), output (reset = false);
  parameter Real p1.lambda = 0.0001;
  parameter Real p2.lambda = 0.0001;
  parameter Real p1.gamma = 0.01;
  parameter Real p2.gamma = 0.01;
  event p1.failure(delay = exponential(p1.lambda));
  event p2.failure(delay = exponential(p2.lambda));
  event p1.start(delay = 0, expectation = 1 - p1.gamma);
  event p1.failureOnDemand(delay = 0, expectation = p1.gamma);
  event p2.start(delay = 0, expectation = 1 - p2.gamma);
  event p2.failureOnDemand(delay = 0, expectation = p2.gamma);
  transition
    p1.failure:  (p1.state==WORKING)  -> p1.state := FAILED;
    p2.failure:  (p2.state==WORKING)  -> p2.state := FAILED;
    p1.start:  (p1.state==STOPPED) and p1.active ->
        p1.state := WORKING;
    p1.failureOnDemand:  (p1.state==STOPPED) and p1.active ->
        p1.state := FAILED;
    p2.start:  (p2.state==STOPPED) and p2.active ->
        p2.state := WORKING;
    p2.failureOnDemand:  (p2.state==STOPPED) and p2.active ->
        p2.state := FAILED;
  assertion
    input := true;
    p1.input := input;
    p2.input := input;
    p1.output := if p1.state==WORKING then p1.input else false;
    p2.output := if p2.state==WORKING then p2.input else false;
    output := p1.output or p2.output;
    p1.active := true;
    p2.active := (p1.state == FAILED);
end
```

In this example we deal with two types of events:

- Timed stochastic events: *p1.failure* and *p2.failure*. Their delays are exponentially distributed random variables.
- Immediate events: *p1.start* and *p1.failureOnDemand*, *p2.start* and *p2.failureOnDemand*. They should occur immediately, when the guard of the transition is satisfied.

Transitions, labeled by immediate events, are represented by dashed lines and those, labeled by timed stochastic events, are represented by plane lines (see Figure 3). The immediate events also have an attribute *expectation*. If several transitions are fireable at the same moment, the value of expectation determines the probability for each transition to be fired.

4 COMPILATION INTO FAULT TREES

It is possible to transform a Guarded Transition System into a set of Boolean formulae (a Fault Tree). This compilation is of interest for several reasons: first, it is easier and less time consuming to automatically generate Fault Trees from high-level models rather than create them from scratch, second, high-level models improves greatly the design, the sharing and the maintenance of models, finally, assessment tools for Boolean models are much more efficient than those for states/transitions models. However, the price to pay is the loss of sequencing among events: sequences of events are compiled into conjuncts of events. If the GTS is combinatorial, its compilation to Fault Trees is efficient and does not loose information. Many real-life models are relatively simple extensions of Reliability Block Diagrams and, thus, can be compiled efficiently into Fault Trees.

In this section we introduce the algorithm of compilation of Guarded Transition Systems to Fault Trees. This algorithm is an extension of the compilation algorithm of AltaRica Data-Flow, described in (Rauzy 2002).

4.1 *Principle of the compilation*

Each hierarchical AltaRica 3.0 model can be flattened into a unique Guarded Transition System. From now, we assume that the GTS $G = \langle V, E, T, A, \iota \rangle$, obtained by flattening an AltaRica 3.0 model, describes a system that may fail. The graph $\Gamma = (\Sigma, \Theta)$ is the reachability graph of G. The initial state, or initial variable assignment, $\sigma_0 \in \Sigma$ represents the nominal state of the system. Events from E represent failures of system components. Some states (variables assignments) $\sigma \in \Sigma$ represent failure states. Paths from σ_0 to these states represent scenarios of failure. The compilation captures failure scenarios into a set of Boolean equations. It produces a Boolean formula $\phi_{v,c}$ for each pair

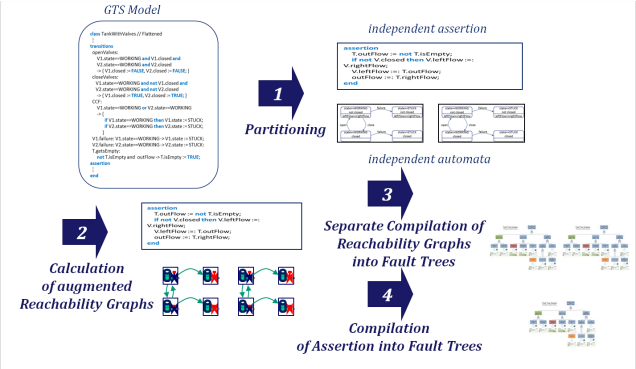

Figure 4: Algorithm of compilation of GTS to FT

(v, c), where v is a variable from V and c is its value, $c \in dom(v)$, such that the variables of $\phi_{v,c}$ are events from E, if $\{e_1, \ldots, e_k\}$, $e_i \in E \forall i = 1..k$ is a minimal cutset of $\phi_{v,c}$, then there is a path in the reachability graph Γ, such that $(\sigma_0, e_1, \sigma_1) \in \Theta, (\sigma_1, e_2, \sigma_2) \in \Theta, \ldots, (\sigma_{k-1}, e_k, \sigma_k) \in \Theta$, and $\sigma_k(v) = c$.

The algorithm of compilation of GTS into Fault Trees includes 4 steps (see Figure 4):

1. The GTS model is partitioned into independent GTS and an independent assertion.
2. Reachability graphs of each independent GTS are calculated.
3. Each reachability graph is separately compiled into Boolean equations.
4. The independent assertion is compiled into Boolean equations.

Each of these steps is described in details in the following sections.

The generated Fault Tree could be assessed with any Fault Tree calculation engine supporting Open-PSA format (Hibti et al. 2012). For example, the calculation engine XFTA (Rauzy 2012) can be used to calculate minimal cutsets, events probabilities, importance factors, etc.

4.2 *GTS partitioning*

Partitioning is a key point of the algorithm that ensures its efficiency. In practice, components of a system fail in general in a relatively independent way. In that case a partitioning is possible. Partitioning of a GTS $G = \langle V, E, T, A, \iota \rangle$ consists in representing G in the following way (see Figure 5 as an illustration):

$$G = G_1 \uplus G_2 \uplus \ldots \uplus G_n \cup \langle \tilde{V}, \tilde{A}, \tilde{\iota} \rangle,$$

where
$G_i = \langle V_i, E_i, T_i, A_i, \iota_i \rangle$ are independent Guarded Transition Systems, and
$\langle \tilde{V}, \tilde{A}, \tilde{\iota} \rangle$ is an assertion, also called a glue. The last part (the glue) does not contain any behavior. Variables in $\tilde{V}$ are only flow variables, they depend

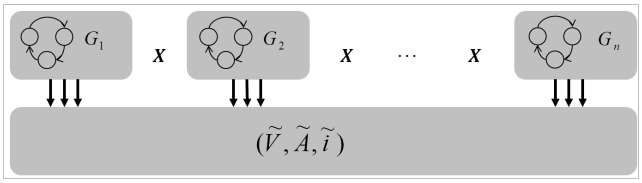

Figure 5: Partitioning of GTS

on state variables and flow variables of independent GTSs G_i. A similar idea can be found in Gössler and Sifakis (2005).

Let denote by $var(E)$ variables used in the expression E, and by $var(I)$ variables used in the instruction I. Let denote by V_t variables used by the transition $t = \langle e, G, P \rangle$: $V_t = var(G) \cup var(P) \cup V'$, where V' are variables, such that variables from $var(G) \cup var(P)$ depend on them via the assertion A. We say that two transitions t_1 and t_2 are independent if $V_{t1} \cap V_{t2} = \emptyset$.

Consider an undirected graph with nodes labeled by GTS transitions. There is an edge between two nodes of this graph if the transitions labeling the nodes are dependent (in the sense of the definition given earlier). The connected components of this graph give us a partition of GTS transitions and, therefore, of variables and events. The dependency graph built from the assertion A enables to detect flow variables, belonging to each independent GTS, and to partition the assertion A. The remaining flow variables and instructions from A constitute the independent assertion $< \tilde{V}, \tilde{A}, \tilde{\imath} >$, also called the glue.

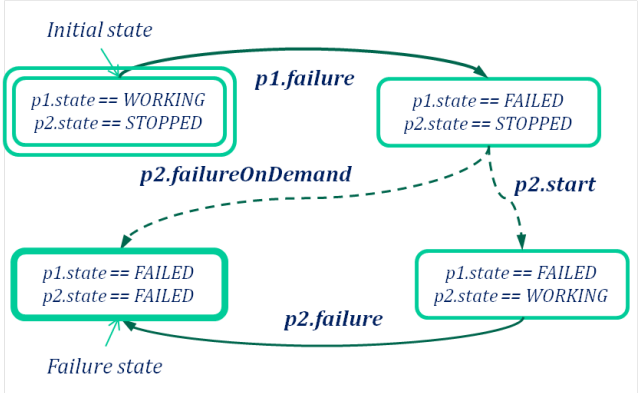

Figure 6: Reachability graph of two spare pumps

4.3 *Reachability graph generation*

The reachability graph is constructed according to the definition given in Section 3.2. In case of Stochastic models the algorithm is slightly modified. First of all we need to abstract from dates of transition firings. We consider only two possible cases: immediate events and timed stochastic events. For all immediate events the date of firing is set to 0, for all other transitions it is set to ∞. Then a reachability graph is constructed.

The reachability graph corresponding to the Spare pumps system in Section 3.3 is given Figure 6. Immediate transitions are represented by dashed lines and timed transitions by plane lines.

By bisimulation it is possible to transform this reachability graph into an equivalent one keeping only timed transitions. In that way we obtain another reachability without immediate transitions for the Spare pumps system (see Figure 7). Transitions *p1.failure* and *p2.start* are replaced by one timed transition labeled with a sequence of events *p1.falure, p2.start*. Transitions *p1.failure* and *p2.failureOnDemand* are replaced by one timed transition labeled with a sequence of events *p1.failure, p2.failureOnDemand*.

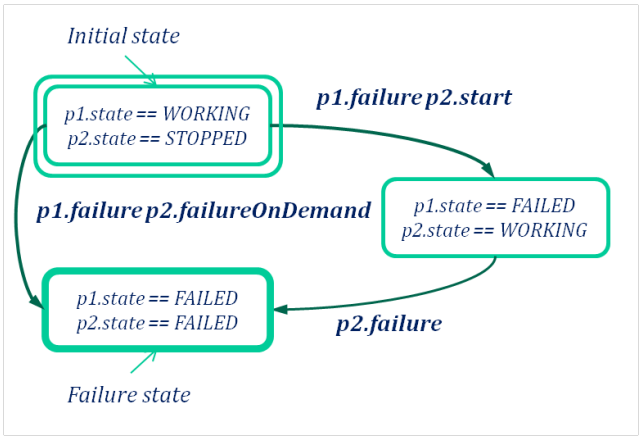

Figure 7: Reachability graph of two spare pumps obtained by bisimulation

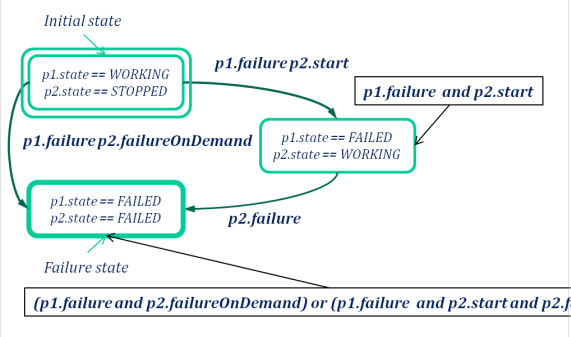

Figure 8: Boolean formulae associated with each state of the Reachability Graph

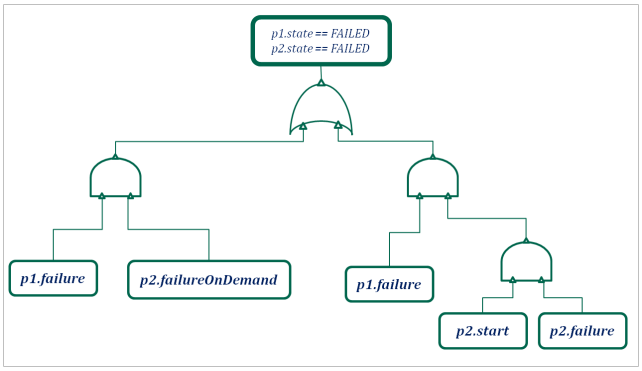

Figure 9: Generated Fault Tree

4.4 *Compilation of Reachability Graphs into Boolean formulae*

For each independent GTS G its reachability graph $\Gamma = (\Sigma, \Theta)$ is constructed. First of all, we search for all paths π from the initial state σ_0 to each state of the graph σ. These paths are transformed into the conjunction of events that label π. In order to avoid conflicts raised by the composition of components (for more details, see Rauzy (2002)) we need to consider not only events occurring along the path but also those that do not. Let denote by ϕ_π the conjunction of events that occur along the path and the negation of those that do not occur. We associate with each state $\sigma \in \Sigma$ the disjunction of ϕ_π over all paths π from the initial state σ_0 to the state σ. Finally, we associate with each pair (v, c), where v is a variable and $c \in dom(v)$ is its value, the disjunction of formulae that are associated with the states $\sigma \in \Sigma$, such that $\sigma(v) = c$.

The example of simplified Boolean formulae associated with each state of a Reachability Graph for the Spare pumps system is given Figure 8.

4.5 *Compilation of the assertion into Boolean formulae*

The assertion $< \tilde{V}, \tilde{A}, \tilde{\iota} >$ is also transformed into a set of Boolean formulae. For each pair (v, c), where $v \in \tilde{V}$ is a flow variable and $c \in dom(v)$ is its value we construct a Boolean formula $\phi_{v,c}$ according to the instructions in the assertion $\tilde{A}$ and Boolean formulae $\phi_{v',c'}$ obtained from the compilation of independent GTS.

As an illustration, the Fault Tree generated for the Spare pumps system is given Figure 9. As it was mentioned earlier, the sequencing between events is lost during the compilation: for example, the sequence of events *p1.failure, p2.failureOnDemand* is transformed into the conjunction of events *p1.failure and p2.failureOnDemand*.

5 RELATED WORKS

The HiP-HOPS workbench (Pasquini et al. 1999) enables to add reliability data to models imported from different modeling tools: Matlab/SIMULINK, Eclipse-based UML tools, etc., and then to automatically generate Fault Trees and FMEA tables. The underlying formalism of Hip-HOPS is a pseudo-Boolean formalism in which the system is described by hierarchies of blocks and the outputs of the blocks are written as a discrete function of internal failures and inputs. Guarded Transition Systems generalize this kind of models and the algorithm described in this article is very efficient on them.

In Bozzano et al. (2007) authors analyze different algorithms of symbolic model-checking techniques in order to automatically generate Fault Trees by calculating minimal cutsets.

In Bouissou et al. (1991) authors mention the automatic generation of Fault Trees from high-level models, using Figaro modeling language. Figaro is a textual modeling language dedicated to dependability assessment of complex systems, developed by EDF R&D. It combines object-orientation languages features, such as inheritance, and first order production rules (interaction and occurrence rules) and is used as a description language for the workbench KB3 (Bouissou 2005), to automatically perform systems dependability assessment.

6 CONCLUSIONS

In this article, we introduced an algorithm of compilation of Guarded Transition Systems into Fault Trees. This algorithm is a generalization of the algorithm for AltaRica Data-Flow. The compilation of GTS to Fault Trees is of interest for two reasons: first, assessment tools for Boolean models are much more efficient than those for states/transitions models; second, the automated generation of Fault Trees from high-level models makes easier their maintenance through the life cycle of systems under study.

This work is a part of AltaRica 3.0 project which aims to propose a set of authoring, simulation and assessment tools for high-level Model-Based Safety Analyses. AltaRica 3.0 is in the heart of this project. It is an entirely new version of the language. It improves AltaRica Data-Flow into two directions:

1. Its semantic is based on the new underlying mathematical model – Guarded Transition Systems (GTS) – that makes it possible to represent acausal components and to handle looped systems.

2. It provides new constructs to structure models, that greatly improves its capacity of reuse and knowledge capitalization.

The development of a complete set of freeware authoring and assessment tools is planed, so to make them available to a wide audience. The assessment tools already include prototypes of a compiler to Fault Trees, a compiler to Markov chains, a stochastic and a stepwise simulators.

Out future works will focus on testing and improvement of the algorithm (and its implementation) for industrial scale models and also on its adaption for generation of critical sequences of events.

REFERENCES

Bernard, R., J.-J. Aubert, P. Bieber, C. Merlini, & S. Metge (2007). Experiments in model-based safety analysis: flight controls. In *Proceedings of IFAC workshop on Dependable Control of Discrete Systems, Cachan*.

Bieber, P., J.-P. Blanquart, G. Durrieu, D. Lesens, J. Lucotte, F. Tardy, M. Turin, C. Seguin, & E. Conquet (2008, January). Integration of formal fault analysis in assert: Case studies and lessons learnt. In *Proceedings of 4th European Congress Embedded Real Time Software, ERTS 2008*, Toulouse (France).

Boiteau, M., Y. Dutuit, A. Rauzy, & J.-P. Signoret (2006). The altarica data-flow language in use: Assessment of production availability of a multistates system. *Reliability Engineering and System Safety 91*, 747–755.

Bouissou, M. (2005). Automated dependability analysis of complex systems with the kb3 workbench: the experience of edf r&d. In *Proceedings of the International Conference on Energy and Environment*.

Bouissou, M., H. Bouhadana, M. Bannelier, & N. Villatte (1991). Knowledge modelling and reliability processing: presentation of the figaro modelling language and associated tools. In *Proceedings of Safecomp'91*.

Bozzano, M., A. Cimatti, & F. Tapparo (2007). Symbolic fault tree analysis for reactive systems. In *Proceedings of the 5th international conference on Automated technology for verification and analysis*, Berlin, Heidelberg, pp. 162–176. Springer-Verlag.

Gössler, G. & J. Sifakis (2005). Composition for component-based modeling. *Science of Computer Programming 55*(1-3), 161–183.

Hibti, M., T. Friedlhuber, & A. Rauzy (2012, June). Overview of the open psa platform. In R. Virolainen (Ed.), *Proceedings of International Joint Conference PSAM'11/ESREL'12*.

Pasquini, A., Y. Papadopoulos, & J. McDermid (1999). Hierarchically performed hazard origin and propagation studies. *Computer Safety, Reliability and Security 1698 of LNCS*, 688–688.

Prosvirnova, T. & A. Rauzy (2012, Octobre). Guarded transition systems: Pivot modelling formalism for safety analysis. In J. Barbet (Ed.), *Actes du Congrès Lambda-Mu 18*.

Rauzy, A. (2002). Modes automata and their compilation into fault trees. *Reliability Engineering and System Safety 78*, 1–12.

Rauzy, A. (2008). Guarded transition systems: a new states/events formalism for reliability studies. *Journal of Risk and Reliability 222*(4), 495–505.

Rauzy, A. (2012, June). Anatomy of an efficient fault tree assessment engine. In R. Virolainen (Ed.), *Proceedings of International Joint Conference PSAM'11/ESREL'12*.