Introduction d'une notion de temps abstrait pour la simulation des modèles AltaRica 3.0

Michel Batteux, Antoine Rauzy, Tatiana Prosvirnova

INTRODUCTION D'UNE NOTION DE TEMPS ABSTRAIT POUR LA SIMULATION DES MODELES ALTARICA 3.0

INTRODUCING AN ABSTRACT NOTION OF TIME FOR THE STEPWISE SIMULATION OF ALTARICA 3.0 MODELS

Michel Batteux
IRT SystemX
Saclay, France

TatianaProsvirnova
CentraleSupélec
Gif-sur-Yvette, France

Antoine Rauzy
NTNU
Trondheim, Norway

Résumé

AltaRica 3.0 est un langage de modélisation événementiel et orienté objet, dédié aux analyses probabilistes du risque de systèmes complexes. Plusieurs outils d'évaluation ont été développés pour AltaRica 3.0, y compris un simulateur pas-à-pas. Cet outil est d'une grande aide pour la conception et la validation des modèles AltaRica 3.0. Dans cet article, nous montrons comment le simulateur pas-à-pas d'AltaRica 3.0 a été grandement renforcée par l'introduction d'une notion d'abstraction du temps. Ce résultat important ouvre la voie vers la conception d'algorithmes efficaces de vérification de modèles, par exemple, pour la conception de générateurs de séquences d'événements menant à un état donné de défaillance.

Summary

AltaRica 3.0 is an event-based, object-oriented modeling language dedicated to (probabilistic) safety analyses of complex systems. 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. 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.
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.

Introduction

AltaRica 3.0 est un langage de modélisation événementiel et orienté objet, dédié aux analyses probabilistes du risque de systèmes complexes (Prosvirnova *et al.*, 2013). Il permet de modéliser les systèmes à plus haut niveau que les formalismes classiques d'analyse du risque (arbres de défaillance, chaînes de Markov, réseaux de Petri stochastiques, etc.), sans augmenter la complexité des calculs d'indicateurs du risque.
La sémantique d'AltaRica 3.0 est définie au moyen des systèmes de transitions gardées (GTS) (Rauzy, 2008). L'exécution des modèles AltaRica 3.0 est similaire aux autres formalismes à événements discrets dans le sens où chaque fois qu'une transition est tirable, elle est alors planifiée et sera alors potentiellement tirée après un certain délai (Cassandras *et al.*, 2008), (Zimmerman, 1976). Ces délais peuvent être déterministes ou stochastiques ; et dans ce dernier cas, AltaRica 3.0 fournit les délais usuels (du type exponentiel, Weibull, etc.) et des délais empiriques.

Plusieurs outils d'évaluation sont disponibles pour AltaRica 3.0 (Prosvirnova *et al.*, 2015), (Brameret *et al.*, 2015), (Aupetit *et al.*, 2015), notamment un simulateur pas-à-pas. Cet outil permet de réaliser des simulations interactives, c'est-à-dire d'aller et venir dans les séquences d'événements, permettant de ce fait de trouver des erreurs de modélisation, des comportements anormaux, etc.

Nous avons grandement amélioré le pouvoir de traitement du simulateur pas-à-pas en introduisant la notion du temps abstrait. Jusqu'à présent, cet outil ne considérait pas le temps car il était trop compliqué pour l'analyste de manipuler à la main les délais associés aux transitions stochastiques : une infinité de valeurs pouvant être choisies, laissant l'analyste perplexe quant aux choix des valeurs les plus appropriées pour son besoin. Toutefois, ignorer les délais avait un inconvénient important car le simulateur pas-à-pas permettait de réaliser des séquences d'événements sans équivalent en simulation stochastique et, plus généralement, qui n'obéissaient pas à la sémantique temporisée de AltaRica 3.0.

L'idée introduite est d'abstraire le temps dans les simulations pas-à-pas. Nous avons associé à chaque transition un intervalle temporel durant lequel la transition est tirable. Cet intervalle pouvant être réduit à un singleton ou être une union d'intervalles contenant le délai infini. De ce fait, le tir d'une transition peut modifier les intervalles de temps associés aux autres transitions déjà planifiées.

Cette idée d'abstraction temporelle a été largement étudiée dans le domaine de l'intelligence artificielle pour raisonner à propos du temps (Cousot *et al.*, 1977). Le problème à investiguer fut donc de rendre cela fonctionnel dans le cadre particulier de la simulation stochastique à événements discrets. La propriété mathématique clé est que les simulations abstraites et concrètes sont bisimilaires (voir (Milner, 1989) pour cette notion importante de bisimilarité). Cela signifie que toute exécution concrète (temporisée ou stochastique) peut être simulée par une exécution abstraite ; et réciproquement toute exécution abstraite correspond à au moins une exécution concrète. Finalement les exécutions abstraites et concrètes sont conformes avec la sémantique du langage AltaRica 3.0.

Les premières expérimentations sur le simulateur pas-à-pas permettent d'explorer, au travers de simulations interactives, plus finement le comportement des modèles AltaRica 3.0. Le simulateur pas-à-pas permet de jouer le même rôle pour la modélisation événementielle que les outils de debug comme GDB ou DDD pour les langages de programmation (Matloff *et al.*, 2008).

Enfin ce résultat important ouvre la voie vers la conception d'algorithmes efficaces de model-checking (voir (Clarke *et al.*, 2000) pour une introduction), avec en particulier la conception de générateurs de séquences d'événements menant à un état donné de défaillance.

Des travaux sur une extension temporisée d'AltaRica à l'aide d'horloges ont été réalisés pour vérifier les propriétés temps réelles (Cassez *et al.*, 2004). La notion de temps abstrait est différente car n'implique pas d'horloges et reste complètement compatible avec la sémantique temporisée/stochastique du langage AltaRica.

La suite de cet article est organisé comme suit. La section 2 présente un exemple illustratif. La section 3 rappelle les notions fondamentales sur le temps et les systèmes de transitions gardées stochastiques. La section 4 introduit la notion de temps abstrait. La section 5 conclut cet article et donne quelques perspectives.

Exemple

A titre d'exemple illustratif, nous considérerons un système composé de deux composants testés périodiquement qui évoluent indépendamment. Le comportement d'un tel composant est représenté par le diagramme d'état illustré Figure 1.

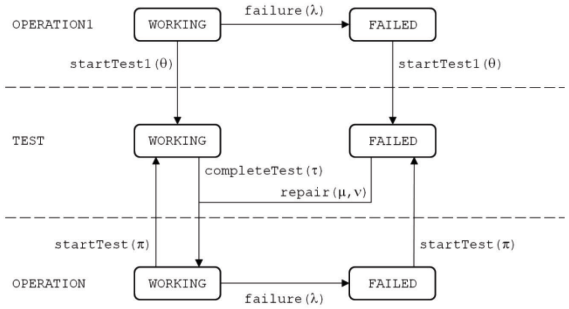

Figure 1: Diagramme d'état pour un composant testé périodiquement.

Le composant alterne les phases de fonctionnement et de test. Il est en marche initialement et commence avec une première phase d'opération qui dure un temps constant θ. Toutes les phases d'opération suivantes durent un temps constant π. Le composant peut échouer en fonctionnement, avec un taux de défaillance λ. Si le composant fonctionne lorsqu'il entre dans une phase de maintenance, cette phase de maintenance dure un temps constant τ. Si, au contraire, il tombe en panne lorsqu'il entre dans une phase de maintenance, la durée de sa réparation est uniformément répartie entre deux valeurs μ et ν. Enfin, on suppose que le composant est aussi bon que neuf après une réparation. Dans le diagramme d'état de la Figure 1, les transitions **failure** et **repair** sont donc stochastiques, tandis que les transitions **startTest1**, **startTest** et **completeTest** sont déterministes.

Le Tableau 1 montre une évolution possible d'un tel composant.

Transition	Date à laquelle la transition a été tiré
startTest1	d1 = θ
completeTest	d2 = d1 + τ
failure	d3 = d2 + δ1, 0<δ1≤ π
startTest	d4 = d2 + π
repair	d5 = d4 + δ2, μ≤δ2≤ν
startTest	d6 = d5 + π
completeTest	d7 = d6 + τ
...	...

Tableau 1: Une évolution possible d'un composant périodiquement testé.

Nous pouvons supposer que θ et π sont relativement gros par rapport à τ, μ et ν, et que τ est plus petit que μ (lui-même plus petit que ν).

Afin de ne pas avoir les deux composants A et B hors service en raison de test ou de maintenance en même temps, il est raisonnable de prendre différentes les valeurs de θ pour A et pour B (les valeurs des autres paramètres étant identiques pour les deux composants). Par exemple, si une phase d'opération dure normalement six mois (π = 4380h), le composant A peut être testé après trois mois (A.θ = 2190h) tandis que le composant B est testé après six mois (B.θ = 4380h). De cette façon, les tests / maintenances de A et de B sont décalés de trois mois, ce qui améliore la disponibilité globale du système. Le Tableau 2 donne quelques valeurs typiques de paramètres pour les composants A et B que nous utiliserons tout au long de l'article.

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

Tableau 2: Valeurs typiques des paramètres.

Avec ces valeurs de paramètres à l'esprit, le lecteur voit immédiatement le problème d'utiliser un simulateur pas à pas qui ne tient pas compte des délais associés aux transitions. De nombreuses exécutions qui seraient impossibles avec une sémantique temporisée deviennent possibles avec une sémantique non temporisée, par exemple:

```
B.startTest1->...->A.startTest1->
A.completeTest->A.startTest->
A.completeTest->...
```

D'un autre côté, demander à l'analyste d'introduire des délais des transitions stochastiques de manière interactive n'est pas une solution pratique. Non seulement cela serait fastidieux, mais cela laisserait l'analyste faire face au choix de délais appropriés, ce qui devient rapidement déroutant. D'où la nécessité d'une notion du temps abstrait qui permet de prendre en compte les délais de transitions sans demander à l'analyste de les entrer interactivement.

Pour répondre à ce besoin, l'idée est de raisonner en termes d'intervalles de temps plutôt qu'en termes de dates. Pour expliquer comment cette idée fonctionne, nous rappellerons d'abord la sémantique régulière de AltaRica dans la section suivante. Ensuite, nous introduirons sa sémantique abstraite (en termes d'intervalles de temps) dans la Section 4.

Systèmes de Transitions Gardées Stochastiques

La sémantique du langage AltaRica 3.0 est définie en termes de systèmes de transitions gardées stochastiques (Rauzy, 2008), (Batteux, 2017). Nous rappellerons ici seulement les notions qui sont importantes aux fins de cet article. Le lecteur devrait se reporter aux articles cités pour des présentations plus approfondies.

1. Définition

Un systèmes de transitions gardées est un quintuplet $\langle V, E, T, A, \imath \rangle$, où :

1. V est un ensemble de variables. Chaque variable v de V possède un type : peut prendre ses valeur dans un ensemble de constantes (Booléennes, entières, réelles ou constantes symboliques. V est un ensemble disjoint d'un sous-ensemble de variables d'état S et d'un sous-ensemble de variables de flux F.
2. E est un ensemble d'événements. On associe à chaque événement e de E une fonction non-décroissante et inversible delay(e) de [0,1] à R+ ∪ {+∞}. L'inverse de cette fonction delay^{-1} est une distribution de probabilité cumulative.
3. T est un ensemble de transitions. Une transition est un triplet $\langle e, G, P \rangle$, noté aussi $G \xrightarrow{e} P$, où e

est un événement de E, G est une expression Booléenne sur les variables de V appelée la garde, et *P* est un instruction qui change les valeurs de certaines variables d'état.

4. A est une instruction, appelée une assertion, qui calcule les valeurs des variables de flux en fonction des valeurs des variables d'état.
5. Finalement, ι est une fonction qui donne des valeurs initiales aux variables d'état et des valeurs par défaut aux variables de flux.

Exemple :

Voici un système de transitions gardées qui modélise le composant testé périodiquement présenté dans la section précédente.
L'état du système est représenté par deux variables d'état : `state` qui prend ses valeurs dans l'ensemble `{WORKING, FAILED}` et `phase` qui prend ses valeurs dans l'ensemble `{OPERATION1, TEST, OPERATION}`.
Les événements sont `startTest1`, `startTest`, `completeTest`, `failure` et `repair`. Leurs fonctions delay(e) ont été décrites dans la section 2.
Les transitions sont comme suit.

```
state==WORKING and phase != TEST ->
   failure-> state := FAILED;
phase==OPERATION1 -> startTest1 ->
   phase := TEST;
phase==OPERATION -> startTest ->
   phase := TEST;
state == WORKING and phase==TEST ->
   completeTest -> phase := OPERATION;
state == FAILED and phase==TEST -> repair
   -> state := WORKING; phase := OPERATION;
```

L'état initial pour les variables est
`state=WORKING, phase=OPERATION1.`

2. Composition

L'un des avantages des systèmes de transitions gardées par rapport à d'autres formalismes similaires c'est qu'ils sont compositionnels.
Soit M_1: $\langle V_1, E_1, T_1, A_1, \iota_1\rangle$ et M_2: $\langle V_2, E_2, T_2, A_2, \iota_2\rangle$ deux systèmes de transitions gardées. La composition $M_1 \otimes M_2$ est un système de transitions gardées $\langle V, E, T, A, \iota\rangle$ tel que $V = V_1 \cup V_2$, $E = E_1 \cup E_2$, $T = T_1 \cup T_2$, $A = A_2 \circ A_1$ et $\iota = \iota_2 \circ \iota_1$.
Le principe ci-dessus s'étend à n'importe quel nombre de systèmes de transitions gardées.

Exemple :

Pour représenter le système d'exemple discuté dans la section précédente, il suffit de créer deux copies du système de transitions gardées ci-dessus et de les composer (ce qui est fait automatiquement par le compilateur AltaRica).
Dans notre exemple, il peut être utile d'introduire une variable booléenne de flux `failed` pour dire quand le système est en panne. L'assertion définissant cette variable pourrait être la suivante :

```
failed := A.state==FAILED and B.state ==
FAILED;
```

3. Sémantique

La sémantique des systèmes de transitions gardées *M*: $\langle V = S \uplus F, E, T, A, \iota\rangle$ est un ensemble d'exécutions possibles. Pour définir formellement la notion d'exécution, nous allons définir d'abord la notion d'état et d'ordonnanceur.
Un état σ de *M* est une affectation de variables de V qui vérifie $\sigma(F) = A(\sigma(S))$.
Un ordonnanceur de M est une fonction de T à $\mathrm{R}^+ \cup \{+\infty\}$. Un ordonnanceur Γ est compatible avec un état σ et la date $d \in \mathrm{R}^+$ si pour toutes les transitions *t*: $\langle e, G, P\rangle$ de T, $d \le \Gamma(t)$ si $G(\sigma) = \text{true}$ et $\Gamma(t) = +\infty$ sinon.
Une exécution de *M* est une séquence :

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

où $n \ge 0$, les σ_i's sont les états de *M*, les d_i's sont les dates, des nombres qui vérifient $0 = d_0 \le d_1 \le \ldots \le d_n$, chaque Γ_i est un ordonnanceur compatible avec l'état σ_i et la date d_i et les t_i's sont les transitions de *M*.
L'ensemble d'exécutions valides est défini récursivement. Une exécution vide $\langle \sigma_0, 0, \Gamma_0\rangle$ est une exécution valide si $\sigma_0(S) = \iota$, $\sigma_0(F) = A(\iota)$ et l'ordonnanceur Γ_0 est tel que pour toutes les transitions t: $\langle e, G, P\rangle$ de T :

- $\Gamma_0(t) = delay_e(z)$ pour un certain nombre $z \in [0,1]$ si $G(\sigma_0) = true$.
- $\Gamma_0(t) = +\infty$ si $G(\sigma_0) = false$.

Si la séquence suivante est valide

$$\Lambda = \langle \sigma_0, d_0, \Gamma_0\rangle \xrightarrow{t_1} \langle \sigma_1, d_1, \Gamma_1\rangle \xrightarrow{t_2} \ldots \xrightarrow{t_n} \langle \sigma_n, d_n, \Gamma_n\rangle, \quad n \ge 0$$

alors la séquence suivante l'est aussi

$$\Lambda \xrightarrow{t_{n+1}} \langle \sigma_{n+1}, d_{n+1}, \Gamma_{n+1}\rangle$$

si les conditions suivantes sont satisfaites :

- $t_{n+1} = \langle e_{n+1}, G_{n+1}, P_{n+1}\rangle$
- $G_{n+1}(\sigma_n) = true$.
- $\sigma_{n+1} = A(P_{n+1}(\sigma_n))$, le tir de la transition t_{n+1} se déroule en deux étapes: d'abord, les variables d'état sont calculées grâce à l'action P_{n+1} de la transition, ensuite les variables de flux sont mises à jour grâce à l'assertion A.
- $d_{n+1} = \Gamma_n(t_{n+1})$ et il n'y a pas de transition *t* dans *T* tel que $\Gamma_n(t) < \Gamma_n(t_{n+1})$.
- Γ_{n+1} est calculé à partir de Γ_n de la manière suivante: pour toute transition *t*: $\langle e, G, P\rangle$ dans T
  - Si $G(\sigma_{n+1}) = true$, alors:
    - Si $G(\sigma_n) = true$ et $t \ne t_{n+1}$ (si la transition a été déjà tirable avant), alors $\Gamma_{n+1}(t) = \Gamma_n(t)$, i.e. la valeur précédente est gardée.
    - Sinon, $\Gamma_{n+1}(t) = d_{n+1} + delay_e(z)$ pour un certain nombre $z \in [0,1]$, i.e. une nouvelle date de tir est choisie.
  - Si $G(\sigma_{n+1}) = false$, alors $\Gamma_{n+1}(t) = +\infty$.

Notez que les exécutions sont déterminées par les choix des *z*'s.

Exemple :

Reprenons l'exemple présenté dans le section précédente.
A l'instant 0, 4 transitions sont tirables :

Transition	Date
A.startTest1	2190
A.failure	5617
B.startTest1	4380
B.failure	4111

La transition **A.startTest1** a la date de tir la plus proche, elle est tirée à la date 2190. Après le tir de cette transition, 3 transitions sont tirables :

Transition	Date
A.completeTest	2190 + 0 = 2190
B.startTest1	4380
B.failure	4111

La transition **A.completeTest** a la date de tir la plus proche, elle doit être tirée en premier à la date 2190. Après le tir de cette transition 4 transitions sont tirables :

Transition	Date
A.startTest	2190 + 4380 = 6570
A.failure	2190 + 6020 = 8210
B.startTest1	4380
B.failure	4111

La transition **B.failure** a la date de tir la plus proche, elle est tirée à la date 4111. Après son tir 3 transitions sont tirables :

Transition	Date
A.startTest	6570
A.failure	8210
B.startTest1	4380

La transition **B.startTest1** a la date de tir la plus proche, elle est tirée à la date 4380. Après le tir de cette transition 3 transitions sont tirables :

Transition	Date
A.startTest	6570
A.failure	8210
B.repair	4400

La transition **B.repair** a la date de tir la plus proche, elle est tirée à la date 4400. Après le tir de cette transition, 4 transitions sont tirables :

Transition	Intervalle
A.startTest	6570
A.failure	8210
B.startTest	4400+4380=8780
B.failure	4400+5201=9601

Cette séquence montre comment les transitions déterministes et stochastiques peuvent être mélangées. En particulier, les dates des tests ne sont pas décidées une fois pour toutes. Elles dépendent des temps de défaillance et de réparation du composant.

Sémantique abstraite

4. Principe

La première idée pour abstraire les exécutions consiste à associer un délai abstrait $delay^*$ à chaque événement du modèle. $delay^*(e)$ est simplement l'image de la fonction *delay*, c'est-à-dire un intervalle de nombres réels non négatifs.

Nous devons être prudent, car certains intervalles qui sont les images des distributions, sont fermés alors que d'autres sont ouverts (à gauche et / ou à droite) et que nous devons considérer des limites infinies. Une solution consiste à travailler uniquement avec des intervalles fermés, mais dans une arithmétique non-standard construite sur un ensemble

$R' = R \cup \{\varepsilon\} \cup \{\infty\}$, où

ε et ∞ sont respectivement infiniment petit et infiniment grand nombres vérifiant :

- $\varepsilon + \varepsilon = \varepsilon$
- $x + \infty = \infty$, pour tout x de R'.

De cette façon, l'intervalle (a, b) peut être codé comme [a + ε, b - ε].

Le Tableau 3 donne les délais abstraits associés aux distributions les plus largement utilisées en AltaRica 3.0. De plus, notez qu'une transition dont la garde n'est pas satisfaite dans l'état courant est programmée dans l'intervalle [∞, ∞].

Délai concret	Délai abstrait
Diract(T)	[T,T]
UnifromDeviate(l, h)	[l+ε, h-ε]
Exponential(λ)	[0+ε, ∞]
Weibull(α,β)	[0+ε, ∞]
Distribution empirique	[0+ε, ∞]

Tableau 3: Intervalles associés aux fonctions Delay.

La deuxième idée est de ne pas considérer la date à laquelle les transitions sont déclenchées, mais un intervalle de temps dans lequel elles sont tirées.

Supposons que nous construisons une séquence pas-à-pas et que la dernière transition qu'il faut tirer est tirable dans l'intervalle de temps [l, h]. Supposons en outre que les transitions $t_1, t_2, \ldots, t_n$ sont planifiées dans des intervalles de temps $[l_1, h_1], [l_2, h_2], \ldots, [l_n, h_n]$.

Ensuite, nous pouvons faire les remarques suivantes :

1. Nous devons avoir $\mathbf{l \le l_i}$ pour tout $\mathbf{i = 1, \ldots, n}$, car la transition suivante ne peut pas être planifiée dans le passé.
2. Nous devons aussi avoir $\mathbf{h \le h_i}$ pour tout i = 1,..., n, car si $h_i < h$ pour un certain i, cela signifie que la transition t_i doit être tirée avant h_i, par conséquent, la dernière transition doit également être tirée avant h_i.
3. Pour la même raison, nous pouvons choisir t_i comme la prochaine transition à tirer seulement s'il n'y a pas d'autre transition t_j telle que $h_j < l_i$.
4. Toujours pour la même raison, si la transition t_i est tirée, elle est nécessairement tirée dans l'intervalle $[l_i, h_{min}]$, où h_{min} est le plus petit des h_j.
5. Si la transition t_i est tirée et que la transition t_j est telle que $l_j < l_i$, alors l_j doit être changé en l_i pour obéir à notre première remarque.
6. Enfin, si la transition t_i a été tirée et une transition t associée à l'intervalle $[l_t, h_t]$ devient tirable (t peut être la transition t_i elle-même), alors t doit être planifiée dans l'intervalle $[l_i, h_{min}] + [l_t, h_t] = [l_i + l_t, h_{min} + h_t]$.

Nous pouvons maintenant définir formellement la sémantique abstraite des systèmes de transitions gardées (et donc de AltaRica 3.0).

5. Définition formelle

La sémantique abstraite d'un système de transitions gardées S: $\langle V, E, T, A, \imath \rangle$ est définie comme l'ensemble de ses exécutions abstraites possibles.

Pour définir formellement une exécution abstraite, nous devons introduire la notion d'ordonnanceur abstrait. Un ordonnanceur abstrait de S = $\langle V, E, T, A, \imath \rangle$ est une

fonction de l'ensemble *T* vers des intervalles fermés sur *R'*.

Un ordonnanceur Γ^* est compatible avec un état σ du système de transitions gardées S et avec la date abstraite d*=[l, h] si les conditions suivantes sont valables pour toutes les transitions t dans T, avec Γ^* (t) = [l_t, h_t] :

- $l \le l_t < \infty$ et $h \le h_t$ si G (σ) = true .
- $l_t = h_t = \infty$ si G (σ) = false.

Une exécution abstraite de S est une séquence:

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

où $n \geq 0$, les σ_i sont les états de S, les d_i^*'s sont les dates abstraites, c'est-à-dire des intervalles de temps, chaque Γ_i^* est un ordonnanceur abstrait compatible avec σ_i et avec d_i^* et enfin les t_i sont les transitions de S.

L'ensemble des exécutions valides est défini de manière récursive comme suit.

L'exécution abstraite vide $\langle \sigma_0, [0, 0], \Gamma_0^* \rangle$ est une exécution abstraite valide si l'ordonnanceur abstrait Γ_0^* est tel que pour toutes les transitions t: $\langle e, G, P \rangle$ de T :

- $\Gamma_0^*(t) = delay^*(e)$ si $G(\sigma_0) = true$.
- $\Gamma_0^*(t) = [\infty, \infty]$ si $G(\sigma_0) = false$.

Si la séquence Λ est une exécution abstraite valide

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

$n \geq 0$

Alors l'exécution

$$\Lambda \xrightarrow{t_{n+1}} \langle \sigma_{n+1}, d_{n+1}^*, \Gamma_{n+1}^* \rangle$$

l'est aussi si les conditions suivantes sont satisfaites :

- $t_{n+1} = \langle e_{n+1}, G_{n+1}, P_{n+1} \rangle$, $\Gamma_n^*(t_{n+1}) = [\, l^*, h^* \,]$, $h_{min} = \min_{[l,h]=\Gamma_n^*(t), t \in T} h$
- $G_{n+1}(\sigma_n) = true$.
- $\sigma_{n+1} = A(P_{n+1}(\sigma_n))$.
- Il n'y a pas de transition t dans T tel que $\Gamma_n^*(t) = [\, l, h \,]$ et $h < l^*$.
- $[\, l_{n+1}, h_{n+1} \,] = [\, l^*, h_{min} \,]$
- Γ_{n+1}^* est calculé à partir de Γ_n de la manière suivante: pour toute transition t: $\langle e, G, P \rangle$ dans T
  - Si $G(\sigma_{n+1}) = true$, alors :
    - Si $G(\sigma_n) = true$ et $t \neq t_{n+1}$ (si la transition a été déjà tirable avant), alors $\Gamma_{n+1}^*(t) = [\, \max(l_{n+1}, l), h \,]$.
    - Sinon, $\Gamma_{n+1}^*(t) = [\, l_{n+1}, h_{n+1} \,] + delay^*(e)$
  - Si $G(\sigma_{n+1}) = false$, alors $\Gamma_{n+1}^*(t) = [\infty, \infty]$.

Exemple :

Considérons la version abstraite de l'exécution donnée dans la section précédente.

A l'instant 0, 4 transitions sont tirables :

Transition	Intervalle
A.startTest1	[2190, 2190]
A.failure	[0 + ε, ∞]
B.startTest1	[4380, 4380]
B.failure	[0 + ε, ∞]

La transition **A.startTest1** est tirée dans l'intervalle [2190, 2190]. Après le tir de cette transition, 3 transitions sont tirables :

Transition	Intervalle
A.completeTest	[2190, 2190] + [0,0] = [2190, 2190]
B.startTest1	[4380, 4380]
B.failure	[2190 + ε, ∞]

La transition **A.completeTest** est tirée dans l'intervalle [2190, 2190]. Après le tir de cette transition, 4 transitions sont tirables :

Transition	Intervalle
A.startTest	[2190, 2190] + [4380, 4380] = [6570, 6570]
A.failure	[2190, 2190] + [0 + ε, ∞] = [2190 + ε, ∞]
B.startTest1	[4380, 4380]
B.failure	[2190 + ε, ∞]

La transition **B.failure** est tirée dans l'intervalle [2190 + ε, ∞]. Après le tir de cette transition, 3 transitions sont tirables :

Transition	Intervalle
A.startTest	[6570, 6570]
A.failure	[2190 + ε, ∞]
B.startTest1	[4380, 4380]

La transition **B.startTest1** est tirée dans l'intervalle [4380, 4380]. Après le tir de cette transition, 3 transitions sont tirables :

Transition	Intervalle
A.startTest	[6570, 6570]
A.failure	[4380 + ε, ∞]
B.repair	[4380, 4380] + [12+ε, 24-ε] = [4392+ε, 4404-ε]

La transition **B.repair** est tirée dans l'intervalle [4392+ε, 4404-ε]. Après le tir de cette transition, 4 transitions sont tirables :

Transition	Intervalle
A.startTest	[6570, 6570]
A.failure	[4392+ε, ∞]
B.startTest	[4380, 4380] + [4392+ε, 4404-ε] = [8872 + ε, 8884 - ε]
B.failure	[4392+ε, 4404-ε] + [0 + ε, ∞] = [4392+ε, ∞]

6. Bisimulation

La propriété mathématique clé est la suivante. Les exécutions concrètes sont bisimilaires : toute l'exécution (temporisée, stochastique) peut être simulée par une exécution abstraite et réciproquement toute exécution abstraite correspond à au moins une exécution concrète.

Théorème 1 : Pour toute exécution concrète

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

d'un système de transitions gardées
M = $\langle V = S \uplus F, E, T, A, \iota \rangle$
il existe une exécution abstraite

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

telle que les propriétés suivantes sont vérifiées :
- $\forall i \geq 0 \quad d_i \in d_i^*$
- $\forall i \geq 0 \; \forall t \in T \quad \Gamma_i(t) \in \Gamma_i^*(t)$.

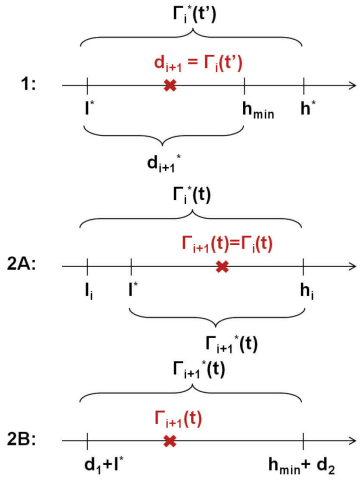

Figure 2 : Différents cas à prouver.

Preuve : A l'état initial :

- $d_0 = 0, d_0^* = [\,0, 0\,]$ donc $d_0 \in d_0^*$,
- $\Gamma_0(t) = delay_e(z)$ pour un certain nombre $z \in [0,1]$ si $G(\sigma_0) = true$, $\Gamma_0(t) = +\infty$ si $G(\sigma_0) = false$.
- $\Gamma_0^*(t) = delay^*(e)$ si $G(\sigma_0) = true$, $\Gamma_0^*(t) = [\,\infty, \infty\,]$ si $G(\sigma_0) = false$.

Par construction $delay_e(z) \in delay^*(e)\ \forall\ e \in E$ et par conséquent $\Gamma_0(t) \in \Gamma_0^*(t)\ \forall\ t \in T$.

Maintenant supposons que pour un certain $i > 0$

- $d_i \in d_i^*$,
- $\Gamma_i(t) \in \Gamma_i^*(t)\ \forall\ t \in T$, et
- la transition ***t'=<e', G', P'>*** est la prochaine à être tirée, donc $d_{i+1} = \Gamma_i(t') = \min_{t \in T} \Gamma_i(t) \leq \Gamma_i(t) \forall\ t \in T$

Il faut montrer que :

1. $d_{i+1} \in d_{i+1}^*$,
2. $\Gamma_{i+1}(t) \in \Gamma_{i+1}^*(t)\ \forall\ t \in T$.

Différents cas à démontrer sont illustrés Figure 2.

1. $d_{i+1} = \Gamma_i(t') = \min_{t \in T} \Gamma_i(t) \leq \Gamma_i(t) \forall\ t \in T$

$d_{i+1}^* = [l^*, h_{min}]$

$\Gamma_i^*(t') = [\,l^*, h^*\,]$

On a $d_{i+1} = \Gamma_i(t') \in \Gamma_i^*(t') = [\,l^*, h^*]$

Il reste à montrer que $d_{i+1} \leq h_{min}$.

Supposons que $d_{i+1} > h_{min}$. Mais d'après les remarques 2-4 plus haut ce n'est pas possible car dans ce cas il existe une transition qui aurait dû être tirée avant t'. Donc $d_{i+1} \leq h_{min}$ et $d_{i+1} \in d_{i+1}^*$.

2. Montrons ensuite que $\Gamma_{i+1}(t) \in \Gamma_{i+1}^*(t)\ \forall\ t \in T$.

Deux cas sont à démontrer :

- 2A: la transition t différente de t' était tirable avant le tir de t' et reste tirable après;
- 2B: la transition t n'était pas tirable et devient tirable après le tir de t' et la transition t' elle-même si elle est à nouveau tirable.

2A. Dans ce cas nous avons

$\Gamma_{i+1}(t) = \Gamma_i(t) \in \Gamma_i^*(t) = [l_i, h_i]$.

Il faut montrer que

$\Gamma_{i+1}(t) \in \Gamma_{i+1}^*(t) = [\,\max(l_i, l^*), h_i]$.

Si $l^* \geq l_i$, nous avons

$l^* \leq d_{i+1} \leq \Gamma_i(t)\ \forall\ t \in T$ donc $\Gamma_{i+1}(t) \in \Gamma_{i+1}^*(t)$.

2B. Dans ce cas nous avons

- $\Gamma_{i+1}(t) = d_{i+1} + delay_e(z),\ z \in [0,1]$
- $\Gamma_{i+1}^*(t) = d_{i+1}^* + delay^*(e) = [\,l_{i+1}, h_{i+1}\,]$

Nous avons montré que

$l^* \leq d_{i+1} \leq h_{min}$

Soit $delay^*(e) = [d_1, d_2\,]$.

Par construction

$d_1 \leq delay_e(z) \leq d_2$

Par conséquent

$l^* + d_1 \leq d_{i+1} + delay_e(z) \leq h_{min} + d_2$.

$l_{i+1} = l^* + d_1$,

$h_{i+1} = h_{min} + d_1$ et donc

$\Gamma_{i+1}(t) \in \Gamma_{i+1}^*(t)$.

Théorème 2 : Pour toute exécution abstraite

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

d'un système de transitions gardées

$M = \langle V = S \uplus F, E, T, A, \iota \rangle$

il existe au moins une exécution concrète

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

telle que les propriétés suivantes sont vérifiées :

- $\forall\ i \geq 0\ \ d_i \in d_i^*$
- $\forall\ i \geq 0\ \forall\ t \in T\ \ \Gamma_i(t) \in \Gamma_i^*(t)$.

Preuve : Soit une exécution abstraite

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

Construisons pas à pas une exécution concrète possible qui lui correspond.

A l'état initial

- $d_0 = 0 \in d_0^* = [0,0]$
- $\Gamma_0(t) = delay_e(z) \in \Gamma_o^*(t) = delay^*(e)$, si la transition t est tirable,
- $\Gamma_0(t) = \infty \in \Gamma_0^*(t) = [\infty, \infty\,]$, sinon.

De plus, pour toutes les transitions stochastiques tirables choisissons les dates de tir pour que la propriété suivante soit vérifiée :

$\Gamma_0(t_1) \leq \Gamma_0(t_2) \leq \dots \leq \Gamma_0(t_n)$.

Dans ce cas là t_1 est bien la prochaine transition tirable.

Soit pour un certain $i > 0$:

- $d_i \in d_i^*$,
- $\Gamma_i(t) \in \Gamma_i^*(t)\ \forall\ t \in T$ et
- $\Gamma_i(t_{i+1}) \leq \Gamma_i(t)\ \forall\ t \in T$
- $\Gamma_i(t_{i+1}) \leq \Gamma_i(t_{i+2}) \leq \dots \leq \Gamma_i(t_n)$

Dans ce cas t_{i+1} est la prochaine transition tirable.

Il faut montrer que

- $d_{i+1} \in d_{i+1}^*$
- $\forall\ t \in T\ \ \Gamma_{i+1}(t) \in \Gamma_{i+1}^*(t)$.

1. $d_{i+1} = \Gamma_i(t_{i+1}) \in d_{i+1}^*$ (voir le cas 1 de la preuve précédente).

2. Dans le cas où la transition t différente de t_{i+1} était tirable et reste tirable après le tir de t_{i+1}:

$\Gamma_{i+1}(t) = \Gamma_i(t) \in \Gamma_{i+1}^*(t)$ (voir le cas 2A de la preuve précédente).

Dans le cas où la transition t n'était pas tirable et devient tirable après le tir de t_{i+1}ou la transition t_{i+1} est à nouveau tirable :

$\Gamma_{i+1}(t) = d_{i+1} + delay_e(z) \in \Gamma^*_{i+1}(t), z \in [0,1]$ (voir le cas 2B de la preuve précédente).
De plus, pour toutes les transitions stochastiques tirables nous choisissons $delay_e(z)$ pour que les propriétés suivantes soit vérifiées :

- $\Gamma_{i+1}(t_{i+2}) \leq \Gamma_{i+1}(t)\ \forall\ t\ \in T$
- $\Gamma_{i+1}(t_{i+2}) \leq \Gamma_{i+1}(t_{i+3}) \leq \cdots \leq \Gamma_{i+1}(t_n)$.

Dans ce cas t_{i+2} est la prochaine transition tirable.

Conclusion

Dans cet article, nous montrons comment le simulateur pas-à-pas de AltaRica 3.0 a été amélioré par l'introduction d'une notion de temps abstrait. La notion de temps abstrait permet de réconcilier à la fois la simulation stochastique et la simulation pas-à-pas des modèles AltaRica 3.0.

Les simulations abstraites et concrètes sont bisimilaires : toute exécution concrète (temporisée, stochastique) peut être simulée par une exécution abstraite et réciproquement toute exécution abstraite correspond à au moins une exécution concrète.

Comme illustration nous utilisons un exemple qui mélange à la fois les transitions stochastiques et les transitions déterministes.

L'introduction de la notion abstraite de temps au simulateur pas-à-pas ouvre la voie à la conception des algorithmes efficaces pour la vérification des modèles, et en particulier à la conception de générateurs de séquences d'événements conduisant à un état de panne.

La prochaine étape de notre travail est l'application des résultats présentés pour le développement d'un générateur de séquences efficace pour les modèles AltaRica 3.0.

7. Références

Aupetit, B., M. Batteux, A. Rauzy, and J.-M. Roussel, 2015, Improving performance of the AltaRica 3.0 stochastic simulator, Proceedings of Safety and Reliability of Complex Engineered Systems: ESREL 2015, pp. 1815–1824. CRC Press.
Batteux, M., T. Prosvirnova, and A. Rauzy, 2017, Altarica 3.0 assertions: the why and the wherefore. Journal of Risk and Reliability.
Brameret, P.-A., A. Rauzy, J.-M. Roussel, 2015, Automated generation of partial markov chain from high level descriptions. Reliability Engineering and System Safety, vol. 139, pp. 179–187.
Cassandras, C. G, S. Lafortune, 2008, Introduction to Discrete Event Systems. New-York, NY, USA: Springer.
Cassez F., Pagetti C., Roux O., 2014, A timed extension for AltaRica, vol. 62, Fundamenta Informaticae, pp. 191-332.
Clarke, E. M., O. Grumberg, D. A. Peled, 2000, 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, The altarica 3.0 project for model-based safety ssessment. In Proceedings of 4th IFAC Workshop on Dependable Control of Discrete Systems, DCDS'2013, York, Great Britain, pp. 127–132.
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, vol. 222(4), pp. 495–505.
Zimmermann, A, 1976, Stochastic Discrete Event Systems. Berlin, Heidelberg, Germany: Springer.