AltaRica

Constraint automata as a description language

G. Point
IXI/LaBRI - 110, rue Achard 33300 Bordeaux - FRANCE

A. Rauzy
LaBRI - 351, cours de Libération 33405 Talence - FRANCE

ABSTRACT : This paper presents an overview of the AltaRica language. This language is the core of the AltaRica workbench, which is dedicated to reliability and dependability analyses of critical systems. The AltaRica language is both formal, to ensure mathematical soundness, and graphical, to be user-friendly. It stands at a high level and is designed to be compiled into lower level formalisms such as Boolean formulae, Petri nets or finite state automata. It can be seen as a convenient mean to describe constraint automata.

KEYWORDS : High level description languages, reliability workbenches.

1. Introduction

Reliability analyses of critical systems are, in general, twofold. First, they aim to determine, as exhaustively as possible, the various *scenarii* of failure of the system under study. Second, they try to quantify these *scenarii*, i.e. to assess the probability that the system fails during a given mission time. Several formalisms are used as support languages of such analyses, e.g. Fault Trees, Petri nets or Markov graphs (see [AND 93] for a review). Each of them has distinguishing features that make it suitable to capture some (but not all) of the aspects of the behavior of the system. They share however two important characteristics. First, they are both formal and graphical. This ensures their mathematical soundness on the one hand and the user-friendship of the tools implementing them on the other hand. Second, they stand at a rather low level. This makes it possible to design simple and efficient assessment algorithms. This has also the drawback to make it difficult to design and to maintain the models, because of their inherent distance to the system under study.

The AltaRica workbench is designed to tackle this latter difficulty. It is based on the high level AltaRica language. The AltaRica language is also both formal and graphical. Moreover, it is designed to be compiled into the mentioned low level formalisms. The idea to combine graphical and textual descriptions is indeed very na-

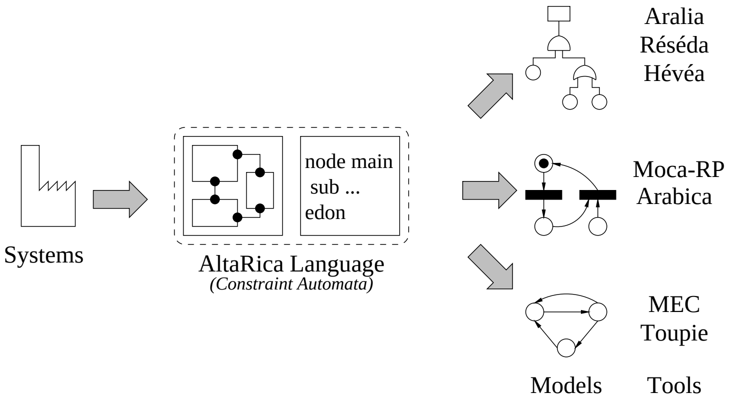


Figure 1. *The AltaRica workbench at a glance*

tural. It is applied in many areas, including the design of communication protocols (e.g. SDL [TUR 93]), the design of programmable logic controllers (e.g. IEC 1131-3 languages[BON 97]) or the design of reactive systems (e.g. Statecharts [HAR 87], Argos [MAR 91]). However, none of these formalisms is well suited for reliability analyses for they are designed to specify programs and not to describe behaviors of physical systems in presence of diseases. Several tools were designed for this latter purpose (e.g. FIABEX[HUT 94], FIGARO[BOU 93]). The originality of AltaRica (w.r.t. these tools) is twofold. First, the language has well-defined semantics. Second, the workbench is designed to embed tools stemmed from both the reliability engineering framework (Aralia [DUT 97], Moca-RP [SIG 95]) and the formal methods framework (MEC [ARN 94b], Toupie [COR 97]), as illustrated Fig. 1.

The AltaRica semantics is defined in terms of constraint automata [BRL 94]. Constraint automata are usual state automata, except that states and transitions are kept implicit, i.e. are described by means of constraints over variables. One of they key advantages of this presentation is that it makes it possible to describe long distance interactions (through shared variables) without loosing the locality of the descriptions. Constraint automata generalize both finite state machine and Petri nets. Compiling an AltaRica description into a Petri net or a finite state automaton is therefore rather easy. The compilation into Fault Trees is also possible, although this process transforms sequences of events into sets of events (loosing in that way the notion of event schedule).

This paper aims to provide the reader with an overview of AltaRica. We put the emphasis on the language for it plays a central role in the whole project. The section 2 is devoted to the main features of the language. We present constraint automata that are its mathematical basis. An illustrative example is presented section 3. Finally, section 4 discuss the problems raised by the compilation of AltaRica descriptions into Boolean formulae.

2. The AltaRica language

2.1. *Constraint Automata*

The mathematical foundation of the AltaRica language is the notion of constraint automata [BRL 94, COR 97]. Constraint automata are like finite state automata except that states are described implicitly by means of constraints over variables.

Intuitively, a constraint automaton is a set of transitions of the form

$$G(V) \xrightarrow{e} V = \sigma(V)$$

where V is a set of variables that take their values into a finite or infinite domain D, $G(V)$ is a constraint (or a guard, or a Boolean condition) over these variables, e is an event name and finally $V = \sigma(V)$ is an assignment of variables of V. A state of the automaton is just a valuation (with elements of D) of the variables of V. A transition $G(V) \xrightarrow{e} V = \sigma(V)$ means that if the automaton is in a state that verifies $G(V)$ and if the event e occurs, then it goes into the new state $\sigma(V)$.

The name “guarded automata” could be used instead of “constraint automata”. However, the term constraint refers here to algorithms used to handle set of states. These algorithms are actually inspired from constraint solving techniques (see for instance [HEN 89]).

As we shall see, several constraint automata can be synchronized in order to give a larger constraint automata. The synchronization mechanism is derived from the Arnold-Nivat model [ARN 82, ARN 94a].

AltaRica is basically a language to describe constraint automata. The language is hierarchical : each component describes a constraint automaton and components are combined together through the synchronization process. For the sake of the convenience, several features are added to the basic model :

- The description is hierarchical, as we just mentioned, and events may be synchronized.
- Priorities may be set among events.
- Variables are separated into two categories, namely states variables and flow variables. The formers are local to components. The latter's, that depend functionally on the formers, describe the interfaces of components.
- Some constraints may be set on the states that constrain the automaton to stay into a subset of $D^{|V|}$.

Formally, a *constraint automaton* is a tuple $\mathcal{A} = \langle D, S, F, E, T, A, I \rangle$ where :

- D is a finite or infinite domain.
- S and F is are two sets of variables (respectively called state variables and flow variables) such that $S \cap F = \emptyset$.
- E is a set of event names.
- T is a set of transitions. A transition is a triplet (g, e, a) where, $g \subseteq D^{|S \cup F|}$ is a constraint over $S \cup F$, called the *guard* of the transition, $e \in E$ and a is a mapping that defines the successor states : $a : D^{|S \cup F|} \rightarrow D^{|S|}$.
- $A \subseteq D^{|S \cup F|}$ is an assertion over the variables values.
- $I \subseteq D^{|S \cup F|}$ defines the set of initial states of the automaton.

```
node SwitchModel
 flow  f1,f2 : bool;
 event Open, Close;
 state open : bool;
 trans
   not open |- Open -> open := true;
   open     |- Close -> open := false;
 assert (not open) => (f1=f2);
edon
```

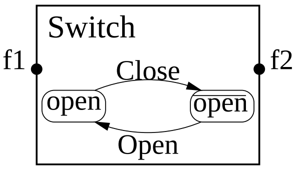


Figure 2. *An AltaRica model of a switch*

The states of an automaton are elements of $D^{|S \cup F|}$. A transition $t = (g, e, a)$ is *valid* in the state (s, f), which is denoted by $t \in valid(s, f)$, if :

– $(s, f) \in g$

– $s' = a(s, f) \wedge \exists f' \in D^{|F|}$, s.t. $(s', f') \in A$

A *state graph* is a tuple $G = (S, E, T, I)$, where S is a set of states, E is a set of events, $T \subseteq S \times E \times S$ is a set of transitions and $I \subseteq S$ is the set of initial states. A state graph characterizes all behaviors of a finite state machine. A *behavior* in the state graph is a sequence $s_0 t_0 s_1 t_1 \ldots$ such that :

– $s_0 \in I$ and $\forall i \geq 0$, $s_i \in S$.

– and $\forall i \geq 0, t_i = (s_i, e, s_{i+1}) \in T$.

A state s of G is *reachable* if there exists a behavior in G yielding to s.

Given a constraint automaton $\mathcal{A} = \langle D, F, E, S, T, A, I \rangle$, the state graph $G_{\mathcal{A}} = (S_{\mathcal{A}}, E_{\mathcal{A}}, T_{\mathcal{A}}, I)$ associated to $\mathcal{A}$ is a follows.

– $S_{\mathcal{A}} = D^{|S \cup F|}$, $E_{\mathcal{A}} = E$.

– $\langle (s, f), e, (s', f') \rangle \in T_{\mathcal{A}}$ if and only if

  – $(s, f) \in A$

  – $\exists (g, e, a) \in valid(s, f)$, $s' = a(s, f) \wedge (s', f') \in A$

– $I_{\mathcal{A}} = I \cap A$

Usually, $G_{\mathcal{A}}$ is assumed to be restricted to its reachable states.

The reader should notice that the flow variables are not directly assigned by the transitions ; their values are only defined through A by the values of state variables.

2.2. *Components*

As already said, an AltaRica description consists in a hierarchy of components. Each basic component describes a constraint automaton.

As an illustration, consider a simple switch which connects or disconnects two wires. The AltaRica description for such a switch is given on the Fig. 2.

The interface of the switch consists of two Boolean flow variables, `f1` and `f2`. A Boolean state variable, `open`, describes the position of the switch. Variables may be of different types : Boolean, integers, enumerations. The events, `Open` and `Close`, and the corresponding transitions describe the opening and the closure of the switch. Note that the event `Close` cannot occur when the switch is already closed. Finally, the assertion specifies how `f1` and `f2` depend on `open`. Guards and assertions are built using the usual Boolean connectives (and, or, not, imply, . . .), the arithmetic predicate

(=, <, ≤, . . .) and operators (+, ×, . . .).

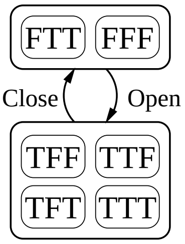


Figure 3. *State graph of the switch*

The state graph associated with the switch is depicted on the Fig. 3. On this figure, states with the same value for the state variable (the first component of triplets) are grouped. Note that the switch controls its flows only by means of a constraint. Therefore, values of flows may change even if the switch does "nothing".

2.3. *The time in AltaRica*

In AltaRica, as in synchronous languages (see [HAL 93] for a survey), the notion of physical (chronometrical) time is replaced by a simple notion of order among events.

The only relevant notions are simultaneity and precedence between events. The schema 4 shows the time evolution as view by an AltaRica system. When an event occurs some state variables of the system are modified. Then, instantaneously, all flow variables are updated from the new state variables values.

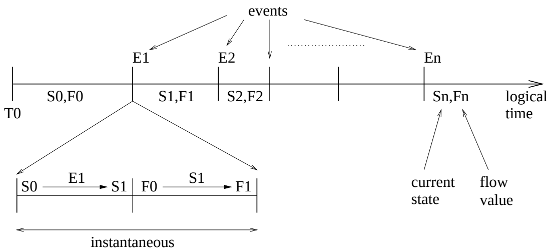


Figure 4. *Evolution of the time in AltaRica*

2.4. *Priorities*

It is often convenient to set priorities among transitions (or events). Consider, for instance, the above switch as a circuit-breaker. If a short-circuit occurs somewhere in the circuit, it is opened instantaneously, i.e. before any user action. The transition short-circuit is therefore of higher priority than user actions.

Formally, a *constraint automaton with priorities* is a couple $(\mathcal{A}, <)$ where $\mathcal{A}$ is a constraint automaton and $<$ is a partial order over events of $\mathcal{A}$. To be fire-able in a state (s, f), a transition $t = (g, e, a)$ of $(\mathcal{A}, <)$ must be not only valid in $\mathcal{A}$ but also *maximal*, i.e. such that :

$$\forall (g', e', a') \in valid(s, f), [e \neq e' \Rightarrow e \not< e'] \qquad (1)$$

Note that priorities act on fire-able transitions, i.e. that one first considers fire-able transitions and then selects those of highest priority.

Coming back to the switch, in order to model short-circuits, a Boolean flow variable `sc` and an event `Short-Circuit` are added to the description together with the following transition.

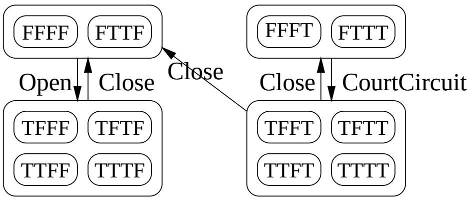


Figure 5. *State graph of the circuit-breaker*

```
sc and not open |- Court-Circuit -> open := true
```

The partial order `Open<Court-Circuit` and `Close<Court-Circuit`, is declared by associating an integer to each event. The higher the integer, the higher the priority. The state graph of the circuit-breaker is depicted on Fig. 5 (the fourth component of states is the value of `sc`).

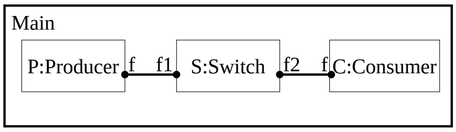
2.5. *Hierarchy*

As already said, an AltaRica description consists in a hierarchy of components. Components are combined together by two means : assertions and synchronizations. From a graphical point of view, a box corresponds to each level of the hierarchy. Wires between boxes (at the same level) denotes the presence of assertions that constrain component interfaces.

Let $\mathcal{A}_1, \ldots, \mathcal{A}_k$ be k constraint automata and let be A_s be an assertion over the flow variables of these automata. The *free product* of $\mathcal{A}_1, \ldots, \mathcal{A}_k$ is a constraint automata $\mathcal{A} = \langle D, S, F, E, T, A, I \rangle$, where D, S, F, E and T are the unions of respectively the domains, the state variables, the flow variables, the events and the transitions of the $\mathcal{A}_i$'s, A is the conjunction of the assertions of the A_i's together with the assertion A_s and finally I is the conjunction of the initial states of the $\mathcal{A}_i$'s.

As an illustration, the Fig. 6 depicts a system made of three components in series : a produced, a switch and a consumer. Two assertions constrain to be equal the output of the producer and the left flow of the switch on the one hand, the right flow of the switch and the input of the consumer on the other hand. These assertions are set at the system level.

When the switch is closed, the output of the producer is therefore constrained to be equal to the input of the consumer. This illustrates long distance interactions that are enabled by flow variables.



```
node Main
 sub S : Switch;
     P : Producer;
     C : Consumer;
edon
```

Figure 6. *Graphical and textual representation of a system*

2.6. *Synchronized products*

According to the AltaRica paradigm, events are assumed to represent diseases. Diseases are in general assumed to be independent one another and therefore not to occur simultaneously. There are some cases however where the same event has consequences on two different components. Another way to put that is to say that two distinct events occurring in two distinct components have to be simultaneous or that they are actually the same event occurring in two different places. This leads to the notion of synchronized product.

Let $\mathcal{A}_1, \ldots, \mathcal{A}_k$ be k constraint automata, let be A_s be an assertion over the flow variables of these automata, and let $\vec{v_1}$,..., $\vec{v_r}$ be r synchronization vectors. A synchronization vector contains at most one event per $\mathcal{A}_i$. The *synchronized product* $\mathcal{A} = \langle D, S, F, E, T, A, I \rangle$ of $\mathcal{A}_1, \ldots, \mathcal{A}_k$ w.r.t. to A_s and the $\vec{v_j}$'s is as their free product except that :

– First, the set E of is the union of the events of the $\mathcal{A}_i$'s not occurring in a $\vec{v_j}$, together with the $\vec{v_j}$'s.

– Second, the transition associated with a $\vec{v_j}$ is a triple $(g, \vec{v_j}, a)$ where g is the conjunction of the guards of the individual transitions labelled with the events occurring in $\vec{v_j}$ and a is the composition of the corresponding assignments.

Note that since it is assumed that synchronization vectors contains at most one event per component, this definition is correct even if two or more transitions of an $\mathcal{A}_i$ are labelled with the same event. In this case, there are several transitions labelled with the same vector in the synchronized product (these transitions are obtained by considering in some sense the Cartesian product of individual transitions).

Consider for instance, the switch described above and suppose that the system (depicted on the Fig. 6) contains a fourth component to model an user. The user may act on the switch in order to connect or to disconnect the consumer. It is described with two transitions that are labelled with events `OpenSwitch` and `CloseSwitch` :

```
node User
 event OpenSwitch, CloseSwitch;
 trans true |- OpenSwitch ->;
       true |- CloseSwitch ->;
edon
```

If we consider the following events “the switch opens” (`Open` in the model of the switch) and “the user opens the switch” (`OpenSwitch` in the model of the user) then it is a reasonable hypothesis to assume these events to be simultaneous. This synchronism is written as a vector as follows.

```
node Main
 sub  U : User; ...
 sync <U.OpenSwitch,S.Open>;
edon
```

In the synchronized product, the guard of transition labelled with the vector <`U.OpenSwitch,S.Open`> is the conjunction of the guards of the transition labelled with `OpenSwitch` of the user and the transition labelled `Open` of the switch. Similarly, the assignment of this transition is the concatenation of the assignments of these two transitions.

$$\begin{tikzcd}
 & \langle \mathcal{A}_1.a, \mathcal{A}_2.b, \mathcal{A}_3.c \rangle \arrow[dl] \arrow[dr] \arrow[dd] & \\
\langle \mathcal{A}_1.a, \mathcal{A}_2.b \rangle \arrow[dr] & & \langle \mathcal{A}_1.a, \mathcal{A}_3.c \rangle \arrow[dl] \\
 & \langle \mathcal{A}_1.a \rangle &
\end{tikzcd}$$

Figure 7. *The partial order induced by the broadcast vector* $\langle \mathcal{A}_1.a, \mathcal{A}_2.b?, \mathcal{A}_3.c? \rangle$

2.7. *Broadcast*

The synchronization mechanism presented above is somewhat rigid. This the reason why a weaker but more general mechanism, called *broadcasting*, is introduced in AltaRica. The broadcast mechanism is based on the notion of emitters and receivers (e.g. [HAR 86]). A component (the emitter) sends a message and the others react or not to this message. This is not a free choice : a receiver that can react must react. In AltaRica broadcast vectors, there may be an arbitrary number of "emitters" and "receivers" (including 0). The events that are allowed to be absent are tagged with a question mark ?.

Consider, for instance, the vector $\langle \mathcal{A}_1.a, \mathcal{A}_2.b?, \mathcal{A}_3.c? \rangle$. The events b of $\mathcal{A}_2$ and c of $\mathcal{A}_3$ are allowed to be absent. In other words, the broadcast vector $\langle \mathcal{A}_1.a, \mathcal{A}_2.b?, \mathcal{A}_3.c? \rangle$ is a syntactic short-cut for the four vectors $\langle \mathcal{A}_1.a, \mathcal{A}_2.b, \mathcal{A}_3.c \rangle$, $\langle \mathcal{A}_1.a, \mathcal{A}_2.b \rangle$, $\langle \mathcal{A}_1.a, \mathcal{A}_3.c \rangle$ and $\langle \mathcal{A}_1.a \rangle$ together with the priorities induced by the partial order $\subseteq$: if $\vec{v} \subseteq \vec{w}$, then $\vec{v}$ is of lower priority than $\vec{w}$. The partial order for $\langle \mathcal{A}_1.a, \mathcal{A}_2.b?, \mathcal{A}_3.c? \rangle$ is depicted on Fig. 7.

AltaRica allows a slight generalization of the above broadcast vectors : it is possible to constrain the number of tagged events that should occur to be a greater than a given constant. For instance, $\langle \mathcal{A}_1.a, \mathcal{A}_2.b?, \mathcal{A}_3.c? \rangle \geq 1$ means that at least one of the events b or c should occur (and therefore the vector $\langle \mathcal{A}_1.a \rangle$ is not allowed).

3. Reliability networks as a test case

As a test case, we consider here reliability networks. Reliability networks are one of the formalisms that are used in reliability studies to model physical systems in which an information (a message, a fluid, a current) propagates through a network whose components are subject to failures (see [SHI 91] for a detailed presentation of this formalism). As we shall see, reliability networks are quite difficult to handle and they are a good candidate to be a test case for AltaRica (and more generally for all of the tools with the same purpose).

3.1. *Reliability networks*

A reliability network is a graph, directed or not, with two distinguished vertices : a source vertex s and a target vertex t. Vertices as well as edges are subject to failures (that cut them off). They are assumed to fail independently according to known probability laws.

Two questions may be asked about a reliability network :

1. What are the minimal (for set inclusion) s-t paths ?
2. What is the probability that there is at least one working s-t paths.

As one may expect, the second problem is #P-complete and even approximations are hard to compute [BAL 86].

We shall assume here, without a loss of generality, that vertices are perfect, i.e. that only edges may fail.

At a first glance, the problem could seem simple to model :

– If a vertex is supplied by one of its in-edges then all of its out-edges are supplied as well.
– If an edge is working then its two adjacent vertices are either both supplied or both not supplied, otherwise there is *a priori* no relationship between its adjacent edges.

The pitfall stands in strongly connected components. If such a set of vertices contains the source vertex, then it is supplied. If it does not contain the source vertex, then, within the above model, it can be either supplied or not supplied, a non-sense. The problem is therefore to compel it to be not supplied in the latter case.

3.2. *Description using AltaRica*

In order to describe reliability networks within AltaRica, we shall design an AltaRica node for each type of components : edges, source vertex and other vertices (the target vertex does not differ from the other vertices).

A small reliability network together with its AltaRica graphical description is pictured Fig. 8.

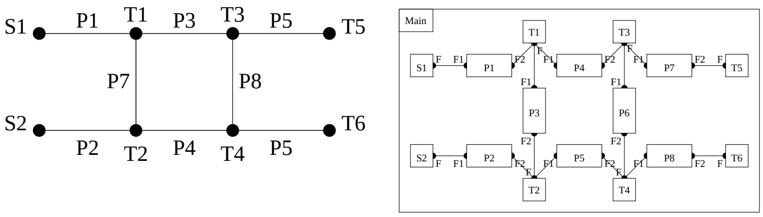


Figure 8. *A reliability network and its graphical AltaRica representation*

Now, the idea is to use broadcasting to compel strongly connected components that do not contain the source vertex to be unsupplied. Each time an edge fails, the vertices try to go into a state in which they are not supplied. If a vertex is still connected to the source vertex, such a transition is not allowed.

The skeleton for the AltaRica description we sketched is given Fig. 9.

Another solution consists in defining a broadcast vector `Update` made only of the `Reaction`'s of the vertices (with a constraint to ensure that at least one of the vertices

```
node Source
 flow F : bool;
 assert F = true;
edon

node Vertex
 flow  F : bool;
 state supplied : bool;
 event Reaction;
 assert supplied = F;
 trans
   supplied |- Reaction -> supplied := false;
edon

node Edge
 flow  F1, F2 : bool;
 state broken : bool;
 event Failure;
 assert (not broken) => (F1 = F2);
 trans
   not broken |- Failure -> broken := true;
edon

node Main
sub S : Source;
    V1,V2,...,Vk : Vertex;
    E1,E2,...,Em : Edge;
assert S.F = E1.F1;
       E1.F2 = V1.F;
       ...
sync <E1.Failure,V1.Reaction?,...,Vk.Reaction?>;
     ...
edon
```

Figure 9. *Skeleton of an AltaRica description for reliability networks*

reacts) and to assign to this vector a high priority. In this way, failures and information propagation through the network are separated.

4. Compilation of AltaRica descriptions in Boolean formulae

One of the key issue for the AltaRica project is the ability to design an efficient compiler to translate AltaRica descriptions into Boolean formulae. Since any AltaRica description can be flattened into a constraint automaton, the problem is to compile the textual description of a such an automaton $\mathcal{A}$ into a Boolean formula $\phi_{\mathcal{A}}$ that verifies the following properties.

- The input variables of $\phi_{\mathcal{A}}$ are the events of $\mathcal{A}$.
- The prime implicants of $\phi_{\mathcal{A}}$ correspond one to one to the minimal *scenarii* of failure described by $\mathcal{A}$, i.e. the minimal paths in the state graph associated with $\mathcal{A}$ that go from the initial state to a failure state.

Indeed, this compilation process looses the schedule among events. This is the price to pay to be able to compile a dynamic description into a static one. Now, dealing with a Boolean formula rather than with a constraint automaton presents many advantages :

- Probabilistic assessments can be performed in a very efficient way once the Binary Decision Diagram that encodes this formula is computed (see for instance [DUT 99c, DUT 99b]).
- *Scenarii* of failure can be handled in a very efficient way as well by means of Zero-suppressed BDDs (see for instance [DUT 97, DUT 99a].

Basically, the compilation is achieved by means of the following algorithm.

1. Compute the state graph $G_{\mathcal{A}}$ of $\mathcal{A}$.

2. Consider G as reliability network, i.e. apply any algorithm that determines $s - t$ paths (see for instance [SHI 91] for a survey on these algorithms), where s is an initial state of $\mathcal{A}$ and t is any failure state.

If the state graph $G_{\mathcal{A}}$ contains no loop, it is pretty easy to compile it into a Boolean formula (in this case, $G_{\mathcal{A}}$ can be seen a block diagram model [AND 93]). If G does contain loops, its compilation into a set of Boolean equations is much more difficult, even if some recent works provide new ideas to handle that problem [MAD 94, DUT 96].

The above algorithm is roughly inefficient for it is well known that the state graph is often huge, even for small size systems. Several techniques, such as partial orders [GOD 96], can be used to reduce the graph actually considered.

5. Conclusion

In this paper, we presented an overview the AltaRica language. Its expressiveness makes it suitable to perform reliability and dependability of critical systems. Moreover, its sound and clear semantics in terms of constraint automata allows its compilation into the lower level formalisms such as fault trees, Petri nets or finite state machines. This opens perspectives for AltaRica to be used as a support language of both functional and dysfunctional analyses of critical systems.

There remains many works to do around AltaRica. Among them are the improvement of compilation algorithms, the design of an AltaRica based model-checker or the introduction of real time. The AltaRica workbench gives us the opportunity to get feedback from the industrial partners of the project. We consider this aspect as mandatory in order to ensure the adequacy of the language to real-life studies.

The AltaRica project

The AltaRica project federates works done at the LaBRI on both reliability analyses and formal methods. From a scientific point of view, our ambition is to develop a corpus of algorithms and tools to assess efficiently reliability models. This requires also to consider methodological issues on the different way a real-life system may be modeled.

From an industrial point of view, the AltaRica project is supported by a group of industrial companies, including ELF-Aquitaine, CEA, Dassault-Aviation, Thomson-CSF and Renault.

Acknowledgments

This work is supported by the Altarica Group and receives a grant of the European funds FEDER OBJECTIVE 2.

We would like to thank here A. Arnold and A. Griffault. André and Alain can be considered as co-authors of the present article, since the definition of the AltaRica language and its semantics is a joined work with them.

Bibliographie

[AND 93] ANDREWS J. et MOSS T., *Reliability and Risk Assessment*. John Wiley and Sons, 1993.

[ARN 82] ARNOLD A. et NIVAT M., « Comportements de processus ». *Colloque AFCET "Les Mathématiques de l'informatique"*, 1982.

[ARN 94a] ARNOLD A., *Finite Transition Systems*. Prentice-Hall, 1994.

[ARN 94b] ARNOLD A., BÉGAY D. et CRUBILLÉ P., *Construction and analysis of transition systems with MEC*. World Scientific Publishers, 1994.

[BAL 86] BALL M., « Computational complexity of network reliability analysis : an overview ». *IEEE Transactions on Reliability*, vol. R-35, p. 230–239, 1986.

[BON 97] BONFATTI F., MONARI P. et SAMPIERI U., *IEC 1131-3 Programming Methodology*. ISBN 2-9511585-0-5, 1997.

[BOU 93] BOUISSOU M., « The FIGARO Dependability Evaluation Workbench in Use : Case Studies for Fault-Tolerant Computer Systems ». *23th Annual Symposium on Fault Tolerant Computing, FTCS'93*, 1993.

[BRL 94] BRLEK S. et RAUZY A., « Synchronization of Constrained Transition Systems ». HONG H., Ed., *Proceedings of the First International Symposium on Parallel Symbolic Computation (PASCO'94)*, p. 54–62, Linz, Ostreich, 1994. World Scientific Publishing.

[COR 97] CORSINI M.-M. et RAUZY A., « Toupie : the μ-calculus over finite domains as a constraint language ». *Journal of Automated Reasonning*, vol. 17, p. 143–171, 1997.

[DUT 96] DUTUIT Y., RAUZY A. et SIGNORET J.-P., « Réséda : a Reliability Network Analyser ». CACCIABUE C. et PAPAZOGLOU I., Eds., *Proceedings of European Safety and Reliability Association Conference, ESREL'96*, vol. 3, p. 1947–1952. Springer Verlag, 1996. ISBN 3-540-76051-2.

[DUT 97] DUTUIT Y. et RAUZY A., « Exact and Truncated Computations of Prime Implicants of Coherent and non-Coherent Fault Trees within Aralia ». *Reliability Engineering and System Safety*, vol. 58, p. 127–144, 1997.

[DUT 99a] DUTUIT Y. et RAUZY A., « A Guided Tour of Minimal Cutsets Handling by means of Binary Decision Diagrams ». *Proceedings of Probabilistic Safety Assessment conference, PSA'99*, 1999. To appear.

[DUT 99b] DUTUIT Y. et RAUZY A., « New algorithms to compute importance factors CPr, MIF, CIF, DIF, RAW and RRW ». *Proceedings of the European Safety and Reliability Association Conference, ESREL'99*. European Safety and Reliability Association, 1999. to appear.

[DUT 99c] DUTUIT Y., RAUZY A. et SIGNORET J.-P., « Evaluation of Systems Reliability by means of Binary Decision Diagram ». *Proceedings of the Probabilistic Safety Assessment Conference, PSA'99*, 1999. to appear.

[GOD 96] GODEFROID P., *Partial-Order Methods for the Verification of Concurrent Systems*, vol. 1032. LNCS, 1996. ISBN 3-540-60761-1.

[HAL 93] HALBWACHS N., *Synchronous programming of reactive systems*. Kluwer Academic Publishers, 1993.

[HAR 86] HAREL D., PNUELI A., SCHMIDT J. et SHERMAN R., « On the Formal Semantics of Statecharts ». *Proceeding of the First IEEE Symposium on Logic in Computer Science*, p. 54–64, New-York, 1986. IEEE Press.

[HAR 87] HAREL D., « StateCharts : a visual approach to complex systems ». *Science of Computer Programming*, vol. 8, p. 231–275, 1987.

[HEN 89] VAN HENTENRYCK P., *Constraint Satisfaction in Logic Programming*. Logic Programming Series. MIT Press, 1989. ISBN 0-262-08181-4.

[HUT 94] HUTINET T., LAJEUNESSE S. et MARTIN L., « Atelier FIABEX, vers une intégration des études SdF en phase de conception ». *Actes du Congrés $\lambda\mu$ 94, ESREL'94*, p. 694–700, La Baule, 1994.

[MAD 94] MADRE J.-C., COUDERT O., FRAÏSSÉ H. et BOUISSOU M., « Application of a New Logically Complete ATMS to Digraph and Network-Connectivity Analysis ». *Proceedings of the Annual Reliability and Maintainability Symposium, ARMS'94*, p. 118–123, 1994. Annaheim, California.

[MAR 91] MARANINCHI F., « The Argos language : Graphical Representation of Automata and Description of Reactive Systems ». *IEEE Workshop on Visual Languages*, Kobe, Japan, October 1991.

[SHI 91] SHIER D., *Network Reliability and Algebraic Structures*. Oxford Science Publications, 1991.

[SIG 95] SIGNORET J.-P., « Moca-RP V9 ». Rapport technique, Elf-Aquitaine, 1995. rapport interne ELF Aquitaine Production – Direction Recherche et Développement Exploration Production – réf. EP/P/SE/MRT-ARF/JPS9634 – simulation de Monte-Carlo de réseaux de Petri stochastiques.

[TUR 93] TURNER K., *Using Formal Description Techniques*. John Wiley & Sons, 1993. ISBN 0-471-93455-0.