Enhancement of the AltaRica 3.0 stepwise simulator by introducing an abstract notion of time

Michel Batteux, Tatiana Prosvirnova, Antoine Rauzy

Enhancement of the AltaRica 3.0 stepwise simulator by introducing an abstract notion of time

M. Batteux
IRT SystemX
Palaiseau, France

T. Prosvirnova
LGI
CentraleSupélec, Gif-sur-Yvette, France

A. Rauzy
MTP
Norwegian University of Science and Technology, Trondheim, Norway.

ABSTRACT: AltaRica 3.0 is an event-based, object-oriented modeling language dedicated to (probabilistic) safety analyses of complex systems. It makes it possible to design models at higher level than done with formalisms traditionally used for safety analyses (fault trees, Markov Chains, stochastic Petri nets, etc.), without increasing the complexity of calculations of risk indicators. Several assessment tools have been developed for AltaRica 3.0, including a stepwise simulator. This tool is of a great help for the design and the validation of AltaRica 3.0 models. It is the analog for modeling of debuggers for programming.
In this article, we show how the AltaRica 3.0 stepwise simulator has been greatly enhanced by the introduction of an abstract notion of time. The key mathematical property is that abstract and concrete simulation are bisimilar: any concrete (timed, stochastic) execution can be simulated by an abstract execution and reciprocally any abstract execution corresponds to at least one concrete execution. This important result paves the way to the design of efficient model-checking algorithms, e.g. generators of sequences of events leading to a failure state.

1 INTRODUCTION

AltaRica 3.0 is an event-based, object-oriented modeling language dedicated to (probabilistic) safety analyses of complex systems (Prosvirnova, Batteux, Brameret, Cherfi, Friedlhuber, Roussel, & Rauzy 2013). It makes it possible to design models at higher level than done with formalisms traditionally used for safety analyses (fault trees, Markov Chains, stochastic Petri nets, etc.), without increasing the complexity of calculations of risk indicators.

The semantics of AltaRica 3.0 is defined in terms of stochastic guarded transition systems (Rauzy 2008). AltaRica 3.0 executions are similar to those of other discrete event modeling formulisms: each time a transition gets fireable, it is scheduled and possibly fired after a certain real-valued delay, see e.g. (Cassandras & Lafortune 2008, Zimmermann 1976) for introductions to (stochastic) discrete event systems. Events labeling transitions may be either deterministic or stochastic. In the later case, AltaRica 3.0 provides both built-in distributions (exponential, Weibull, ...) and empirical distributions.

Several assessment tools have been developed for AltaRica 3.0, see e.g. (Prosvirnova & Rauzy 2015, Brameret, Rauzy, & Roussel 2015, Aupetit, Batteux, Rauzy, & Roussel 2015), including a stepwise simulator. This tool is of a great help for the design and the validation of AltaRica 3.0 models. It makes it possible to perform interactive step by step simulations, i.e. to go forth and back in sequences of events, enabling in this way to track modeling errors, unexpected behaviors and so on. With that respect, stepwise simulators play a similar role for discrete event modeling as debuggers like GDB or DDD for programming, see e.g. (Matloff & Salzman 2008) for an introduction to the latter.

In this article, we show how the AltaRica 3.0 stepwise simulator has been greatly enhanced by the introduction of an abstract notion of time. So far, it did not consider the time at all. The reason was that it would have been much too tedious for the analyst to

enter by hand the delay associated with a stochastic transition each time this transition gets fireable. Moreover, infinitely many real-valued delays can be chosen, letting the analyst pondering which one is the most suitable for her or his purpose. However, ignoring delays had a major drawback: the stepwise simulator allowed the firing of sequences of events with no counterpart in stochastic simulation and more generally that did not obey the timed semantics of AltaRica 3.0.

The idea is therefore to abstract away the time in stepwise simulation: each transition is now associated with a time interval. Firing a transition may modify the time intervals associated with already scheduled transitions. This idea is by no means new: it enters into the general framework of Cousot's abstract interpretation (Cousot & Cousot 1977). The problem at stake was to make it work for the particular case of stochastic discrete event simulations. The key mathematical property here is that abstract and concrete simulations are bisimilar, see e.g. (Milner 1989) for an introduction to this important notion: any concrete (timed, stochastic) execution can be simulated by an abstract execution and reciprocally any abstract execution corresponds to at least one concrete execution. In a word, abstract executions are in agreement with AltaRica 3.0 semantics.

This important result paves the way to the design of efficient model-checking algorithms, see e.g. (Clarke, Grumberg, & Peled 2000) for an introduction. In particular, it makes it possible the design of generators of sequences of events leading to a failure state.

The remainder of this article is organized as follows. Section 2 presents an illustrative example. Section 3 recalls fundamental notions about timed and stochastic guarded transition systems. Section 4 introduces the abstract notion of time. Section 5 concludes this article and gives some perspectives.

2 ILLUSTRATIVE EXAMPLE

As an illustrative example, we shall consider a system made of two identical, periodically tested, components that evolve independently one another.

The behavior of such a component is represented by the state diagram pictured Figure 1.

The component alternates operation and test phases. It is initially working and starts with a first operation phase that lasts a constant time θ. All subsequent operation phases last a constant time π.

The component may fail when in operation, with a failure rate λ.

If the component is working when it enters a maintenance phase, this maintenance phase lasts a constant time τ. If, on the contrary, it is failed when it enters a maintenance phase, the duration of its repair is uniformly distributed between two values μ and ν.

Finally, the component is as-good-as-new after a repair.

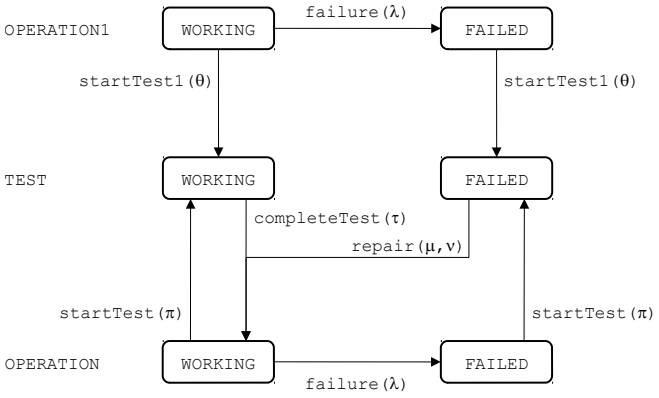

Figure 1: State diagram for a periodically tested component.

In the state diagram of Figure 1, transitions `failure` and `repair` are thus stochastic while transitions `startTest1`, `startTest` and `completeTest` are deterministic.

Table 1 shows a possible evolution of such a component.

Table 1: A possible evolution of component for a periodically tested component.

Transition	Firing date
`startTest1`	$d_1 = \theta$
`completeTest`	$d_2 = d_1 + \tau$
`failure`	$d_3 = d_2 + \delta_1,\ 0 < \delta_1 \leq \pi$
`startTest`	$d_4 = d_2 + \pi$
`repair`	$d_5 = d_4 + \delta_2,\ \mu \leq \delta_2 \leq \nu$
`startTest`	$d_6 = d_5 + \pi$
`completeTest`	$d_7 = d_6 + \tau$
$\vdots$	$\vdots$

We can assume that θ and π are relatively big compared to τ, μ and ν and that τ is smaller than μ (itself smaller than μ).

In order not to have the two components `A` and `B` out of service due to test or maintenance at the same time, it is reasonable to take different values of θ for `A` and for `B` (the values of the other parameters being identical for the two components). For instance, if an operation phase lasts normally six months ($\pi = 4380h$), the component `A` can be tested after three months ($\texttt{A}.\theta = 2190h$) while the component `B` is tested after six months ($\texttt{B}.\theta = 4380h$). It this way, tests/maintenances of `A` and `B` are shifted by three months which improves the overall availability of the system. Table 2 gives some typical values of the parameters for components `A` and `B` that we shall use throughout the article.

Table 2: Typical values of the parameters

parameter	`A`	`B`
θ	2190	4380
π	4380	4380
τ	0	0
μ	12	12
ν	24	24

With this values of parameters in mind, the reader sees immediately the problem of using a stepwise simulator that does not consider delays associated to transitions. Many executions that would be impossible with a timed semantics become possible with a non-timed semantics, e.g.

$$\cdot \xrightarrow{\texttt{B.startTest1}} \ldots$$

$$\cdot \xrightarrow{\texttt{A.startTest1}} \cdot \xrightarrow{\texttt{A.completeTest}} \cdot \xrightarrow{\texttt{A.startTest}} \cdot \xrightarrow{\texttt{A.completeTest}} \ldots$$

On other hand, asking the analyst to introduce interactively delays of stochastic transitions is not a practical solution. Not only it would be tedious, but it would let the analyst facing the choice of suitable delays, which gets quickly puzzling. Hence the need of an abstract notion of time which makes it possible to take into account delays of transitions without asking the analyst to enter them interactively.

To fulfill this need, the idea is to reason in terms of time intervals rather than in terms of dates. To explain how this idea works, we shall first recall the regular semantics of AltaRica in the next section. Then, we shall introduce its abstract semantics (in terms of time intervals) in Section 4.

3 TIMED/STOCHASTIC GUARDED TRANSITIONS SYSTEMS

The semantics of AltaRica 3.0 is defined in terms of stochastic guarded transitions systems (Rauzy 2008), (Batteux, Prosvirnova, & Rauzy 2017). We shall recall here only the notions that are important for the purpose of this article. The reader should refer to the cited articles for in depth presentations.

3.1 *Definition*

A guarded transitions system is a quintuple $\langle V, E, T, A, \iota \rangle$, where:

- V is a set of variables. V is the disjoint union of the set S of state variables and the set F of flow variables: $V = S \uplus F$. Each variable v of V takes its value into a finite or infinite set of constants called the domain of v and denoted as $dom(v)$. The global state of the system is thus a variable valuation, i.e. a member of the Cartesian product $\prod_{v \in V} dom(v)$.
- E is a set of events. Each event e of E is associated with a function $delay(e)$ that returns a non-negative real number. $delay(e)$ may be deterministic, in which case it returns always the same value, or stochastic, in which case it returns a value according to a certain cumulative probability distribution.
- T is a set of transitions, i.e. of triples $\langle e, G, P \rangle$, where e is an event of E, G is a Boolean expression built on variables of V (called the guard of the transition) and P is an instruction that modifies the value of state variables (called the action of the transition). For the sake of the clarity, we shall write a transition $\langle e, G, P \rangle$ as $G \xrightarrow{e} P$.
- A is an assertion, i.e. an instruction that modifies the values of flow variables.
- ι is a valuation of the variables of V, called the initial state.

Example The GTS encoding the periodically tested components described in the previous section is as follows.

The state of the component is represented by means of two state variables: `state` that takes its value in $\{\texttt{WORKING}, \texttt{FAILED}\}$ and `phase` that takes its value in $\{\texttt{OPERATION1}, \texttt{TEST}, \texttt{OPERATION}\}$.

The events are `startTest1`, `startTest`, `completeTest`, `failure` and `repair`. They are associated with the delays described in the previous section.

The transitions are as follows.

$$\texttt{state} = \texttt{WORKING} \wedge \texttt{phase} \neq \texttt{TEST} \xrightarrow{\texttt{failure}} \texttt{state} \leftarrow \texttt{FAILED}$$

$$\texttt{phase} = \texttt{OPERATION1} \xrightarrow{\texttt{startTest1}} \texttt{phase} \leftarrow \texttt{TEST}$$

$$\texttt{phase} = \texttt{OPERATION} \xrightarrow{\texttt{startTest}} \texttt{phase} \leftarrow \texttt{TEST}$$

$$\texttt{state} = \texttt{WORKING} \wedge \texttt{phase} = \texttt{TEST} \xrightarrow{\texttt{completeTest}} \texttt{phase} \leftarrow \texttt{OPERATION}$$

$$\texttt{state} = \texttt{FAILED} \wedge \texttt{phase} = \texttt{TEST} \xrightarrow{\texttt{repair}} \texttt{state} \leftarrow \texttt{WORKING}, \texttt{phase} \leftarrow \texttt{OPERATION}$$

Finally, the initial state is defined by the variable valuation: $\texttt{state} = \texttt{WORKING}, \texttt{phase} = \texttt{OPERATION1}$.

3.2 *Composition*

One of the advantages of guarded transitions systems over some other similar formalisms is that they are highly compositional.

Formally, let $M_1 : \langle V_1, E_1, T_1, A_1, \iota_1 \rangle$ and $M_2 : \langle V_2, E_2, T_2, A_2, \iota_2 \rangle$ be two guarded transitions systems. Then the composition of M_1 and M_2, denoted as $M_1 \otimes M_2$, is simply the guarded transitions system $\langle V, E, T, A, \iota \rangle$ such that $V = V_1 \cup V_2$, $E = E_1 \cup E_2$, $T = T_1 \cup T_2$, $A = A_2 \circ A_1$ and $\iota = \iota_2 \circ \iota_1$.

The above principle extends to any number of guarded transitions systems.

Example To represent the system of discussed example in the previous section, it suffices to create two copies of the above guarded transitions system and to compose them (which is done automatically by the AltaRica compiler).

In our example, it may be worth to introduce a flow Boolean variable `failed` to tell when the system is failed. The assertion defining this variable could be as follows.

$\texttt{failed} \leftarrow \texttt{A.state} = \texttt{FAILED} \wedge \texttt{B.state} = \texttt{FAILED}$

3.3 *Semantics*

The semantics of a guarded transitions system $S : \langle V, E, T, A, \iota \rangle$ is defined as the set of its possible executions.

To define formally the executions, we need to introduce the notion of schedule. A schedule of $S : \langle V, E, T, A, \iota \rangle$ is a function from T to $\mathbb{R}^+ \cup \{+\infty\}$.

A schedule Γ is compatible with a state σ of the guarded transitions system and a date d if the following conditions hold for all transitions $t : G \xrightarrow{e} P$ of T.

- $d \leq \Gamma(t) < +\infty$ if $G(\sigma) = true$.
- $\Gamma(t) = +\infty$ if $G(\sigma) = false$.

Intuitively, an execution of S is a sequence:

$$\langle \sigma_0, d_0, \Gamma_0 \rangle \xrightarrow{t_1} \langle \sigma_1, d_1, \Gamma_1 \rangle \xrightarrow{t_2} \ldots \xrightarrow{t_n} \langle \sigma_n, d_n, \Gamma_n \rangle$$

where $n \geq 0$, the σ_i's are states of S, the d_i's are dates i.e. non negative real numbers verifying $0 = d_0 \leq d_1 \leq \ldots \leq d_n$, each Γ_i is a schedule compatible with σ_i and d_i and finally the t_i's are transitions of S.

The set of valid executions is defined recursively as follows.

The empty execution $\langle \iota, 0, \Gamma_0 \rangle$ is a valid execution if the schedule Γ_0 is such that for all transitions $t : G \xrightarrow{e} P$ of T:

- $\Gamma_0(t) = delay(e)$ if $G(\iota) = true$.
- $\Gamma_0(t) = +\infty$ if $G(\iota) = false$.

Now, if $\Lambda = \langle \sigma_0, d_0, \Gamma_0 \rangle \xrightarrow{t_1} \ldots \xrightarrow{t_n} \langle \sigma_n, d_n, \Gamma_n \rangle$, $n \geq 0$, is a valid execution, then so is the execution $\Lambda \xrightarrow{t_{n+1}} \langle \sigma_{n+1}, d_{n+1}, \Gamma_{n+1} \rangle$ if the following conditions hold, assuming $t_{n+1} = G_{n+1} \xrightarrow{e_{n+1}} P_{n+1}$.

- $G_{n+1}(\sigma_n) = true$.
- $\sigma_{n+1} = A(P_{n+1}(\sigma_n))$, i.e. the firing of the transition t_{n+1} is performed in two steps: first, state variables are updated by means of the action P_{n+1} of the transition, then flow variables are updated by means of the assertion A.
- $d_{n+1} = \Gamma_n(t_{n+1})$ and there is no transition t of T such that $\Gamma_n(t) < \Gamma_n(t_{n+1})$.
- Γ_{n+1} is obtained from Γ_n by applying the following rules to all transitions $t : G \xrightarrow{e} P$ of T.
  - If $G(\sigma_{n+1}) = true$, then:
    - If $G(\sigma_n) = true$ and $t \neq t_{n+1}$, then

      $\Gamma_{n+1}(t) = \Gamma_n(t)$

    - Otherwise,

      $\Gamma_{n+1}(t) = d_{n+1} + delay(e)$

  - If $G(\sigma_{n+1}) = false$, then:

    $\Gamma_{n+1}(t) = +\infty$

Example Consider again our system of two components.

At time 0, 4 transitions are fireable:

Transition	Firing date
`A.startTest1`	2190
`A.failure`	5617
`B.startTest1`	4380
`B.failure`	4111

As `A.startTest` has the earliest firing date, it is fired (at 2190). After its firing, 3 transitions are fireable:

Transition	Firing date
`A.completeTest`	$2190 + 0 = 2190$
`B.startTest1`	4380
`B.failure`	4111

As `A.completeTest` has the earliest firing date, it is fired (at 2190). After its firing, 4 transitions are fireable:

Transition	Firing date
`A.startTest`	$2190 + 4380 = 6570$
`A.failure`	$2190 + 6020 = 8210$
`B.startTest1`	4380
`B.failure`	4111

As `B.failure` has the earliest firing date, it is fired (at 4111). After its firing, 3 transitions are fireable:

Transition	Firing date
`A.startTest`	6570
`A.failure`	8210
`B.startTest1`	4380

As `B.startTest1` has the earliest firing date, it is fired (at 4380). After its firing, 3 transitions are fireable:

Transition	Firing date
`A.startTest`	6570
`A.failure`	8210
`B.repair`	4400

As `B.repair` has the earliest firing date, it is fired (at 4400). After its firing, 4 transitions are fireable:

Transition	Firing date
`A.startTest`	6570
`A.failure`	8210
`B.startTest`	$4400 + 4380 = 8780$
`B.failure`	$4400 + 5201 = 9601$

And so on...

This sequence shows how deterministic and stochastic transitions can be intricated. In particular, dates of tests are not decided once for all. They depend on times to failure and to repair of the component.

4 ABSTRACT SEMANTICS

4.1 *Principle*

The first idea to abstract the executions consists in associating an abstract delay $delay^\star$ with each event of the model. $delay^\star(e)$ is simply the image of the function $delay$, i.e. an interval of non-negative real numbers.

We have to be a bit careful because some intervals that are the images of distributions are closed while some others are open (to the left and/or to the right) and that we have to consider infinite bounds. A solution consists in working only with closed intervals, but in a non-standard arithmetic built over the set $\overline{\mathbb{R}}^+ = \mathbb{R}^+ \cup \{\epsilon, \infty\}$, where ϵ and ∞ are respectively infinitely small and infinitely big numbers verifying: $\epsilon + \epsilon = \epsilon$ and $\infty + x = \infty$ for all $x \in \overline{\mathbb{R}}^+$. In this way, the interval $]a, b[$, $a, b, \in \mathbb{R}^+$, can be encoded as $[a + \epsilon, b - \epsilon]$. Table 3 gives the abstract delays associated with the most widely used distributions in AltaRica 3.0. Moreover, a transition whose guard is not satisfied in the current state is scheduled in the interval $[\infty, \infty]$.

Table 3: Intervals associated with delay functions

Concrete delay	Abstract delay
Dirac(t)	$[t, t]$
UniformDeviate(l, h)	$[l, h]$
Exponential(λ)	$[0 + \epsilon, \infty]$
Weibull(α, β)	$[0 + \epsilon, \infty]$
Empirical distribution	$[0 + \epsilon, \infty]$

The second idea is to consider not the date at which transitions are fired, but an interval of time within which they are fired.

Assume that we are building the sequence under study step by step and that the last transition we considered must be fired in the time interval $[l, h]$. Assume moreover that transitions $t_1, t_2, \ldots t_n$ are scheduled in time intervals $[l_1, h_1]$, $[l_2, h_2]$, ..., $[l_n, h_n]$. Then, we can make the following remarks.

1. We must have $l \leq l_i$ for all $i = 1, \ldots, n$, because the next transition cannot be scheduled in the past.

2. We must have also $h \leq h_i$ for all $i = 1, \ldots, n$, because if $h_i < h$ for some i, it means that the transition t_i must be fired before h_i, therefore the last transition must also be fired before h_i.

3. For the same reason, we can choose t_i as the next transition to be fired only if there is no other transition t_j such that $h_j < l_i$.

4. Again for the same reason, if the transition t_i is fired, it is necessarily fired in the interval $[l_i, h_{min}]$, where h_{min} is the smallest of the h_j's.

5. If the transition t_i is fired and the transition t_j is such that $l_j < l_i$, then l_j must be changed to l_i so to obey our first remark.

6. Finally, if the transition t_i is fired and a transition t associated with the interval $[l_t, h_t]$ becomes fireable (t can be the transition t_i itself), then t must be scheduled in the interval $[l_i, h_{min}] + [l_t, h_t] = [l_i + l_t, h_{min} + h_t]$.

We are now able to define formally the abstract semantics of guarded transitions systems (and therefore for AltaRica 3.0).

4.2 *Formal Definition*

The abstract semantics of a guarded transitions system $S : \langle V, E, T, A, \iota \rangle$ is defined as the set of its possible abstract executions.

To define formally the abstract executions, we need to introduce the notion of abstract schedule. An abstract schedule of $S : \langle V, E, T, A, \iota \rangle$ is a function from T to closed intervals over $\overline{\mathbb{R}}^+$.

A schedule $\Gamma^\star$ is compatible with a state σ of the guarded transitions system and the abstract date $[l, h]$ if the following conditions hold for all transitions $t : G \xrightarrow{e} P$ of T, with $\Gamma^\star(t) = [l_t, h_t]$.

- $l \leq l_t < \infty$ and $h \leq h_t$ if $G(\sigma) = true$.
- $l_t = h_t = \infty$ if $G(\sigma) = false$.

An abstract execution of S is a sequence:

$$\langle \sigma_0, d_0^\star, \Gamma_0^\star \rangle \xrightarrow{t_1} \langle \sigma_1, d_1^\star, \Gamma_1^\star \rangle \xrightarrow{t_2} \ldots \xrightarrow{t_n} \langle \sigma_n, d_n^\star, \Gamma_n^\star \rangle$$

where $n \geq 0$, the σ_i's are states of S, the $d_i^\star$'s are abstract dates, i.e. time intervals, each $\Gamma_i^\star$ is an abstract schedule compatible with σ_i and $d_i^\star$ and finally the t_i's are transitions of S.

The set of valid executions is defined recursively as follows.

The empty abstract execution $\langle \sigma_0, [0, 0], \Gamma_0^\star \rangle$ is a valid abstract execution if the abstract schedule $\Gamma_0^\star$ is such that for all transitions $t : G \xrightarrow{e} P$ of T:

- $\Gamma_0^\star(t) = delay^\star(e)$ if $G(\iota) = true$.
- $\Gamma_0^\star(t) = [\infty, \infty]$ if $G(\iota) = false$.

If $\Lambda = \langle \sigma_0, [0,0], \Gamma_0^\star \rangle \xrightarrow{t_1} \ldots \xrightarrow{t_n} \langle \sigma_n, [l_n, h_n], \Gamma_n^\star \rangle$, $n \geq 0$, is a valid abstract execution, then so is the abstract execution $\Lambda \xrightarrow{t_{n+1}} \langle \sigma_{n+1}, [l_{n+1}, h_{n+1}], \Gamma_{n+1}^\star \rangle$ if the following conditions hold, assuming $t_{n+1} = G_{n+1} \xrightarrow{e_{n+1}} P_{n+1}$, $\Gamma_n^*(t_{n+1}) = [l^\star, h^\star]$ and

$$h_{min} = \min_{[l,h]=\Gamma_n^\star(t), t \in T} h.$$

- $G_{n+1}(\sigma_n) = true$.
- $\sigma_{n+1} = A(P_{n+1}(\sigma_n))$.
- There is no transition t of T such that $\Gamma_n^\star(t) = [l,h]$ and $h < l^\star$.
- $[l_{n+1}, h_{n+1}] = [l^\star, h_{min}]$.
- $\Gamma_{n+1}^\star$ is obtained from $\Gamma_n^\star$ by applying the following rules to all transitions $t : G \xrightarrow{e} P$ of T and $\Gamma_n^\star(t) = [l,h]$.
  - If $G(\sigma_{n+1}) = true$, then:
    - If $G(\sigma_n) = true$ and $t \neq t_{n+1}$, then

      $\Gamma_{n+1}^\star(t) = [max(l_{n+1}, l), h]$
    - Otherwise,

      $\Gamma_{n+1}^\star(t) = [l_{n+1}, h_{n+1}] + delay^\star(e)$
  - If $G(\sigma_{n+1}) = false$, then:

    $\Gamma_{n+1}^\star(t) = [\infty, \infty]$

Example We shall consider the abstract version of the execution given in the previous section.

At time 0, 4 transitions are fireable:

Transition	Abstract date
`A.startTest1`	$[2190, 2190]$
`A.failure`	$[0+\epsilon, \infty]$
`B.startTest1`	$[4380, 4380]$
`B.failure`	$[0+\epsilon, \infty]$

`A.startTest1` is fired at the abstract date $[2190, 2190]$. After its firing, 3 transitions are fireable:

Transition	Abstract date
`A.completeTest`	$[2190, 2190] + [0,0] = [2190, 2190]$
`B.startTest1`	$[4380, 4380]$
`B.failure`	$[2190+\epsilon, \infty]$

`A.completeTest` is fired at the abstract date $[2190, 2190]$. After its firing, 4 transitions are fireable:

Transition	Abstract date
`A.startTest`	$[2190, 2190] + [4380, 4380] = [6570, 6570]$
`A.failure`	$[2190, 2190] + [0+\epsilon, \infty] = [2190+\epsilon, \infty]$
`B.startTest1`	$[4380, 4380]$
`B.failure`	$[2190+\epsilon, \infty]$

`B.failure` is fired at the abstract date $[2190 + \epsilon, 4380]$. After its firing, 3 transitions are fireable:

Transition	Abstract date
`A.startTest`	$[6570, 6570]$
`A.failure`	$[2190+\epsilon, \infty]$
`B.startTest1`	$[4380, 4380]$

`B.startTest1` is fired at the abstract date $[4380, 4380]$. After its firing, 3 transitions are fireable:

Transition	Abstract date
`A.startTest`	$[6570, 6570]$
`A.failure`	$[4380+\epsilon, \infty]$
`B.repair`	$[4380, 4380] + [12, 24] = [4392, 4404]$

`B.repair` is fired at the abstract date $[4392, 4404]$. After its firing, 4 transitions are fireable:

Transition	Abstract date
`A.startTest`	$[6570, 6570]$
`A.failure`	$[4392+\epsilon, \infty]$
`B.startTest`	$[4392, 4404] + [4380, 4380] = [8872, 8884]$
`B.failure`	$[4392, 4404] + [0+\epsilon, \infty] = [4392+\epsilon, \infty]$

And so on...

4.3 *Bisimulation*

The key mathematical property is that abstract and concrete executions are bisimilar: any concrete (timed, stochastic) execution can be simulated by an abstract execution and reciprocally any abstract execution corresponds to at least one concrete execution.

Theorem 1. *For any concrete execution* $\Lambda = \langle \sigma_0, d_0, \Gamma_0 \rangle \xrightarrow{t_1} \ldots \xrightarrow{t_n} \langle \sigma_n, d_n, \Gamma_n \rangle$*, of the Timed* $GTS = \langle V, E, T, \iota, A, delay \rangle$ *it exists an abstract execution* $\Lambda_a = \langle \sigma_0, d_0^\star, \Gamma_0^\star \rangle \xrightarrow{t_1} \langle \sigma_1, d_1^\star, \Gamma_1^\star \rangle \xrightarrow{t_2} \ldots \xrightarrow{t_n} \langle \sigma_n, d_n^\star, \Gamma_n^\star \rangle$*, such that the following properties hold:*

- $\forall n \geq 0\ d_n \in d_n^\star$;
- $\forall n \geq 0\ \forall t \in T\ \Gamma_n(t) \in \Gamma_n^\star(t)$.

Theorem 2. *Any abstract execution* $\Lambda_a = \langle \sigma_0, d_0^\star, \Gamma_0^\star \rangle \xrightarrow{t_1} \langle \sigma_1, d_1^\star, \Gamma_1^\star \rangle \xrightarrow{t_2} \ldots \xrightarrow{t_n} \langle \sigma_n, d_n^\star, \Gamma_n^\star \rangle$ *of the Timed* $GTS = \langle V, E, T, \iota, A, delay \rangle$ *corresponds to at least one concrete execution* $\Lambda = \langle \sigma_0, d_0, \Gamma_0 \rangle \xrightarrow{t_1} \ldots \xrightarrow{t_n} \langle \sigma_n, d_n, \Gamma_n \rangle$*, such that the following properties hold:*

- $\forall n \geq 0\ d_n \in d_n^\star$;
- $\forall n \geq 0\ \forall t \in T\ \Gamma_n(t) \in \Gamma_n^\star(t)$.

5 CONCLUSION AND PERSPECTIVES

In this article, we show how the AltaRica 3.0 stepwise simulator has been greatly improved by the introduction of an abstract notion of time. The abstract notion of time enables to reconcile both stochastic and stepwise simulations of AltaRica 3.0 models.

We show that abstract and concrete simulations are bisimilar: any concrete (timed, stochastic) execution

can be simulated by an abstract execution and reciprocally any abstract execution corresponds to at least one concrete execution.

We illustrate our purpose using a motivating example that mix both stochastic and deterministic transitions.

The introduction of the abstract notion of time to the stepwise simulator paves the way to the design of efficient model-checking algorithms, and in particular to the design of generators of sequences of events leading to a failure state.

The next step of our work is the application of the presented results for the development of an efficient sequence generator for AltaRica 3.0 models.

REFERENCES

Aupetit, B., M. Batteux, A. Rauzy, & J.-M. Roussel (2015, September). Improving performance of the AltaRica 3.0 stochastic simulator. In L. Podofillini, B. Sudret, B. Stojadinovic, E. Zio, and W. Kröger (Eds.), *Proceedings of Safety and Reliability of Complex Engineered Systems: ESREL 2015*, pp. 1815–1824. CRC Press.

Batteux, M., T. Prosvirnova, & A. Rauzy (2017, September). Altarica 3.0 assertions: the why and the wherefore. *Journal of Risk and Reliability*.

Brameret, P.-A., A. Rauzy, & J.-M. Roussel (2015, July). Automated generation of partial markov chain from high level descriptions. *Reliability Engineering and System Safety 139*, 179–187.

Cassandras, C. G. & S. Lafortune (2008). *Introduction to Discrete Event Systems*. New-York, NY, USA: Springer.

Clarke, E. M., O. Grumberg, & D. A. Peled (2000, February). *Model Checking*. Cambridge, MA, USA: MIT Press.

Cousot, P. & R. Cousot (1977). Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, NY, USA, pp. 238–252. ACM Press. Los Angeles, California.

Matloff, N. & P. J. Salzman (2008). *The Art of Debugging with GDB, DDD, and Eclipse*. San Fransisco, CA, USA: No Starch Press.

Milner, R. (1989). *Communication and Concurrency*. Prentice-Hall international series in computer science. Upper Saddle River, New Jersey, USA: Prentice Hall.

Prosvirnova, T., M. Batteux, P.-A. Brameret, A. Cherfi, T. Friedlhuber, J.-M. Roussel, & A. Rauzy (2013, September). The altarica 3.0 project for model-based safety assessment. In *Proceedings of 4th IFAC Workshop on Dependable Control of Discrete Systems, DCDS'2013*, York, Great Britain, pp. 127–132. International Federation of Automatic Control.

Prosvirnova, T. & A. Rauzy (2015). Automated generation of minimal cutsets from altarica 3.0 models. *International Journal of Critical Computer-Based Systems 6*(1), 50–79.

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

Zimmermann, A. (1976). *Stochastic Discrete Event Systems*. Berlin, Heidelberg, Germany: Springer.