SYSTEME DE TRANSITIONS GARDEES : FORMALISME PIVOT DE MODELISATION POUR LA SURETE DE FONCTIONNEMENT

GUARDED TRANSITION SYSTEM : PIVOT MODELLING FORMALISM FOR SAFETY ANALYSIS

PROSVIRNOVA Tatiana, RAUZY Antoine
Ecole Polytechnique
Laboratoire d'Informatique(LIX)
91128 Palaiseau
Tel : (+33) 1 69 33 60 42
Fax : (+33) 1 69 33 30 14
prosvirnova@lix.polytechnique.fr, rauzy@lix.polytechnique.fr

Résumé

Cet article a pour objectif de présenter un formalisme pivot qui généralise les formalismes d'états/transitions utilisés dans le domaine de la sûreté de fonctionnement. Ce formalisme, appelé système de transitions gardées, est au cœur du projet AltaRica qui vise à développer une chaîne d'outils de traitement dédiée aux analyses de risque. Il sera possible de compiler tout formalisme d'états/transitions vers ce formalisme pivot et ensuite lui appliquer cette suite d'outils et réaliser différents types de traitement sur le même modèle. Différentes approches de modélisation de propagation des flux dans les systèmes pour ce formalisme pivot sont aussi discutées.

Summary

The purpose of this article is to present the pivot formalism for state/transition modelling formalisms used to perform safety analyses. This pivot formalism is called guarded transition system (GTS). GTS is a central part of AltaRica project which aims to develop a chain of tools dedicated to safety assessment. The compilation of other modelling formalisms to GTS offers the possibility to assess them with the tools developed for GTS. Different approaches for flow propagation modelling in systems for GTS are also discussed.

Introduction

Parmi les différentes classes de formalismes de modélisation des systèmes utilisés dans le domaine de la sûreté de fonctionnement, on distingue :
– les formalismes combinatoires ;
– les formalismes d'états/transitions.
Les formalismes combinatoires (arbres de défaillance(s), arbres d'événements, blocs-diagrammes de fiabilité) donnent une bonne approximation du comportement des systèmes. Des algorithmes de traitement efficaces ont été développés pour cette classe de modèles. Cependant ces modèles ne permettent pas de prendre en compte certains types de phénomènes, par exemple les reconfigurations des systèmes.
Les formalismes d'états/transitions rajoutent un pouvoir d'expression supplémentaire permettant de capturer l'ordre d'apparition des événements et d'exprimer différents types de dépendance(s). Ces formalismes sont un bon compromis entre le pouvoir d'expression et l'efficacité des algorithmes de traitement.
De nombreux formalismes d'états/transitions ont été inventés. Parmi ces formalismes, on peut citer :
– les chaînes de Markov ;
– les réseaux de Petri ;
– les arbres de défaillance(s) dynamiques ;
– les BDMPs (Boolean logic Driven Markov Processes [4]) ;
– Pepa/Pepa Nets [5] ;
– AltaRica [1].
Des algorithmes de traitement spécifiques ont été développés pour chacun de ces formalismes. Parmi ces différents algorithmes, on peut distinguer cinq grandes familles d'algorithmes :
– la compilation vers les arbres de défaillance(s) ;
– la compilation vers les chaînes de Markov ;
– la génération de séquences critiques ;
– la simulation stochastique ;
– le model-checking.
Dans cet article nous nous intéressons à un formalisme pivot pour les formalismes d'états/transitions qui serait une généralisation de tous les formalismes existants. Ce formalisme pivot doit posséder un certain nombre de bonnes propriétés.
Premièrement, il doit offrir des constructions de haut-niveau permettant de représenter le graphe d'états d'un système de manière implicite. Le graphe d'états d'un système, même relativement petit, peut être gigantesque et impossible à représenter graphiquement de manière compréhensible.
Ensuite, ce formalisme pivot doit aussi être capable de décrire le comportement d'un système par un automate d'états fini.
De plus, ce formalisme doit pouvoir exprimer la propagation de flux à travers le système, c'est-à-dire représenter les actions à distance. On retrouve en particulier le concept de flux qui se propage à travers un réseau dans le formalisme des blocs-diagrammes de fiabilité.
Enfin, ce formalisme pivot doit être compositionnel. On modélise les systèmes par composition à partir des sous-systèmes et des composants. Le formalisme pivot doit donc permettre de représenter le comportement d'un système de manière implicite à partir des représentations de comportements de ses composants. Les contraintes systémiques, tel que défaillances de causes

communes ou équipe unique de réparateurs partagé par plusieurs composants, sont modélisé par synchronisation des automates composés.
Nous avons choisi, pour ce formalisme pivot, le formalisme de Système de Transitions Gardées introduit dans [8] par l'un des auteurs de cette communication. Il possède toutes les bonnes propriétés des formalismes d'états/transitions. Dans le cadre du projet AltaRica, nous comptons développer une suite complète d'outils de traitement pour ce formalisme intégrant toutes les familles d'algorithmes citées plus haut. Ainsi, il sera possible de compiler tout formalisme d'états/transitions vers ce formalisme pivot et ensuite lui appliquer cette suite d'outils et réaliser différents types de traitement sur le même modèle.
Le reste de cette communication est organisé comme suit : la section 2 présente le projet AltaRica, dans lequel ce formalisme pivot occupe une place très importante, la section 3 explique les différentes notions du formalisme et discute de différents moyens pour modéliser la propagation des flux dans un système. La section 4 conclut cet article et présente quelques perspectives.

Le projet AltaRica

La sûreté de fonctionnement des systèmes est un domaine largement étudié. Les ingénieurs fiabilistes maîtrisent les formalismes de modélisation du risque tels que les Analyses des Modes de Défaillance et Evaluation de leur Criticité (AMDEC), les Arbres de Défaillance(s) ou les Arbres d'Evénements. Des algorithmes efficaces et des outils performants sont disponibles pour évaluer ces modèles.
Ces formalismes sont cependant très éloignés des descriptions fonctionnelles des systèmes. Maintenir les modèles tout au long du cycle de vie des systèmes est donc une tâche difficile, coûteuse et susceptible de comporter des erreurs. C'est pourquoi des langages de modélisation de haut niveau, comme AltaRica, ont été proposés et commencent à être utilisés dans l'industrie. Ces formalismes supportent une approche de la sûreté de fonctionnement dite « dirigée par les modèles ».
AltaRica est un langage formel permettant non seulement de décrire des composants sous forme d'automates d'états, mais aussi de créer des bibliothèques de modèles de composants et par ailleurs d'assembler ces modèles en des hiérarchies décrivant des systèmes complexes. Ce langage permet de décrire des modèles à un haut niveau d'abstraction, très proche de l'architecture fonctionnelle des systèmes. Il est entre autres possible d'associer aux modèles AltaRica des représentations graphiques les rendant très proches d'un "Process & Instrumentation Diagram". Le langage AltaRica est utilisé par plusieurs ateliers de sûreté de fonctionnement : Cecilia OCAS (Dassault Aviation), Simfia (EADS Apsys) et Safety Designer (Dassault Systèmes). De nombreuses expériences industrielles réussies ont été menées avec le langage AltaRica [2, 3].
La sémantique formelle du langage, basée initialement sur les automates de modes [7] et ensuite sur les systèmes de transitions gardées [8], a permis de développer des algorithmes de traitement efficaces :
- compilation vers les arbres de défaillance(s),
- génération de séquences critiques,
- génération de graphes de Markov,
- simulation stochastique, ou encore
- outils de model-checking.

L'objectif du projet AltaRica, développé au LIX, est de proposer une palette d'outils de traitement pour la nouvelle version du langage AltaRica. Ces outils seront diffusés en tant que logiciels libres.
Le schéma présenté ci-dessous donne les éléments du futur atelier de modélisation dédié à la sûreté de fonctionnement.

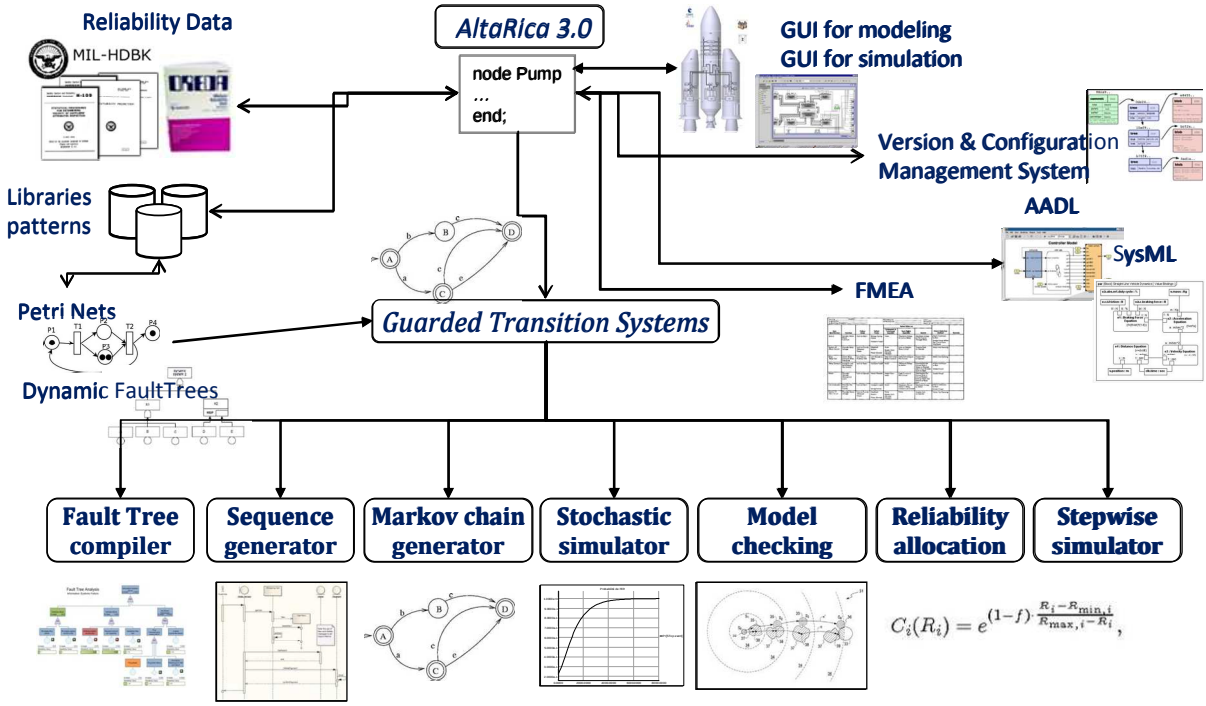


Figure 1. Le projet AltaRica

Au cœur de ce projet se trouve la nouvelle version du langage AltaRica. La partie inférieure du schéma présente les outils de traitement : générateur d'arbres de défaillance(s), générateur de séquences critiques, générateur de chaînes de Markov, simulateur stochastique, model-checker, module d'allocation de fiabilité, simulateur pas-à-pas. Ces outils ne s'appliquent pas au

langage lui-même mais à notre formalisme pivot vers lequel AltaRica est compilé. Cette compilation permet d'augmenter l'expressivité du langage sans changer les outils de traitement. Notre formalisme pivot joue le rôle de formalisme intermédiaire vers lequel peuvent être compilés d'autres langages, pas seulement AltaRica. Ce formalisme pivot est appelé « systèmes de transitions gardées ».
Les outils de traitement présentés offrent une panoplie complète permettant de réaliser différentes expériences virtuelles sur les modèles et de calculer différents types d'indicateurs fiabilistes (ainsi que de faire des vérifications croisées des résultats obtenus).
La partie supérieure du schéma présente les interfaces de création et de simulation des modèles, ainsi que des passerelles entre AltaRica et d'autres formalismes de modélisation.

Système de Transitions Gardées

1 Etats, événements, transitions

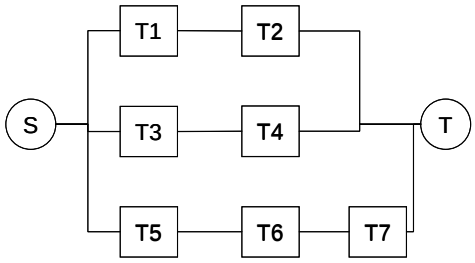


Figure 2. Système de production

Un système de transitions gardées est un type particulier d'automates, dans lequel les états sont représentés par des ensembles de couples *(v, c)*, où *v* est une variable d'état et *c* est sa valeur. Les changements d'états sont modélisés par les événements et les transitions. Une transition est un triplet <e, G, P>, aussi noté e: G ->P, où e est un événement, G est une expression Booléenne, aussi appelé une garde ou une pré-condition, et P est une instruction, appelée une post-condition. Si la garde G de la transition e: G->P est vérifiée dans l'état s et si l'événement e se produit, alors la post-condition P est exécutée et le système change d'état.
Prenons comme exemple le système présenté en Figure 2. C'est un système de production composé d'une source S, d'une cible T et de plusieurs unités de traitement T1, ..., T7. La source et les unités de traitement sont supposées défaillantes avec les taux de panne λs, λ1,..., λ7 et les taux de réparation μs, μ1, ..., μ7. Nous cherchons à déterminer la probabilité que T soit alimentée durant un temps de mission donné. Le comportement de ce système peut être représenté par un système de transitions gardées. Pour représenter l'état du système il faut des variables booléennes représentant l'état de chaque composant : *S.working, T1.working, ..., T7.working*. Chaque unité de traitement peut tomber en panne ou être réparée, ceci est représenté par les événements suivants : *S.failure, s.repair, T1.failure, T1.repair, ..., T7.failure, T7.repair*. Les changements d'états sont représentés par les transitions suivantes :
- *T1.failure*: *T1.working* == true -> *T1.working* := false (composant T1 va de l'état *T1.working*==true dans l'état *T1.working*==false quand l'événement T1.failure se produit) ;
- *T1.repair*: *T1.working* == false -> *T1.working* := true (composant T1 va de l'état *T1.working*==false dans l'état *T1.working*==false quand l'événement *T1.repair* se produit).

Il est de même pour les transitions de tous les autres composants du système.
Un système de transitions gardées peut-être étendu en système de transitions gardées stochastique de la même manière que les réseaux de Pétri sont étendus en réseaux de Petri stochastiques. Un délai, éventuellement stochastique, est associé à chaque événement d'un système de transitions gardées.
Les transitions sont divisées en deux catégories :
- les transitions immédiates qui sont tirées dès qu'elles sont tirables, c'est-à-dire dès que leur garde est satisfaite ;
- les transitions temporisées qui demandent un certain délai, éventuellement stochastique, pour être tirées.

1.1 Transitions immédiates

Le délai associé avec les transitions immédiates est nul. Elles doivent être tirées dès que leurs gardes sont vérifiées. En plus du délai nul, on associe aux événements des transitions immédiates un nombre positif - leur poids. Si dans un état plusieurs transitions immédiates sont tirables en même temps (conflit), chacune d'entre elles a une probabilité non nulle d'être tirée :

$$p(t_i) = \frac{weight(t_i)}{\Sigma_{t\ tirable} weight(t)}$$

Cette probabilité permet de déterminer quelle transition parmi toutes les transitions immédiates tirables en même temps sera tirée en premier.

1.2 Transitions temporisées

Un délai non nul, éventuellement stochastique, est associé à chaque événement d'une transition temporisée. Par défaut, on considère que les transitions sont temporisées et qu'un événement associé à une transition temporisée peut se produire, avec la même probabilité, à n'importe quel instant dans l'intervalle [0, +∞[.
Si une transition devient tirable à l'instant *t*, on détermine son délai *d*. Pour déterminer le délai *d*, on tire un nombre aléatoire dans l'intervalle [0, 1] - probabilité d'occurrence de l'événement. Le délai d est obtenu en inversant la distribution de probabilité associé à l'événement (cf. Figure 3).

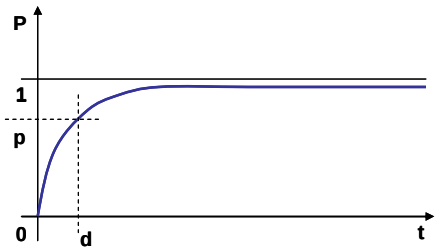


Figure 3. Calcul du délai

La transition sera effectivement tirée à l'instant t+d si :
— Sa garde continue d'être satisfaite entre t et t+d.
— L'expression déterminant son délai n'a pas été modifiée entre t et t+d.
Deux transitions temporisées devant être tirées à une même date ont la même probabilité d'être tirées. Un modèle bien conçu devrait dans ce cas vérifier la « règle du losange » : le tir de la transition « a » suivi du tir de la transition « b » doit conduire au même état que le tir de la transition « b » suivi du tir de la transition « a » (cf. Figure 4).

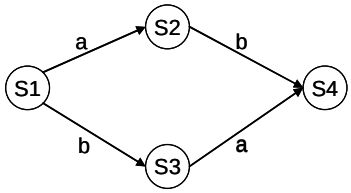


Figure 4. Règle du losange

2 Propagation des flux

Dans l'exemple Figure 2, en plus des états, des événements et des transitions, on aimerait aussi décrire la propagation des flux depuis la source S jusqu'à la cible T. Il serait possible de représenter la propagation des flux par des variables Booléennes :
- *S.out, T1.in, T1.out, ..., T7.in, T7.out, T.in*,
et un ensemble d'équations :
- *S.out = S.working* (le flux sortant de la source est présent si la source est en marche);
- *T1.in = S.out* (le flux en entrée de T1 est le même que le flux en sortie de la source S);
- *T1.out = T1.working and T1.in* (le flux en sortie de T1 est présent si le flux en entrée est présent et si T1 est en marche);
- ...,
- *T7.in = T6.out*,
- *T7.out = T7.working and T7.in*,
- *T.in = T2.out or T4.out or T7.out* (la cible T est alimentée si le flux en sortie de l'une des trois unités T2, T4 ou T7 est présente).
Si la source S tombe en panne, la propagation des flux se fait comme montrée sur la Figure 5 (l'absence de flux est représentée par un trait en pointillé, la direction de propagation des flux est indiquée par des flèches).

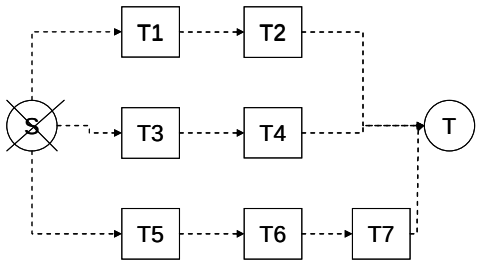


Figure 5. Système de production en panne

On considère maintenant un système un peu différent : un réseau électrique représenté sur la Figure 6. Dans ce réseau, S1 et S2 sont tous les deux les sources d'électricité qui alimentent les unités T1, ..., T7. Ces deux sources S1 et S2, ainsi que toutes les unités T1, ..., T7, peuvent tomber en panne et être réparées. Les connections entre les unités sont supposées sans défaillance. Les taux de panne et de réparation sont supposés connus. On cherche à déterminer la probabilité qu'une unité donnée soit alimentée en électricité sans interruption pendant un temps de mission donné. On souhaite modéliser le comportement de ce réseau par un système de transitions gardées. Comme dans le cas du système de production les états du système seront représentés par les variables d'état de chaque composant *S1.working, S2.working, T1.working, ..., T7.working* et leurs valeurs. En plus des événements décrits pour le système de production, la source électrique S2 peut aussi tomber en panne et être réparée (*S2.failure* et *S2.repair*). Les changements d'états sont représentés par un ensemble de transitions comme pour le système de production. Pour modéliser l'alimentation en électricité des unités, on introduit les variables Booléennes *S1.powered, S2.powered, T1.powered, ..., T7.powered*. Pour modéliser la propagation de l'électricité dans ce système il faut un ensemble d'équations, par exemple comme celui qui suit :
- *S1.powered = S1.working*;

- *S2.powered* = *S2.working*;
- *T1.powered* = *T1.working* and (*S1.powered* or *T2.powered*);
- *T2.powered* = *T2.working* and (*S2.powered* or *T1.powered* or *T4.powered*);
- *T3.powered* = *T3.working* and (*S1.powered* or *T4.powered* or *T5.powered* or *T6.powered*);
- *T4.powered* = *T4.working* and (*S2.powered* or *T2.powered*);
- *T5.powered* = *T5.working* and (*T3.powered* or *T6.powered*);
- *T6.powered* = *T6.working* and (*T3.powered* or *T5.powered* or *T7.powered*);
- T7.powered = *T7.working* and (*S2.powered* or *T6.powered*).

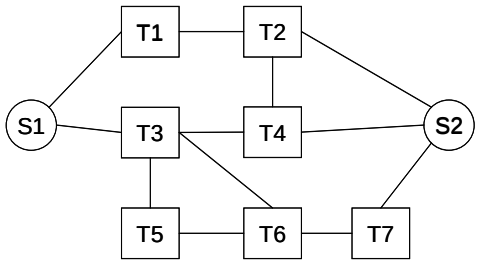


Figure 6. Réseau électrique

La source S1 et les unités T4 et T7 tombent en panne. On s'attend à avoir une solution représentée sur la Figure 7 avec les unités T3, T5 et T6 qui ne sont pas alimentées en électricité. Mais comme vous pouvez le constater, deux solutions satisfont le système d'équations décrit plus haut :
- Première solution : *T3.powered* = false, *T5.powered* = false, *T6.powered* = false ;
- Deuxième solution : *T3.powered* = true, *T5.powered* = true, *T6.powered* = true.

La deuxième solution ne correspond pas à une situation réelle. Les deux solutions sont possibles à cause de l'interdépendance des variables *T3.powered*, *T5.powered* et *T6.powered*. Ce type de systèmes est appelé système bouclé. Les systèmes bouclés présentent des difficultés particulières de modélisation et de traitement.

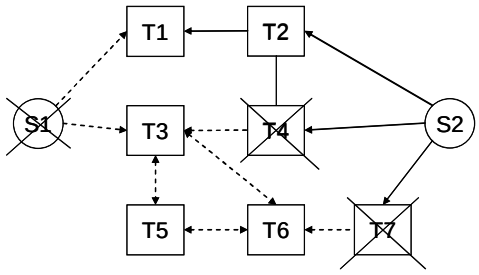


Figure 7. Réseau électrique : S1, T4 et T7 en panne

2.1 Assertions

Les variables *S.powered, T1.powered, ... T7.powered* dans l'exemple ci-dessus sont appelées les variables de flux. À la différence des variables d'états, les variables de flux sont utilisées pour modéliser les flux d'information qui circulent à travers le système. Elles peuvent représenter des connections physiques entre composants, des actions de contrôle commande, l'alimentation électrique, etc.
Les équations présentées ci-dessus sont appelées les assertions. Elles expriment les dépendances entre les variables de flux et les variables d'état. Les assertions sont utilisées pour propager les valeurs des variables de flux à travers le système.
Différentes approches ont été proposées pour calculer les assertions :
- Résolution des contraintes ;
- DataFlow ;
- Point Fixe ;
- DataFlow dynamique.

Chacune de ces méthodes présente ses avantages et ses inconvénients. Elles sont discutées dans la suite.

2.1.1 Résolution des contraintes

Dans [6] les auteurs proposent de considérer les assertions comme un système de contraintes. C'est la solution la plus générique qui permet d'exprimer tout type de dépendances entre les variables. Le système de contraintes peut cependant avoir plusieurs solutions ou ne pas en avoir du tout et ce dernier fait est impossible à détecter à la compilation. Ainsi un appel au solveur de contraintes doit être fait à chaque tir de transition, ce qui ajoute une complexité de traitement supplémentaire.
Si on modélise le système de production Figure 2, les assertions sont représentées par le système de contraintes suivant :
- *S.out* = *S.working*,
- *T1.in* = *S.out*,
- *T1.out* = *T1.working and T1.in*,
- ...,
- *T7.in* = *T6.out*,
- *T7.out* = *T7.working and T7.in*,
- *T.in* = *T2.out or T4.out or T7.out*.

Il est important de noter que la contrainte *T7.in* = *T6.out* est équivalente à *T6.out* = *T7.in,* c'est-à-dire que la direction de flux ne doit pas être connue à l'avance. Ceci est un avantage par rapport à d'autres approches

Dans le cas de l'exemple du système bouclé mentionné plus haut (Figure 6), le système de contraintes admet deux solutions dont l'une (*T3.powered* = true, *T5.powered* = true, *T6.powered* = true) ne correspond pas au comportement réel du système.

2.1.2 DataFlow

Dans [8], l'un des auteurs de cette communication propose une version simplifiée du système de contraintes : des contraintes DataFlow. C'est-à-dire que la direction des flux est connue d'avance et qu'il existe un ordre précis d'exécution des contraintes pour calculer toutes les variables de flux. En d'autres termes, le graphe de dépendance(s) des variables de flux ne possède pas de cycle et permet de déterminer par parcours en profondeur l'ordre dans lequel les variables sont calculées. Donc il existe toujours une solution et elle est unique.
Dans l'exemple du système de production, mentionné plus haut (Figure 2), les contraintes DataFlow décrivant la propagation des variables de flux sont comme suit :
- *S.out* ← *S.working*,
- *T1.in* ← *S.out*,
- *T1.out* ← *T1.working and T1.in*,
- ...,
- *T7.in* ← *T6.out*,
- *T7.out* ← *T7.working and T7.in*,
- *T.in* ← *T2.out or T4.out or T7.out*.

Le graphe de dépendance entre les variables de flux est donné en Figure 8 où les variables d'état sont marquées en gris. Ce graphe ne possède pas de cycle et permet de déterminer l'ordre de calcul de toutes les variables. Si la source S tombe en panne, alors l'unique solution calculée en une passe est :
- *S.out* = false, *T1.in* = false, *T1.out* = false, ..., *T7.in* = false, *T7.out* = false, *T.in* = false.

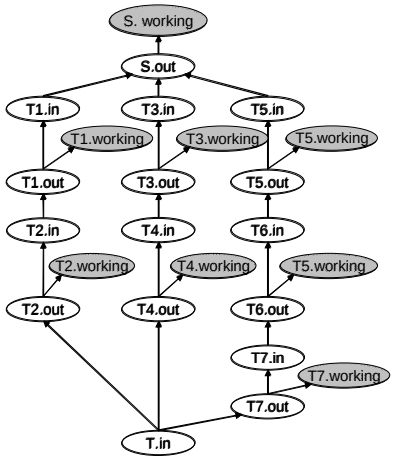


Figure 8. Système de production : graphe de dépendance entre les variables de flux

L'avantage de cette approche est qu'il est possible de vérifier à la compilation que le modèle est correct. La complexité réduite de calcul des assertions a permis de développer des outils de traitement efficaces, qui sont utilisés en industrie [2, 7].
Cependant, il est obligatoire de connaître à l'avance la direction de propagation de flux. Aussi, il est impossible de représenter les systèmes bouclés, tel que le réseau électrique présenté Figure 6.

2.1.3 Point Fixe

Pour être capable de traiter des systèmes bouclés (par exemple, le réseau électrique de la Figure 6), dans [8] l'un des auteurs de cette communication propose de considérer les assertions comme des instructions et de les calculer par point fixe.
Il est donc obligatoire, comme dans le cas précédent, de connaître à l'avance la direction de propagation des flux dans le système. Les instructions de l'exemple de la Figure 6 sont :
- *S1.powered* := *S1.working*;
- *S2.powered* := *S2.working*;
- *T1.powered* := *T1.working* and (*S1.powered* or *T2.powered*);
- *T2.powered* := *T2.working* and (*S2.powered* or *T1.powered or T4.powered*);
- T3.powered := *T3.working* and (*S1.powered* or *T4.powered* or *T5.powered* or *T6.powered*);
- *T4.powered* := *T4.working* and (*S2.powered* or *T2.powered*);
- *T5.powered* := *T5.working* and (*T3.powered* or *T6.powered*);
- *T6.powered* := *T6.working* and (*T3.powered* or *T5.powered* or *T7.powered*);
- *T7.powered* := *T7.working* and (*S2.powered* or *T6.powered*).

Chaque variable de flux doit aussi avoir une valeur par défaut, ce qu'on appelle une valeur de reset. À chaque tir de transition, premièrement l'action de la post-condition est exécutée, ensuite toutes les variables de flux reçoivent leurs valeurs pas défaut, enfin les assertions sont exécutées en dernier par point fixe (en boucle afin d'atteindre un état stable quand aucune des variables ne change plus de valeur). Le point fixe peut ne pas exister.
Dans l'exemple du réseau électrique de la Figure 6, les valeurs par défaut des variables de flux sont :
- *S1.powered* = false, *S2.powered* = false, *T1.powered* = false, ..., *T7.powered* = false.

Supposons que dans l'état initial, les composants S1, T4 et T7 soient en panne (Tableau 1.1). Ensuite le composant T7 est réparé. Quand la transition associée à l'événement T7.repair est tirée, d'abord l'action de la transition est exécutée *T7.working*:=true (Tableau 1.2), ensuite les variables de flux *T3.powered*, *T4.powered, T5.powered*, *T6.powered*, *T7.powered* reçoivent leur valeur par défaut - false (Tableau 1.3), enfin ces variables sont recalculées par point fixe et reçoivent la valeur true (Tableau 1.4).

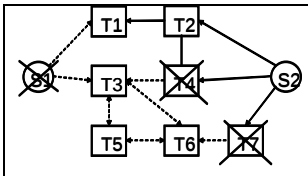
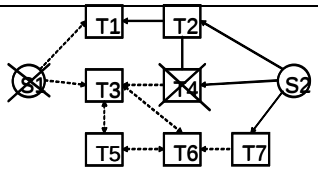
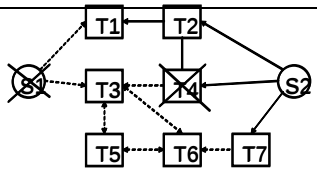
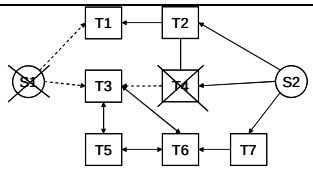
 1. Etat initial	 2. L'unité T7 est réparé	 3. Reset des variables de flux	 4. Calcul des variables de flux par point fixe

Tableau 1. Calcul des assertions par point fixe

Il est impossible d'assurer à la compilation l'existence du point fixe. Le résultat de calcul du point fixe peut aussi dépendre de l'ordre dans lequel sont exécutées les instructions.

2.1.4 DataFlow dynamique

L'approche exposée dans la section précédente permet de modéliser les systèmes bouclés mais exige de connaître au préalable la direction de propagation des flux. Il existe des situations dans lesquelles il est pratique de ne pas spécifier la direction des flux, c'est-à-dire pouvoir écrire x :=: y, ce qui est équivalent à x:= y ou à y:=x.
L'approche de DataFlow dynamique consiste à déterminer la direction de propagation des flux à l'exécution. Après chaque tir de transition, les valeurs des variables d'état et de flux sont mises à jour en deux étapes. D'abord l'action de la transition est exécutée et ensuite toutes les assertions sont exécutées en parallèle. L'ordre dans lequel les assertions sont exécutées ne doit pas influencer le résultat et dans le cas où le résultat dépend de cet ordre d'exécution des assertions, le modèle est considéré incorrect. Dans le cas où après la propagation il reste des variables non affectées, l'une des variables est choisie et est mise à jour à sa valeur par défaut (valeur préférée), puis cette valeur est à nouveau propagée. Le processus est répété jusqu'à ce que toutes les variables soient affectées.
Le DataFlow dynamique offre une capacité d'expression supplémentaire. Il permet de modéliser les systèmes bouclés et autorise à ne pas spécifier la direction des flux. Cependant le choix des valeurs de reset pour les variables de flux est crucial, il est de même dans le cas du point fixe. La solution peut ne pas exister.
On peut représenter les assertions de l'exemple de production comme suit :
- *S.out := S.working*,
- *T1.in :=: S.out*,
- *T1.out := T1.working and T1.in*,
- ...,
- *T7.in :=: T6.out*,
- *T7.out := T7.working and T7.in*,
- *T.in := T2.out or T4.out or T7.out*.

Le Tableau 2 donne un aperçu de comparaison des quatre approches selon quelques critères :
- la capacité à modéliser les flux non dirigés (x = y est équivalent à y = x) ;
- l'assurance de l'existence d'une solution unique à la compilation ;
- la capacité à modéliser les systèmes bouclés ;
- l'efficacité des algorithmes de traitement.

	Contraintes	**DataFlow**	**Point Fixe**	**DataFlow dynamique**
Flux non dirigés	Oui	Non	Non	Oui
Solution unique assurée à la compilation	Non	Oui	Non	Non
Modélisation des systèmes bouclés	Non	Non	Oui	Oui
Efficacité de traitement (de 1 à 3)	*	***	**	**

Tableau 2. Assertions

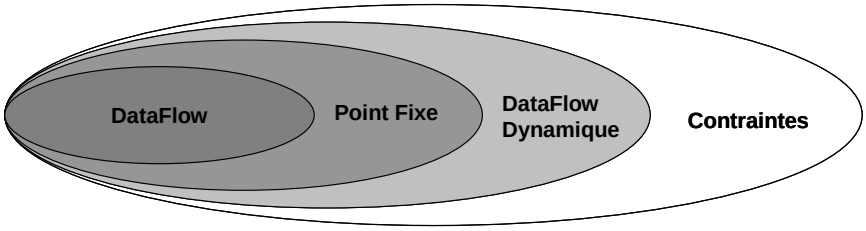


Figure 9. Assertions

Le DataFlow offre l'efficacité de traitement et permet de déterminer l'existence de la solution à la compilation mais pose des limites sur la modélisation. Si on cherche à augmenter le pouvoir d'expression du formalisme (à modéliser les systèmes bouclés ou les flux non dirigés), il n'est plus possible de garantir l'existence de la solution unique à la compilation et l'efficacité de traitement diminue. L'objectif est de trouver un compromis entre l'efficacité de traitement et le pouvoir d'expression.

En résumé, un système de transitions gardées est un n-uplet <V, E, T, i, A>, où
- V est un ensemble de variables ;
- E est un ensemble d'événements ;
- T est un ensemble de transitions. Une transition est un triplet <e, G, P>, aussi noté e: G -> P, où e est un événement de E, G est une expression Booléenne, aussi appelée une garde ou une pré-condition, et P est une instruction, aussi appelée une post-condition.
- i est un ensemble d'initialisations, c'est-à-dire un ensemble de couples (v, c) où v est une variable de V et c est une constante, (sa valeur initiale) ;
- A est un ensemble d'assertions.

Les assertions permettent de calculer la propagation des valeurs des variables de flux. Différentes approches pour représenter des assertions ont été proposées. Chacune d'elles présente ses avantages et ses inconvénients.
La sémantique d'un système de transitions gardées est une structure de Kripke, c'est-à-dire un graphe G = (S, E, T) dont les sommets sont définis par des couples (v, c), où v est une variable d'état et c est sa valeur, et les arêtes sont étiquetées par des événements :
- L'état initial A(i) est un élément de S.
- Si s est un élément de S, e: G->P est une transition tirable dans l'état s et s' est l'état atteint après le tir de la transition, alors s' est un élément de S et la transition e: s ->s' est un élément de T.

Le graphe G peut avoir un nombre infini d'états ou, si toutes les variables prennent leur valeur dans un domaine fini, un nombre fini, mais potentiellement exponentiellement grand par rapport à la taille du système de transitions gardées.

Conclusion

Dans cette communication, nous avons présenté le formalisme de système de transitions gardées - un formalisme pivot pour tous les formalismes d'états/transitions utilisées dans le domaine de la sûreté de fonctionnement. Ce formalisme possède toutes les bonnes propriétés des formalismes d'états/transitions :
- Il permet de représenter le comportement sous forme d'un automate d'états fini.
- Il a la capacité de représenter le graphe d'états du système de manière implicite.
- Il est compositionnel.
- Il offre les moyens pour décrire la propagation des flux à travers le système.

Le formalisme de système de transitions gardées est au cœur du projet AltaRica. C'est un formalisme de bas niveau qui est en entrée de tous les outils de la chaîne de traitement :
- Compilateur vers les arbres de défaillance(s) ;
- Générateur de séquences critiques ;
- Générateur de graphes de Markov ;
- Simulateur stochastique ;
- Model-checker.

Tous les formalismes de haut-niveau, y compris les modèles AltaRica, sont d'abord compilés vers les systèmes de transitions gardées pour ensuite être efficacement traités par les outils. Il est ainsi envisageable de compiler d'autres formalismes d'états/transitions vers les systèmes de transitions gardées pour leur appliquer la suite d'outils, développée pour les GTS, en vue de calculer les indicateurs fiabilistes et faire des vérifications croisées des résultats obtenus.
Aussi dans cette communication, nous avons discuté de différentes approches de propagation des flux :
- résolutions de contraintes,
- DataFlow,
- Point fixe,
- DataFlow dynamique.

Toutes ces approches ont leurs avantages et inconvénients et un bon compromis entre la capacité d'expression et l'efficacité de traitement est le DataFlow dynamique. Cette approche permet de modéliser les systèmes bouclés. La direction des flux est détectée dynamiquement pendant l'exécution du modèle et une bonne implémentation garantie l'efficacité du traitement. Cependant, il est impossible de vérifier la justesse du modèle à la compilation. Dans le cas de plusieurs solutions possibles, la solution préférée par l'utilisateur (via valeurs par défaut/préférées) est choisie.
La suite de nos travaux s'oriente vers l'approche DataFlow dynamique pour les systèmes de transitions gardées et se focalise sur l'adaptation des algorithmes de calcul pour cette approche.

3 Références

1. ARNOLD, A. GRIFFAULT, G. POINT, A. RAUZY, 2000, The AltaRica Language and its Semantics, Fundamenta Informaticae, 34:109–124.
2. R. Bernard, J.-J. Aubert, P. Bieber, C. Merlini, S. Metge, 2007, Experiments in model-based safety analysis: flight controls, in Proceedings of IFAC workshop on Dependable Control of Discrete Systems.
3. M. BOITEAU, Y. DUTUIT, A. RAUZY, J.-P. SIGNORET, 2006, The AltaRica Data-Flow language in use: Assessment of Production Availability of a MultiStates System, Reliability Engineering and System Safety, 91:747-755.
4. M. BOUISSOU, J.L. BON, 2003, A new formalism that combines advantages of fault-trees and Markov models: Boolean logic Driven Markov Processes, Reliability Engineering and System Safety, 82(2):149-163.
5. S. GILMORE, J. HILLSTON, L. KLOUL, M. RIBAUDO, 2003, Pepa Nets: a structured performance modelling formalism, Performance Evaluation, 54:79 – 104.
6. G.Point, A. Rauzy, AltaRica: Constraint automata as a description language, Journal Européen des Systèmes Automatisés, 33(8-9):1033 - 1052, 1999.
7. A. RAUZY, 2002, Modes Automata and their compilation into Fault Trees. Reliability Engineering and System Safety, 78:1-12.
8. A. RAUZY, 2008, Guarded transition systems: a new state/events formalism for reliability studies. Reliability Engineering and System Safety, 222(4):495-505.