Systèmes embarqués communicants
Approches formelles

AFSEC

version 1

20 novembre 2007

Table des matières

Chapitre 1

Outils pour le Model-Checking de Systèmes Temporisés

1.1. Introduction

Dans ce chapitre nous présentons différents outils pour la vérification des systèmes temps-réels. UPPAAL [LAR 97a, BEH 04b] est un outil de model-checking pour les systèmes temps-réels développé conjointement par les universités d'Uppsala et d'Aalborg. La première version d'UPPAAL était delivrée en 1995 [LAR 97a] et a été en développement constant depuis [BEN 98, AMN 01, BEH 01a, BEH 02, DAV 02, DAV 03, DAV 06]. Il a été appliqué avec succès à des études de cas variées allant de protocoles de communication à des applications multimédias [HAV 97, LÖN 97, D'A 97, BOW 98, HUN 00, IVE 00, DAV 00, LIN 01]. L'outil est conçu pour vérifier les systèmes qui peuvent être modélisés en tant que réseaux d'automates temporisés [ALU 90a, ALU 90b, HEN 92, ALU 94] étendus avec des variables entières, des types de données structurées, des fonctions définies par l'utilisateur et des synchronisations par canaux de communication. UPPAAL-CORA est une version spécialisée dUPPAAL qui implémente des algorithmes guidés et minimaux d'atteignabilité étendus avec du coût [BEH 01b, BEH 01c, LAR 01]. L'outil est bien adapté aux problèmes d'ordonnancement à coûts minimaux [BEH 05a, BEH 05b]. UPPAAL-TIGA [BEH 07] est une spécialisation d'UPPAAL conu pour vérifier les systèmes modélisés en tant qu'automates de jeux temporis'es où un contrôleur joue contre un environement. L'outil synthetise du code représenté comme une stratégie pour atteindre un objectif de contrôle [DEA 01, ASA 98, MAL 95, TRI 99]. L'outil est basé sur un

Chapitre rédigé par Alexandre DAVID et Gerd Behrmann et Kim G. Larsen et Paul Pettersson et Jacob Illum Rasmussen et Wang Yi et Morgan Magnin.

algorithme récent de recherche à la volée [CAS 05] et a déjà été appliqué à une étude de cas industriel [JES 07]. Roméo [**?**] est un outil de model-checking pour les réseaux de Pétri temporels [**?**] et une de leur extension permettant de modéliser la préemption [**?**]. Il est développé à l'Institut de Recherche en Communications et Cybernétique de Nantes (IRCCyN). La première version de Roméo a vu le jour en 2001. Le logiciel est depuis l'objet de développements réguliers. Ces travaux visent notamment à élargir la classe de modèles et de propriétés qu'il est possible de vérifier. C'est ainsi que Roméo permet désormais de vérifier des propriétés temporelles quantitatives sur des réseaux de Petri temporels [**?**] mais également sur des modèles intégrant des mécanismes de suspension/reprise de tâches [**?**]. Il propose également une large gamme de traductions vers d'autres modèles (traductions préservant certaines classes de propriétés)

Dans ce chapitre nous présentons l'architecture de ces outils, les algorithmes basiques pour le model-checking et les techniques principales développées ces dernières années pour améliorer les performances à la fois en temps et en mémoire.

1.2. UPPAAL

1.2.1. *Automates Temporisés et Exploration Symbolique*

UPPAAL est basé sur une extension des automates temporisés. Un automate temporisé est une machine d'états finie étendue avec des variables d'horloge. Le modèle sous-jacent est un modèle de temps dense où une variable d'horloge est évaluée à un nombre réel. Toutes les horloges avancent en même temps. Un système est modélisé en tant que réseau de tels automates en parallèle. De plus, le modèle est étendu avec les variables bornées entières. Le language de requêtes utilisé pour specifier les propriétés a vérifier est un sous ensemble de TCTL (timed computation tree logic en anglais) [ALU 90a, HEN 94, BAI 08].

Un état du système est défini par les localités des automates, les valeurs des horloges et les valeurs des variables entières. Le système change d'état en prenant une transition qui peut comprendre un arc dans n'importe quel automate qui peut prendre cet arc ou plusieurs arcs dans le cas de synchronisation (par paire ou diffusion).

Un automate temporisé est un graph dirigé annoté par des conditions et des mises-à-zero de variables d'horloge positives. Nous rappelons ici la définition d'un automate temporisé (**??**). Nous omettons ici F et R.

Définition 1.2.1 (Automate temporisé) *[HEN 94] Un* automate temporisé *est un 6-uplet* $(L, l_0, X, \Sigma, E, Inv)$ *où*

– L *est un ensemble fini de* localités *;*

– l_0 *est la* localité initiale *;*

– X *est un ensemble fini d'*horloges $\tilde{A}$ *valeurs réelles positives ;*

– Σ *est un ensemble fini d'*actions *;*

– $E \subset L \times \mathcal{C}(X) \times \Sigma \times 2^X \times 2^{X^2} \times L$ *est un ensemble fini d'*arcs*. Soit* $e = (l, \delta, \alpha, R, \rho, l') \in E$*. e est l'arc reliant la localité l à la localité l', avec la* garde *δ, l'action α, l'ensemble d'horloges à remettre à zéro R et la fonction d'affectation d'horloges ρ.*

– $Inv \in \mathcal{C}(X)^L$ *associe un* invariant *à chaque localité.*

Nous rappelons qu'une valuation d'horloge (**??**) ν sur un ensemple de variables X est un élément de $\mathbb{R}^X_{\geq 0}$. Pour $\nu \in \mathbb{R}^X_{\geq 0}$ et $d \in \mathbb{R}_{\geq 0}$, $\nu + d$ denote la valuation definie par $(\nu + d)(x) = \nu(x) + d$, et pour $X' \subseteq X$, $\nu[X' \mapsto 0]$ denote la valuation ν' avec $\nu'(x) = 0$ pour $x \in X'$ et sinon $\nu'(x) = \nu(x)$. Nous donnons maintenant la sémantique d'un automate temporisé (1.2.2).

Définition 1.2.2 (Sémantique d'un automate temporisé) *La sémantique d'un automate temporisé $\mathcal{A}$ est définie sous la forme d'un système de transitions temporisé $\mathcal{S}_\mathcal{A} = (Q, q_0, \Sigma, \rightarrow)$ où*

– $Q = L \times (\mathbb{R}^+)^X$ *;*

– $q_0 = (l_0, \mathbf{0})$ *;*

– $\rightarrow \in Q \times (\Sigma \cup \mathbb{R}) \times Q$ *est la relation définie pour $a \in \Sigma$ et $d \in \mathbb{R}^+$ par :*

- la relation de transition discrète *:* $(l, \nu) \xrightarrow{a} (l', \nu')$ *ssi* $\exists (l, \delta, a, R, \rho, l') \in E$ *telle que*

$$\begin{cases} \delta(\nu) = \texttt{true}, \\ \nu' = \nu[R \leftarrow 0][\rho], \\ Inv(l')(\nu') = \texttt{true} \end{cases}$$

- la relation de transition continue *:* $(l, \nu) \xrightarrow{\epsilon(d)} (l, \nu')$ *ssi*

$$\begin{cases} \nu' = \nu + d, \\ \forall d' \in [0, d], Inv(l)(\nu + d') = \texttt{true} \end{cases}$$

Le problème de l'exploration est que la sémantique donne un système de transitions infini. Il existe une abstraction exacte et finie basée sur les polyhèdres convexes dans $\mathbb{R}^X$ appelée zone [YI 94, LAR 95] (une zone peut être représentée par une conjonction dans $\mathcal{C}(X)$). Cette abstraction mène à la sémantique symbolique d'automates temporisés (TA) suivante :

Définition 1.2.3 (Sémantique symbolique de TA) *Soit $Z_0 = Inv(l_0) \wedge \bigwedge_{x,y \in X} x = y = 0$ la zone initiale. La sémantique symbolique d'un automate temporisé $(L, l_0, X, \Sigma, E, Inv)$ sur X est définie par le système de transitions $\langle \mathcal{S}, \int_0, \Rightarrow \rangle$ appelé le* graphe d'atteignabilité symbolique*, où $\mathcal{S} \subseteq L \times \mathcal{C}(X)$ est l'ensemble des états symboliques, $\int_0 = (l_0, Z_0)$ est l'état initial $\Rightarrow$ est la relation de transition définie par les règles suivantes :*

– $(l, Z) \stackrel{\delta}{\Rightarrow} (l, \mathsf{widen}(m, (Z \wedge Inv(l))^{\uparrow} \wedge Inv(l)))$ *et*

– $(l, Z) \stackrel{e}{\Rightarrow} (l', r(\gamma \wedge Z \wedge Inv(l)) \wedge Inv(l'))$ *si* $e = (l, \gamma, a, R, l') \in E$,

où $Z^{\uparrow} = \{u + d \mid u \in Z \wedge d \in \mathbb{R}_{\geq 0}\}$ *(l'opération de* futur*) et* $r(Z) = \{[x \mapsto 0]u \mid x \in R, u \in Z\}$ *(l'opération de mise-à-zéro). La fonction* widen : $\mathbf{N} \times \mathcal{C}(X) \rightarrow \mathcal{C}(X)$ *élargit les contraintes d'horloge par rapport à la constante maximale* m *de l'automate temporisé. Cette opération est aussi appelée normalisation [YI 94] ou extrapolation.*

La relation $\stackrel{\delta}{\Rightarrow}$ représente les transitions de délai et $\stackrel{e}{\Rightarrow}$ les transitions des arcs. La représentation classique d'une zone est la matrice des différences de bornes (DBM – difference bound matrix) [ROK 93, WON 94, BEN 02].

Dans UPPAAL, les automates temporisés sont mis en parallle dans un réseau d'automates sur un ensemble partagé d'horloges et d'actions, ce qui consiste en n automates temporisés $\mathcal{A}_i = (L_i, l_i^0, X, \Sigma, E_i, Inv_i)$, $1 \leq i \leq n$. Un vecteur de localité est un vecteur $\bar{l} = (l_1, \ldots, l_n)$. Nous composons les fonctions d'invariant en une fonction commune sur les vecteurs de localité $I(\bar{l}) = \wedge_i Inv_i(l_i)$. Nous écrivons $\bar{l}[l_i'/l_i]$ pour signifier le vecteur où le i-ème élement l_i de $\bar{l}$ est remplacé par l_i'. Nous définissons la sémantique d'un réseau d'automates temporisés (NTA – network of timed automata) comme suit :

Définition 1.2.4 (Sémantique de NTA) *Soit* $\mathcal{A}_i = (L_i, l_i^0, X, \Sigma_\varepsilon, E_i, Inv_i)$ *un réseau de* n *automates temporisés. Soit* $\bar{l}_0 = (l_1^0, \ldots, l_n^0)$ *le vecteur de localités initial. La sémantique est définie comme un système de transitions* $\langle S, s_0, \rightarrow \rangle$*, où* $S = (L_1 \times \cdots \times L_n) \times \mathbb{R}^X$ *est l'ensemble des états,* $s_0 = (\bar{l}_0, u_0)$ *est l'état inital, et* $\rightarrow \subseteq S \times S$ *est la relation de transition definie comme suit :*

– $(\bar{l}, u) \stackrel{d}{\rightarrow} (\bar{l}, u + d)$ *si* $\forall d' : 0 \leq d' \leq d \implies u + d' \in I(\bar{l})$.

– $(\bar{l}, u) \stackrel{a}{\rightarrow} (\bar{l}[l_i'/l_i], u')$ *s'il existe* $l_i \stackrel{\epsilon, \gamma, R_i}{\longrightarrow} l_i'$ *tel que* $u \in \gamma$, $u' = [R_i \mapsto 0]u$ *et* $u' \in I(\bar{l}[l_i'/l_i])$.

– $(\bar{l}, u) \stackrel{a}{\rightarrow} (\bar{l}[l_j'/l_j, l_i'/l_i], u')$ *s'il existe* $l_i \stackrel{\gamma_i, c?, R_i}{\longrightarrow} l_i'$ *et* $l_j \stackrel{\gamma_j, c!, R_j}{\longrightarrow} l_j'$ *tels que* $u \in (\gamma_i \wedge \gamma_j)$, $u' = [R_i \cup R_j \mapsto 0]u$ *et* $u' \in I(\bar{l}[l_j'/l_j, l_i'/l_i])$.

La sémantique symbolique des automates temporisés est étendue naturellement aux réseaux d'automates temporisés. Nous omettons les conditions sur S qui definissent la validité des vecteurs de localité de 2^L, entre autre, que ces localités n'appartiennent pas au même automate. De plus, nous omettons par soucis de simplicté les variables entières gérées par UPPAAL. Les définitions peuvent être étendues en rajoutant un ensemble d'entiers et d'actions sur ces entiers qui feraient partie des états. Les extensions supplémentaires gérées par UPPAAL sont :

– *Les canaux de diffusion.* Si un canal de communication *c* est declaré *broadcast* (diffusion), alors un processus qui prend une action *c !* se synchronise avec tous les autres processus qui peuvent prendre l'action *c ?*.

– *Canaux urgents.* Si un canal est déclaré *urgent* alors le temps ne peut pas s'écouler dans un état donné si une transition comprenant un canal urgent est possible.

– *Localités urgentes.* Un état qui a une localité urgente dans son vecteur de localités ne peut pas laisser passer le temps.

– *Localités atomiques.* Un état qui a une localité atomique dans son vecteur de localités (committed dans UPPAAL) ne peut pas laisser passer le temps et doit prendre une transition qui quite une localité atomique ou devient bloqué.

D'autres constructions syntaxiques sont définies par dessus pour une utilisation plus facile de l'outil. Ces extensions comprennent les tableaux d'entiers, d'horloges ou de canaux, les fonctions et types définis par l'utilisateur. Pour une référence plus complète une version mise-à-jour de [BEH 04b] est disponible sur *www.uppaal.com*.

UPPAAL implemente un algorithme d'exploration symbolique basé sur la sémantique symbolique des automates temporisés. Dans les algorithmes d'atteignabilité 1.2.4 ou de vivacité 1.2.5, UPPAAL calcules les états successeurs de façon symbolique comme suit : quand une transition est prise, le prochain état laisse passer le temps (si possible) et l'invariant de l'état est appliqué. Ceci donne tous les successeurs temporels pour une transition donnée. Dans les algorithmes qui suivent, le calcul des successeurs réfère à essayer toutes les actions possibles suivies par l'action de délai.

1.2.1.0.1. Exemple

Nous donnons un example de l'algorithme bien connu d'exclusion mutuelle de Fischer [ABA 92, KRI 96] avec deux processus par soucis de simplicité de l'exploration. Le protocole peut inclure un nombre quelconque de processus, chacun aillant son propre identificateur (dans l'exemple, seulement 1 et 2). La figure 1.1 montre le

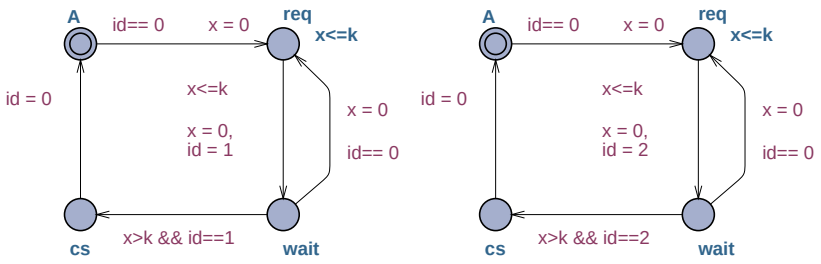

Figure 1.1. *Modèle d'automates temporisés pour le protocole d'exclusion mutuelle de Fischer.*

model du protocole. Les processus veulent eviter d'être dans leur section critique (`cs`)

en même temps. Le protocole utilise un identificateur partagé (`id`) pour choisir quel processus devrait accéder à la section critique et une horloge par processus pour les forcer à attendre au moins k unités de temps avant d'entrer `cs`. Les processus peuvent ré-essayer et revenir à `req` (requête). La figure 1.2 montre l'exploration symbolique

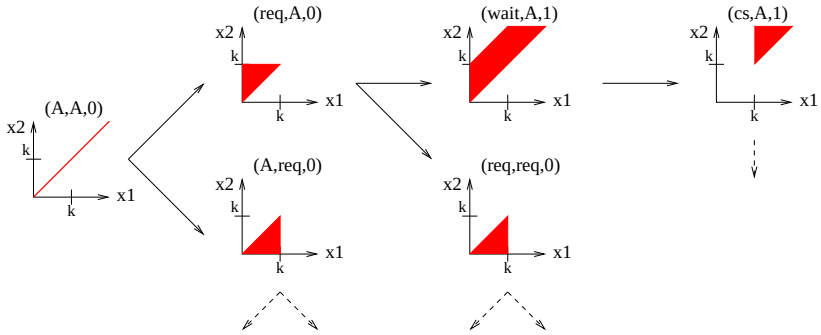

Figure 1.2. *Exploration symbolique du protocole d'exclusion mutuelle de Fischer.*

du modèle. La figure montre les localités et les variables entières comme un tuple et la zone graphiquement. Mettre à zéro une horloge correspond à projeter la zone sur l'axe qui correspond à cette horloge. Laisser passer le temps correspond à enlever les bornes supérieures de la zone (mais en gardant les contraintes diagonales). Appliquer une garde correspond à intersecter la zone avec celle donnée par la garde (ou encore en contraignant la zone par les contraintes de la garde). L'état initial (symbolique) laisse passer le temps depuis le point zéro à l'origine et les successeurs symboliques sont calculés à partir de là. Soit le premier ou le second processus se rend a `req`, mettant à zéro son horloge (projection) suivi par un délai borné par l'invariant à `req`. Depuis `(req,A,o)`, soit le premier processus continue ou le second essaye de nouveau de se rendre à `req`. Si c'est le premier processus alors nous avons de nouveau une mise-à-zéro suivie par un délai (non borné cette fois-ci). Depuis `(wait,A,q)` nous appliquons la contrainte de la garde aux zones et nous obtenons les états qui peuvent atteindre `(cs,A,1)`. L'exploration continue de l'à.

1.2.2. *Requêtes*

Les propriétés qui peuvent être vérifiées par UPPAAL, illustrées sur la figure 1.3, sont définies dans un sous-ensemble de TCTL est sont de la forme :

– $A[]\ \phi$ "pour tous les chemins et tous les états, ϕ",

– $E <>\ \phi$ "il existe un chemin où éventuellement, ϕ",

– $A <>\ \phi$ "pour tous les chemins, éventuellement ϕ",

– $E[]\ \phi$ "il existe un chemin où pour tous les états, ϕ", or

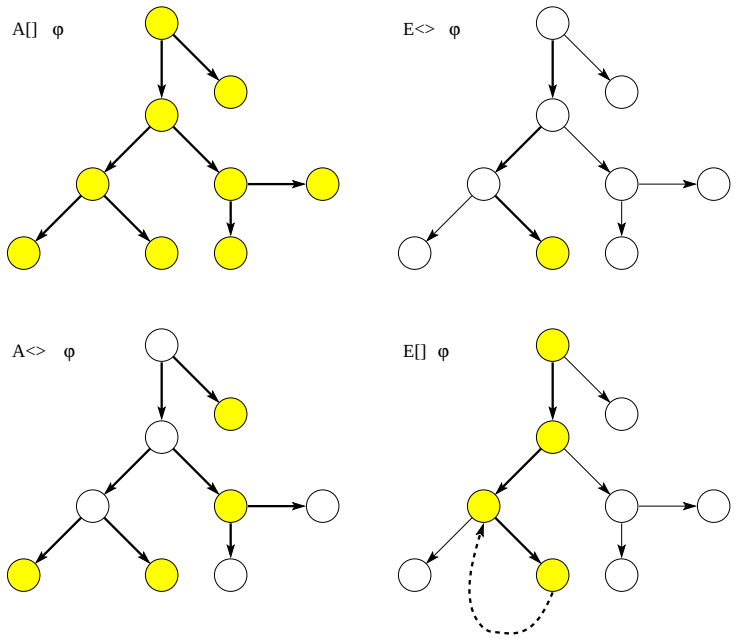

Figure 1.3. *Requêtes de base d'*UPPAAL.

– $\phi \ -- > \psi$ "réponse bornée : ϕ mène toujours à ψ", ce qui est équivalent à $A[](\phi \rightarrow A <> \ \psi)$

où ϕ et ψ sont des expressions booléennes sur les localités, variables entières et horloges. Ces requêtes sont définies sur les chemins : A s'applique pour tous les chemins et E sur un chemin. Les requêtes "[]" s'appliquent à tous les états sur les chemins et $<>$ pour un état sur les chemins. La figure 1.3 montre les traces d'états et les chemins pour lesquels les formules CTL sont satisfaites. Les états grisés sont ceux pour lesquels ϕ est satisfait. Les arcs en gras sont utilisés pour montrer les chemins sur lesquels les formules sont évaluées. La partie temporelle (**T**CTL) vient des contraintes d'horloges dans ϕ (et ψ).

Les formules $A[] \ \phi$ et $E <> \ \phi$ sont des propriétes d'atteignabilité et sont symmétriques : $A[] \ \phi = \neg E <> \ \neg\phi$. Les propriétes $A[] \ \phi$ sont aussi appelées propriétés de sûreté puisqu'elles verifient si une formule est satisfaite pour tous les états. Si une telle propriété n'est pas satisfaite, alors $E <> \ \neg\phi$ caractérise des chemins de contre-exemple. L'algorithme d'atteignabilité vérifie $E <> \ \phi$.

Les formules $A <> \ \phi$ et $E[] \ \phi$ sont des propriétés de vivacité et sont symmétriques aussi : $A <> \ \phi = \neg E[] \ \neg\phi$. Ces propriétés impliquent un algorithme de

détection de boucle puisque $E[]\ \phi$ est vrai pour un chemin infini sur lequel tous les états verifient ϕ. Comme l'espace des états symbolique est fini, l'algorithme cherche des boucles contraintes par ϕ. L'algorithme de vivacité vérifie $E[]\ \phi$.

1.2.3. *Architecture de l'Outil*

L'outil est séparé en deux composants : l'interface graphique utilisateur (GUI en anglais) – le client – et le moteur de model-checking – le serveur. Le GUI est écrit en Java est est déployé facilement sur différentes plateformes alors que le moteur ets recompilé pour chaque plateforme. Nous nous référons ici a l'architecture du moteur [DAV 03], la partie critique pour les performances de l'outil. Les strutures de données du moteur sont conçues autours d'une architecture centrée sur un flot de données qui forme un *pipeline*. Les données qui circulent à travers les composants de filtre sont des états ou des états ainsi qu'une transition.

Les différents composants, montrés sur la figure 1.4, sont les sources où les états sont crées (typiquement pour l'état initial), les drains où les états disparaissent (typiquement pour évaluer une expression), les tampons où les états sont poussés et enlevés et les filtres où les états sont poussés et expédiés aux composants suivants. De plus, une pompe montre où la boucle principale d'atteignabilité ou de vivacité est exécutée. Les bénéfices d'utiliser des composants de pipeline sont la flexibilité, la réutilisation

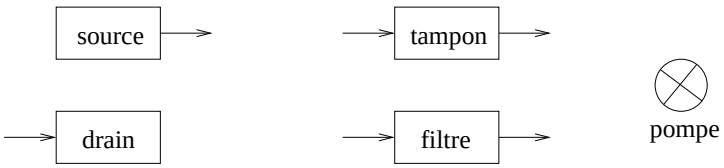

Figure 1.4. *Les filtres dans* UPPAAL.

de code et l'efficacité. La flexibilité vient de la possibilité d'échanger un composant pour un autre pour configurer le pipeline selong les options, typiquement pour utiliser différentes structures de stockage qui implémentent une exploration exacte, une sous-approximation ou une sur-approximation. Une telle configuration dynamique nous permet de sauter completement des étapes du pipeline si elles ne sont pas nécessaires comme par exemple les traces. La réutilisation de code vient de la réutilisation des composants à travers les différents pipelines d'atteignabilité ou de vivacité.

Les composants communs qui sont utilisés dans les pipelines d'atteignabilité 1.2.4 ou de vivacité 1.2.5 sont les suivants :

– *transition* qui calcule quelles transitions peuvent être prises à partir des états entrant,

– *successeur* qui prend les transitions,

– *délai* qui calcule le délai possible dans un état borné par son invariant,

– *extrapolation + réduction des horloges actives* qui applique l'extrapolation en fonction des constantes maximales des horloges du modèle en même temps que la réduction des horloges actives. En fait, donner localement $-\infty$ pour la constante maximale d'une horloge a pour effet de libérer les contraintes de cette horloge avec l'algorithme d'extrapolation.

1.2.4. *Pipeline d'Atteignabilité*

L'algorithme d'atteignabilité stocke ses états dans une structure appelée *PWList* [BEH 03b] qui unifie les queues traditionnelles pour les états en attente (waiting en anglais) et passés utilisées dans les model-checkers. Nous considérons les états symboliques sous la forme (l, Z) où l est le vecteur de localités et Z une zone. Par simplicité nous omettons les variables entières. L'algorithme traditionnel d'atteignabilité a la forme de celui de la figure 1.5. Un tel algorithm utilise deux structures principales, qui sont la liste *passed* des états déjà visités et la liste *waiting* pour stocker les états à explorer. L'algorithm démarre avec l'état initial où $I(l_0)$ réfère à son invariant. L'expression $\forall(l', Z') : (l, Z) \rightarrow (l', Z')$ réfère au calcul des successeursrefers to computing all successors (l', Z') de (l, Z). L'algorithme boucle sur tous les états dans la liste d'attente, teste si l'état recherché (*goal*) est atteint calcule les successeurs et pousse ceux qui sont nouveaux dans la liste d'attente. L'implémentation de cet algorithme utilise en pratique deux tables de hachage pour une recherche efficace des états, une par liste, et vérifier l'inclusion des états, c'est-à-dire si un état symbolique (la zone en fait) est inclu dans une des liste avant de l'y ajouter. Le second test d'inclusion est implicite dans l'algorithme quand un état est ajouté à la liste d'attente. Le test d'inclusion est crucial pour améliorer les performances car il évite des explorations redondantes mais il coûte $O(n^2)$ où n est le nombre d'horloges.

Contrairement à ces deux structures, une structure unifiée contient tous les états où certains d'entre eux sont colorés en attente et d'autres passés (l'implementation utilise un ensemble d'états colorés). Nous notons par (P, W) la structure unifiée où P représente tous les états (considéré passés) et W marque le sous-ensemble qui sont en attente. Une "PWList" est décrite par une paire $(P, W) \in 2^S \times 2^S$, où S est l'ensemble des états symboliques, $W \subseteq P$ et les deux fonctions $put : 2^S \times 2^S \times S \rightarrow 2^S \times 2^S$ et $get : 2^S \times 2^S \rightarrow 2^S \times 2^S \times S$, telles que :

– $get(P, W) = (P, W \setminus \{(l, Z)\}, (l, Z))$ pour un $(l, Z) \in W$.

– $put(P, W, (l, Z)) =$

$$\begin{cases} (P \setminus I, W \cup \{(l, Z)\}) & \text{if } \forall (l, Y) \in P : Z \not\subseteq Y \\ (P, W) & \text{otherwise,} \end{cases}$$

où $I = \{(l, Y) \in P \,\big|\, Y \subset Z\}$.

```
waiting = {(l0, Z0 ∧ I(l0))}
passed = ∅
while waiting ≠ ∅ do
  (l, Z) = select state from waiting
  waiting = waiting \ {(l, Z)}
  if goal(l, Z) then return true
  if ∀(l, Y) ∈ passed : Z ⊈ Y then
    passed = passed ∪ {(l, Z)}
    ∀(l', Z') : (l, Z) → (l', Z') do
      if ∀(l', Y') ∈ waiting : Z' ⊈ Y' then
        waiting = waiting ∪ {(l', Z')}
      endif
    done
  endif
done
return false
```

Figure 1.5. *Algorithme d'atteignabilité traditionnel.*

La fonction get enlève les états de W et les laisse dans P. La fonction put enlève les états de P qui sont inclus (l'ensemble I) dans le nouvel état qui est ajouté a W. Enlever les états de P les enlève aussi implicitement de W puisque $W \subseteq P$. Similairement, les états ajoutés à W le sont aussi à P.

La figure 1.6 montre l'algorithme simplifié qui utilise la structure PWList. A présent il n'y a plus de redondance dans l'ensemble des états et les états ne sont plus déplacés mais seulement re-coloriés. De plus, l'algorithme nécessite seulement d'une seule table de hachage et nous n'avons qu'un test d'inclusion. En pratique nous utilisons une liste de références pour accéder au sous-ensemble W.

```
(P, W) = {(l0, Z0 ∧ I(l0)), (l0, Z0 ∧ I(l0))}
while W ≠ ∅ do
  (P, W, (l, Z)) = get(P, W)
  if goal(l, Z) then return true
  ∀(l', Z') : (l, Z) → (l', Z') do
    (P, W) = put(P, W, (l', Z'))    done
done
return false
```

Figure 1.6. *Algorithme d'atteignabilité d'*UPPAAL.

Le pipeline d'atteignabilité d'UPPAAL est basé sur l'algorithme de la figure 1.6 avec la structure *PWList* à son centre. Le pipeline est montré dans la figure 1.7. Il calcule les successeurs symboliques, ce qui est séparé en le calcul des transitions et l'action de les prendre, suivi par le délai et l'extrapolation. Les expressions sont évaluées a la fin du pipeline. Le lecteur attentif note que l'évaluation de l'état but n'est pas à la même place dans l'algorithme. L'évaluation est ici directement après le calcul des successeurs pour éviter de parcourir la liste d'attente avant de se rendre compte que l'état est en fait déjà trouvé, ce qui permet de terminer plus tôt. L'état initial (qui correspond au point zéro à l'origine) est inséré dans le pipeline au niveau du délai et est traité comme successeur.

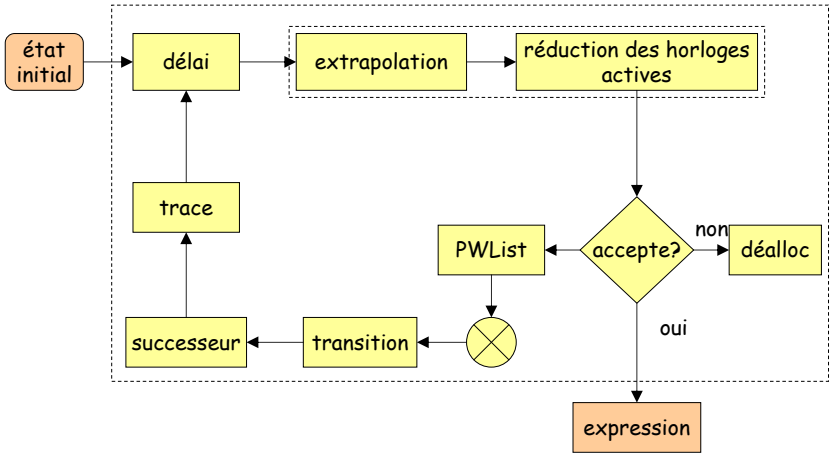

Figure 1.7. *Le pipeline d'atteignabilité d'*UPPAAL.

1.2.5. *Pipeline de Vivacité*

L'algorithme de vivacité est donné dans la figure 1.8. L'algorithme possède un ensemble d'états passés P qui verifient $A <> \ \phi$ et une pile Stk d'états qui verifient $\neg\phi$. Les autres états ne sont pas encore explorés. L'opération de délai est speciale ici et est restrainte aux états qui satisfont $\neg\phi$. Le but de l'algorithme est de trouver une boucle d'états qui verifient $\neg\phi$. Une telle boucle serait un contre-exemple à $A <> \ \phi$. Si l'algorithme trouve un état pour lequel il y a un délai non borné où un blocage alors

c'est aussi un contre-exemple. Le lecteur note que l'appel récursif est fait avec l'action α appropriée en pratique pour garder la trace du chemin courant.

proc Eventually(S_0, ϕ)
  $Stk = \varnothing$
  $P = \varnothing$
  Search($delay(S_0, \neg\phi)$)
  exit(true)
end

proc Search(S)
  if $loop(S, Stk)$ **then**
    exit($false$)
  fi
  $S = S \wedge \neg\phi$
  $push(Stk, S)$
  if $unbounded(S) \vee deadlock(S)$ **then**
    exit($false$)
  fi
  if $\forall S' \in P : S \not\subseteq S'$ **then**
    foreach $S' : S \stackrel{\alpha}{\Rightarrow} S'$ **do**
      Search($delay(S', \neg\phi)$)
    od
  fi
  $P = P \cup \{pop(Stk)\}$
end

Figure 1.8. *L'algorithme de vivacité d'*UPPAAL.

Le pipeline de vivacité basé sur l'algorithme de la figure 1.8 est donné dans la figure 1.9. Ici, l'appel récursif est déplié avec l'aide de la liste des états en attente qui garde les transitions qui doivent être prises (et non simplement les états comme précédemment). Comme l'algorithme le montre, quand la fonction *Search* retourne, les états sont déplacés de la pile Stk vers l'ensemble des états passés P. La boucle principale (la pompe) explore et déplace ces états. Les transitions des successeurs sont poussées vers la liste d'attente. Les états source sont poussés sur la pile s'ils ne sont pas explorés et ils sont aussi testés pour délai non borné ou blocage. Si un état sort du pipeline c'est l'entrée vers un chemin infini, contre-exemple de $A <>\ \phi$.

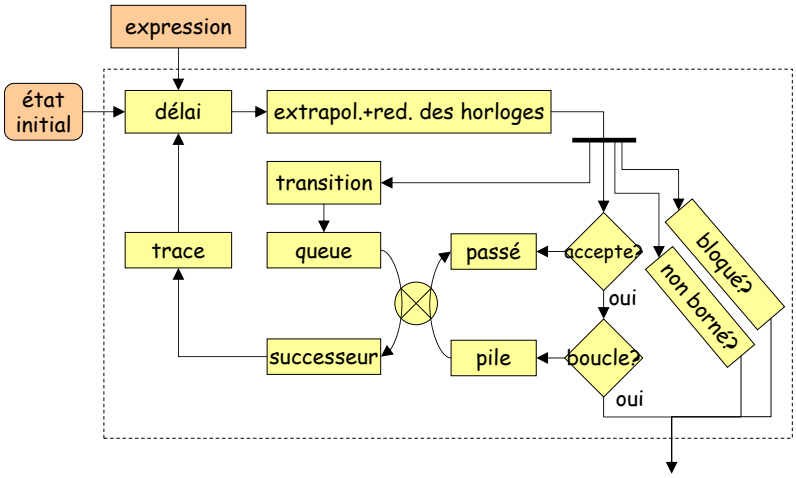

Figure 1.9. *Le pipeline de vivacité d'*UPPAAL.

1.2.6. *Pipeline de Réponse Bornée*

Le pipeline de réponse bornée vérifie les propriétés $A[](\phi \rightarrow A <> \ \psi)$. L'algorithme ici est de lancer une vérification de vivacité pour tous les états qui satisfont ϕ. Ceci est implémenté comme une composition des deux pipelines précédants comme illustré dans la figure 1.10. Les pipelines d'atteignabilité et de vivacité peuvent être utilisés comme composants de filtre eux-même. Les états satisfaisant ϕ sont poussés vers le pipeline de vivacité qui reçoit l'expression ψ. Un état qui sort du pipeline de vivacité est une entrée vers un chemin infini, contre-exemple de $A <>; \psi$.

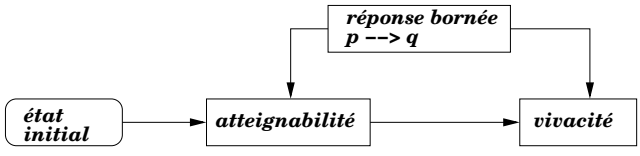

Figure 1.10. *Le pipeline de réponse bornée d'*UPPAAL.

1.2.7. *Active Clock Reduction*

La réduction des horloges actives est une technique qui enlève les horloges dont les valeurs ne sont pas pertinentes dans certains états. Si une horloge est mise-à-zéro avant d'être testée plus tard alors sa valeur n'est pas pertinente avant la mise-à-zéro.

Définition 1.2.5 (Horloge Inactive) *Une horloge x est dite* inactive *dans un état S si sur tous les chemins à partir de S, x est toujours mise-à-zéro avant dêtre testée.*

En pratique, la réduction est faite en deux étapes : i) une analyse statique est faite pour chaque processus, ce qui donne l'ensemble des horloges actives pour chaque localité et ii) pendant la vérification l'ensemble des horloges actives des *états* est obtenue en combinant les horloges actives des localités actives de l'état en question. Seulement les contraintes des horloges actives sont gardées dans les zones.

1.2.8. *Techniques de Réduction d'Espace*

1.2.8.0.2. Éviter de Stocker Tous les États

Une des techniques de réduction accessible par l'option `-S1` du model-checker est d'éviter de stocker tous les états. En fait, pour s'assurer de la fin de l'algorithme nous n'avons besoin que des états qui contiennent des entrées de boucles. Cette option ne stocke que ces états dans la liste des états passés. Une autre heuristique plus agressive (option `-S2`) a été developpée pour stocker encore moins d'états tout en garantissant la fin de l'algorithme [BEH 03a].

1.2.8.0.3. Partager les Données

Les états sont des tuples (L, V, Z) où L est un vecteur de localité, V un vecteur de variables entières et Z une zone (DBM en pratique). Bien que chaque état soit stockés en un exemplaire unique (s'il n'est pas inclus dans un état plus large par rapport à sa zone), il apparaît que les composantes individuelles de l'état L, V et Z sont répétées parmis tous les états. La raison est la suivante : quand le système change d'état, bien souvent il garde une des composantes (localité, variable ou horloge). Nous pouvons ainsi partager ces composantes individuelles parmis tous les états. En pratique, cela donne typiquement 80% de réduction en mémoire [BEH 03b].

1.2.8.0.4. Graphe Minimal

Nous devons resserrer les contraintes des DBMs pour pouvoir tester l'inclusion (en $O(n^2)$ avec n étant le nombre d'horloges). Le test d'inclusion est fait en comparant les contraintes c_{ij} et c'_{ij} par paires. Le resserrement de contraintes est obtenu par un algorithme de chemins les plus courts, typiquement celui de Floyd's [FLO 62].

Ce resserrement resulte en une représentation unique pour une zone donnée, la forme canonique des DBMs. Ceci est utile pour stocker les DBMs de façon unique mais nous avons besoin de $O(n^2)$ en espace. La figure [**?**] montre ce que l'algorithme des chemins les courts calcule (la fermeture par les plus courts chemins de (a) à (b)) si nous regardons une DBM comme un graphe avec les horloges aux sommets et les contraintes $x_i - x_j \leq c_i j$ sur les arcs allant de x_j à x_i qui représentent des distances. Dans [LAR 97b] une réduction a été proposée qui réduit le nombre de contraintes nécessaire au minimum afin de représenter une zone donnée. En appliquand cet algorithme (réduction des chemins les plus courts de (b) à (c)) résulte en le graphe minimal, minimal par rapport au nombre d'arcs. L'algorithme coûte $O(n^3)$ en temps. Bien que nous ayons toujours $O(n^2)$ arcs dans le pire des cas, nous obtenons en pratique $O(n)$ en moyenne. UPPAAL stocke ce nombre réduit d'arcs et peut restorer la DBM complète, c'est-à-dire le graphe complet, en utilisant l'algorithme des chemin les plus courts (et on retrouve (b)).

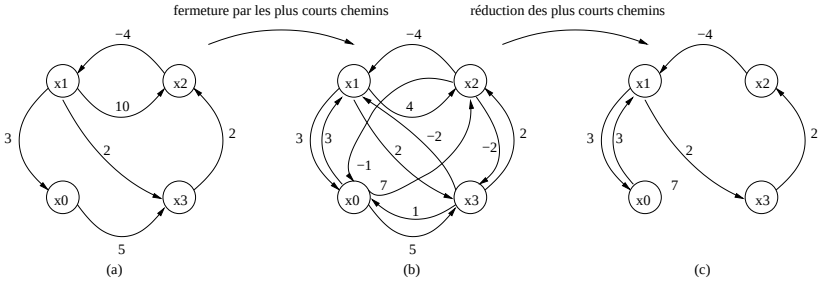

Figure 1.11. *Fermeture et réduction des chemins les plus courts.*

L'algorithme de réduction des chemins les plus courts marche en deux étapes principales : i) il calcule les classes d'équivalence des horloges, ce qui est accompli en detectant les cycles nuls et ii) il choisit un représentant par classe d'équivalence et enlève les arcs redondants entre eux.

Nous notons aussi que le calcul des chemins les plus courts coûte aussi $O(n^3)$ en temps donc il est important pour les performances d'éviter de le faire autant que possible. Il apparaît en fait que la plupart des opérations sur les DBMs (délai, intersection, etc...) peuvent être effectuées sur une DBM canonique de telle sorte qu'elles préservent la forme canonique sans coût supplémentaire. Nous notons aussi que nous ne connaissons aucun moyen efficace d'appliquer ces opérations directement sur la forme du graphe minimal. Cependant, si les contraintes d'une DBM (canonique) sont inférieures ou égales aux contraintes présentes dans un graphe minimal alors nous pouvons quand même déduire l'inclusion de la DBM dans ce sens.

1.2.8.0.5. Réduction par Symmétrie

UPPAAL implemente l'algorithme présenté dans [HEN 03]. Cet algorithme trouve les classes d'équivalence des états par rapport aux symmétries (aussi appelées orbites) et choisit des états représentants en triant les états (dans la même classe d'équivalence). L'algorithme est implementé à l'aide du type *scalar* qui définit un ensemble (d'une taille donnée) de scalaires non-ordonnés. La figure 1.12 montre les gains de performance sur le modèle du protocole de Fischer.

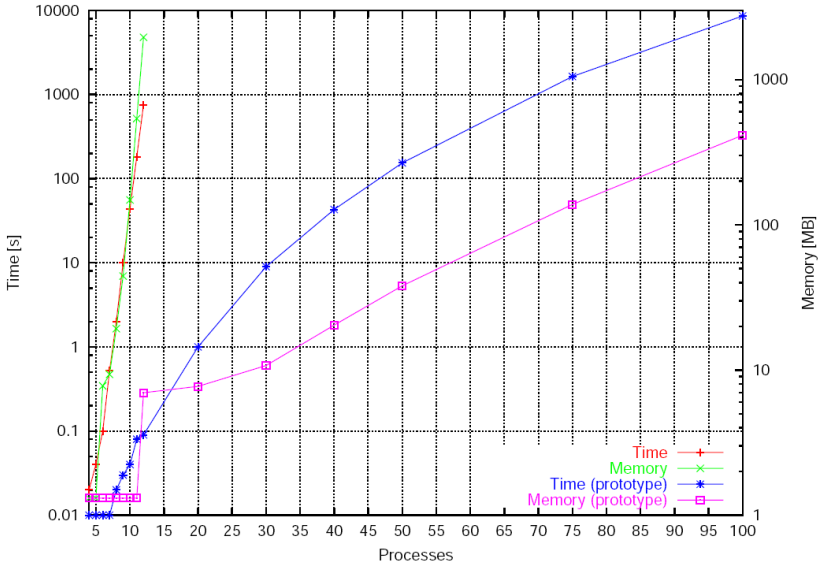

Figure 1.12. *Résultats expérimentaux pour la réduction par symmétrie sur le protocol de Fischer.*

1.2.9. *Techniques d'Approximation*

Quelques fois les systèmes sont trop complexes pour être vérifiés exactement. Pour ces cas là il est utile d'utiliser des techniques approximatives.

1.2.9.0.6. Sur-Approximation : Enveloppe Convexe

UPPAAL implemente la technique de sur-approximation de l'enveloppe convexe [BAL 96] qui consiste à calculer l'enveloppe convex de zones et de la stocker au lieu de garder toutes les zones. La figure 1.13 illustre l'exemple suivant : dans un automate il y a deux chemins qui mènent à l'état $S3$, ce qui donne deux zones différentes qui sont montrées graphiquement. La technique stocke l'union convexe de ces zones (en

fait la plus petite zone qui les contient, d'où le nom d'enveloppe). Même si la technique rajoute des états supplémentaires (non atteints par le modle original), elle est utile en particulier pour les propriétés de sûreté.

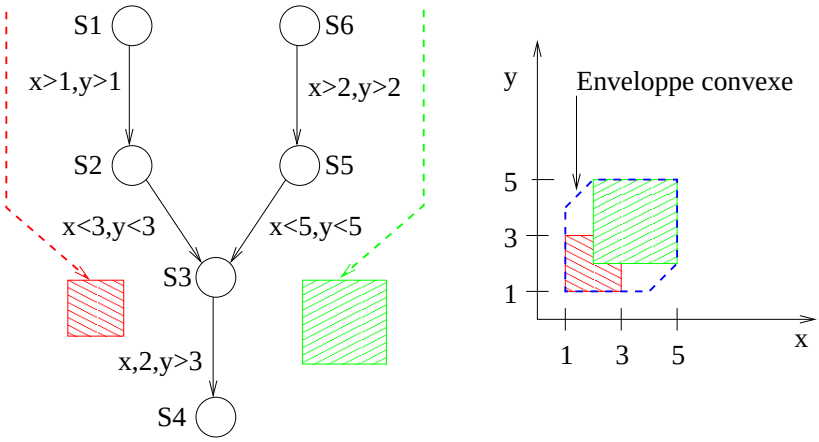

Figure 1.13. *Exemple d'enveloppe convexe.*

1.2.9.0.7. Sous-Approximation : Hachage en Bit des États

UPPAAL implémente la technique de sous-approximation du hachage en bit des états [HOL 91, HOL 98]. Cette technique consiste à stocker seulement un bit par état au lieu de l'état entier (vecteur de localité, variables entières et zone). Ceci est fait en allouant une grande table de hachage de taille N (bits) initialement mise-à-zéro et de mettre à un le bit $hash(tat)\%N$ quand un état est visité (fonction de hachage appliquée à un état modulo N). Il y aura bien sûr des collisions qui peuvent conclure que l'état est visité bien qu'il ne le soit pas et un tel état ne sera pas exploré. La technique est utile en particulier pour les propriété d'atteignabilité.

1.2.10. *Extensions*

1.2.10.0.8. Atteignabilité Robuste

Traditionellement les vérifications sont faîtes en considérant toutes les horloges parfaitement synchrones mais en pratique ce n'est pas le cas et les horloges dérivent légèrement avec le temps. Des algorithmes spécialisés sont alors nécessaires pour prendre en compte cette dérive et faire une analyse dite *robuste*. Deux algorithmes d'atteignabilité robuste sont actuellement disponibles dans UPPAAL avec deux sémantiques légèrement différentes : le premier [DAW 06] est utilisé pour les propriétès de type $E <> * \phi$ et le second [SWA 07] pour les propriétés de type $E <> + \phi$. Cette extensions est displonible dans la version de développement.

1.2.10.0.9. Fusion de DBMs

La technique de l'enveloppe convexe est une technique de sur-approximation qui réduit le nombre de zones. UPPAAL offre une autre technique *exacte* [DAV 05] pour fusioner les DBMs à la volée. Cette technique remplace deux (ou plusieurs) DBMs par son union convexe si cette union est exacte, c'est-à-dire si aucun état supplémentaire n'est rajouté. Les états sont fusionés dans la liste des états passés et aussi dans la liste des états en attente. Nous notons que même si nous utilisons une structure unifiée pour représenter ces deux listes, nous ne les mélangeons pas pour la fusion pour ne pas avoir des explorations redondantes. Cette option est active par défaut dans la version de développement.

1.2.10.0.10. Chronomètres

L'analyse d'atteignabilité des automates temporisés étendus avec les chronomètres est indécidable mais il y a une technique efficace de sur-approximation pour vérifier de tels automates [CAS 00]. La technique consiste à modifier l'opérateur de délai des DBMs de telle sorte que les horloges qui sont arrêtées gardent leurs bornes supérieures. Syntactiquement, l'utilisateur ajoute à l'invariant d'un état une expression du type $x' == expr$ (en conjonction avec l'invariant) où $expr$ est évaluée à 0 ou 1. Cette technique s'est avérée utile pour modéliser les problèmes d'ordonnancement puisque nous voulons vérifier des propriétés de sûreté, c'est-à-dire que les échéances ne sont jamais manquées. Cette extension est disponible dans la version de développement.

1.2.10.0.11. Autres Extensions

UPPAAL implémente la méthode généralisée de "sweep-line" [KRI 02]. L'utilisateur doit définir des mesures de progrès pour profiter de cette technique. Plusieurs algorithmes d'extrapolations [BEH 04a] ont été implementés. Ils permettent de profiter des bornes maximales supérieures mais aussi des bornes maximales inférieures. Une version distribuée d'UPPAAL a été développée [BEH 00] qui s'exécute sur des grilles de calculs homogènes. D'un point de vue du modèle, une technique d'accélération de cycles a été développée qui améliore l'analyse d'atteignabilité sur des modèles qui contiennent des cycles [HEN 02].

1.3. UPPAAL-CORA

Quand UPPAAL calcule des traces qui répondent à des propriétés d'atteignabilité, il est possible d'utiliser aussi un algorithme qui donne le chemin le plus rapide (par rapport au temps et non le nombre de transitions). Cette option peut être exploitée pour résoudre un nombre de problèmes généraux d'ordonnancement tels que le problème du voyageur de commerce.

UPPAAL-CORA est une extension d'UPPAAL qui offre une analyse d'atteignabilité avec coûts minimaux pour des modèles d'automates temporisés étendus avec

des coûts. Ces modèles, appelés automates temporisés à coûts, ont été proposés indépendemment et leur problème d'atteignabilité prouvé décidable dans [BEH 01b] et [ALU 01]. UPPAAL-CORA a été utilisé avec succès dans des études de cas tels que l'ordonnancement de laque [BEH 05a] et l'attérissage d'avions [BEH 05b].

1.3.1. *Automates Temporisés à Coûts*

Un automate temporisé à coûts est défini de façon similaire à un automate temporisé (définition **??**) mais avec en rajoutant une fonction de coût $\mathsf{Cost} : (L \cup E) \rightarrow \mathbb{N}$ qui assigne une valeur entière positive ou nulle aux localitès et arcs. Dans le cas de réseaux d'automates, la sémantique est la suivante : le coût d'un état augmente avec le délai en fonction de la somme des taux des localités actives dans cet état (par unité de temps) et prendre une transition augmente le coût par les coûts des arcs concernés. Autrement dit, le coût peut augmenter de faoṇ continue par le délai ou discrètement par une transition. Les coûts augmentent monotoniquement.

Pour analyser efficacement les automates temporisés à coûts, UPPAAL-CORA utilise la notion de zones à coûts que nous notons $\mathcal{Z}$ [LAR 01]. Les zones à coûts sont des abstractions de valeurs d'horloges similairement aux zones mais dotées d'une fonction affine de coûts appliquée à la zone. La fonction de coût dans UPPAAL-CORA est implémentée par une combinaison linéaire de coefficients entiers associés aux horloges de la zone. L'utilisation des zones à coûts complique le calcul des successeurs discrets et par délai des états symboliques représentés avec les zones à coûts puisque ces opérations nécessitent de couper les zones en un ensemble de zones plus petites et disjointes afin de maintenir la propriété d'affinité des fonctions de coûts. Ceci augmente de façcon significative le nombre détats symboliques à explorer. Pour une description plus complète des algorithmes de calcul des successeurs avec les zones à coût nous renvoyons le lecteur à **??**.

La figure 1.14 montre l'algorithme d'atteignabilité de coût minimal d'UPPAAL-CORA qui calcule le coût minimal pour satisfaire une propriété d'atteignabilité ou ∞ si la propriété ne peut pas être satisfaite. L'algorithme est une variante de l'algorithm classique de séparation et évaluation (branch-and-bound) adapté à UPPAAL. L'algorithme maintient une variable de coût qui garde la meilleure solution trouvée jusqu'à présent. La valeur de la valeur de coût est mise-à-jour chaque fois qu'une solution meilleure est trouvée. L'algorithme est limité en évitant d'explorer les successeurs d'un état qui ne peuvent pas améliorer la meilleure solution. UPPAAL-CORA laisse l'utilisateur définir une estimation de la borne inférieure du coût restant pour atteindre l'état recherche à partir de l'état courant, ce qui permet d'experimenter diverses heuristiques pour l'exploration. Si le coût restant est mal utilisé, par exemple en donnant une borne inférieure invalide, l'algorithme n'est plus garanti d'être correct.

Le pipeline de UPPAAL-CORA est semblable à celui d'UPPAAL pour l'algorithme d'atteignabilité de la figure 1.7. La différence est que l'algorithme ne se termine pas

$(P, W) = \{(l_0, \mathcal{Z}_0 \wedge I(l_0)), (l_0, \mathcal{Z}_0 \wedge I(l_0))\}$
$cost = \infty$
while $W \neq \varnothing$ **do**
  $(P, W, (l, \mathcal{Z})) =$ get(P, W)
  if $goal(l, \mathcal{Z})$ **and** $mincost(\mathcal{Z}) < cost$ **then**
    $cost = mincost(\mathcal{Z})$
    continue
  endif
  if $mincost(\mathcal{Z}) + remain(l, \mathcal{Z}) < cost$
    $\forall (l', \mathcal{Z}') : (l, \mathcal{Z}) \rightarrow (l', \mathcal{Z}')$ **do**
      $(P, W) = put(P, W, (l', \mathcal{Z}'))$
    done
  endif
done
return $cost$

Figure 1.14. *Algorithme d'atteignabilité de coût minimal par séparation et évaluation (branch-and-bound).*

après avoir trouvé une solution au problème d'atteignabilité mais il continue à affiner la solution jusqu'à ce qu'il n'y ait plus d'états à explorer. L'algorithme rapporte alors la valeur de la variable de coût comme solution au problème d'atteignabilité. De plus, il n'y a pas d'extrapolation pour UPPAAL-CORA. Le lecteur se demande alors comment l'algorithme se termine. En fait, la terminaison découle des deux faits suivants. Tout d'abord tout automate temporisé peut être converti en un automate temporisé borné avec des invariants de borne supérieure sur chaque horloge. Cela donne un nombre fini de zones. Ensuite, étant donné que les affectations de coûts d'un automate temporisé à coûts sont entières, nous savons que les fonctions de coûts sur des zones bornées forment un bon pré-ordre, ce qui veut dire que pour une zone donnée, il n'existe pas une séquence infinie de fonctions de coûts sans qu'une ne soit éventuellement inclue dans une autre précédante de la séquence [LAR 01]. Ces faits combinés garantissent la terminaison de l'algorithme.

L'algorithme de coût minimal utilise la structure de donnée PW-List mentionnée précédemment. Cependant, pour l'utiliser correctement, le test d'inclusion doit être modifié pour prendre en compte l'information de coût des zones à coûts. Cela ne suffit pas pour une zone à coût $\mathcal{Z}$ d'en include une autre $\mathcal{Z}'$. En effet, la fonction de coût de $\mathcal{Z}$ doit aussi être inférieure à celle de $\mathcal{Z}'$. Dans UPPAAL-CORA, ce test est implémenté en résolvant le programme linéaire de la minimisation des différences entre les fonctions de coûts de $\mathcal{Z}'$ et de $\mathcal{Z}$ sur la zone de $\mathcal{Z}'$. Si la solution est positive nous déduisons que $\mathcal{Z}$ inclue $\mathcal{Z}'$. De plus, la séparation des états qui ont un coût supérieur à la meilleure solution courante (en prenant en compte l'estimation de coût restant) est aussi implémentée dans l'insertion des états dans la PW-List.

Un des aspects clés de l'algorithme de UPPAAL-CORA est la résolution du programme linéaire qui découle du test d'inclusion des zones à coûts et aussi le calcul des coûts minimaux des zones à coûts. Puisque que les zones possèdent la propriété qu'elles peuvent être décrites uniquement par des contraintes de différences d'horloges, les programmes linéaires peuvent utiliser cette structure. En fait, le problème de minimisation est le problème dual du problème de flux de coût minimal qui est connu pour avoir des algorithmes plus efficaces que les problèmes généraux de programmation linéaires [AHU 93, RAS 06]. Par conséquence, UPPAAL-CORA convertit chaque instance de problème de minimisation de zone à coût en son problème dual de flux de coût minimal et résoud ce problème dual. Cette approche améliore de façon conséquente le temps d'éxécution.

1.3.2. *Exemple*

La figure 1.15 montre un exemple sur comment les taux de coût des localités et les coûts des arcs peuvent être utilisés dans UPPAAL-CORA. Nous notons que cost est une variable spéciale réservée dans UPPAAL-CORA est ne doit pas être déclarée. Le coût augmente de façon continue avec un taux de 1 dans A et de 2 dans B. De plus prendre la transition vers C l'augmente de 3 d'un coup. L'automate pourrait prendre la transition vers B immédiatement mais alors il faudrait attendre dans B ou le taux est plus élevé. De plus il faut attendre au moins une unité de temps dans B à cause de y. Le lecteur peut vérifier que le coût minimal pour atteindre C est 6, en attendant une unité de temps dans A et B.

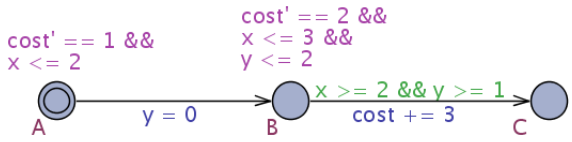

Figure 1.15. *Example de modle d'*UPPAAL-CORA.

1.4. UPPAAL-TIGA

1.4.1. *Automates de Jeux Temporisés*

UPPAAL-TIGA implémente le premier algorithme vraiment à la volée pour résoudre les jeux temporisés [CAS 05]. L'algorithme est une extension de [LIU 98] avec du temps. Notre modèle est spécifié en tant que réseau d'automates de jeux temporisés [MAL 95] (TGA – timed game automata en anglais) où les arcs sont marqués comme étant soit contrôlables soit incontrôlables (voir figure 1.16). Le modèle définit

un jeu à deux joueurs avec d'un côté le *contrôleur* et d'un autre l'*environement*. Les conditions pour gagner le jeu sont spécifiées en tant que formules TCTL. L'outil est conçu pour générer des stratégies pour qu'un contrôleur atteigne un objectif donné ou qu'il maintienne un invariant quelques que soient les actions de l'environement qui joue comme adversaire.

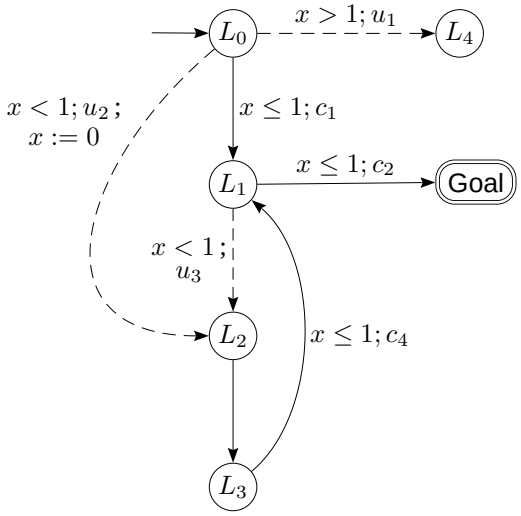

Figure 1.16. *Un exemple d'un automate de jeu temporié.*

Considérons l'exemple de la figure 1.16. Nous avons une horloge x et deux types d'arcs : les contrôlables (c_i) et les incontrôlables (u_i). Le jeu d'atteignabilité consiste à trouver une stratégie pour le contrôleur qui doit atteindre l'état Goal, quelque soient les transitions incontrôlables (u_i) que l'adversaire prend. Pour un état initial de la forme (l_0, x) avec $x \leq 1$, il y a une telle stratégie qui consiste à :

– prendre c_1 immédiatement dans tous les états (l_0, x) avec $x \leq 1$;

– prendre c_2 immédiatement dans tous les états (l_1, x) avec $x \leq 2$;

– prendre c_3 immédiatement dans tous les états (l_2, x) ;

– et attendre dans tous les états (l_3, x) tant que $x < 1$ jusqu'à ce que la valeur de x atteigne 1 où l'arc c_4 est pris.

Définition 1.4.1 (Réseau de d'Automates de Jeux Temporisés (NTGA)) *Un NTGA (network of timed game automata en anglais) est un NTA G avec l'ensemble des transitions E_i de chaque automate $\mathcal{A}_\rangle$ partitioné en actions* contrôlables E_i^c *et* incontrôlables E_i^u. *Nous écrivons $E^c \stackrel{def}{=} \bigcup_{i\in\{1,\ldots,n\}} E_i^c$ et $E^u \stackrel{def}{=} \bigcup_{i\in\{1,\ldots,n\}} E_i^u$. De plus, les*

invariants sont restreints à $Inv_i : L_i \rightarrow \mathcal{C}'(X_i)$ *où* $\mathcal{C}'$ *est le sous-ensemble de* $\mathcal{C}$ *en utilisant les contraintes de la forme* $x \leq k$.

Soit un NTGA G et une propriété de contôle. Le probème de contôle d'*atteignabilité (resp. de sûreté* consiste à trouver une *stratégie* f pour le contrôleur telle que toutes les exécutions de G supervisées par f satisfont la formule. Les différentes propriétés de contrôle disponible dans UPPAAL-TIGA sont les suivantes :

– *control :* $A[\ \phi\ \mathcal{U}\ \psi\ ]$, c.a.d. atteidre ψ tout en évitant $\neg\phi$,

– *control :* $A <>\ \psi$, c.a.d. atteindre ψ, racourci pour $A[\ true\ \mathcal{U}\ \psi\ ]$,

– *control :* $A[\ \phi\ \mathcal{W}\ \psi\ ]$, c.a.d. *peut-être* atteindre ψ tout en évitant $\neg\phi$,

– *control :* $A[]\ \phi$, c.a.d. éviter $\neg\phi$, racourci pour $A[\ \phi\ \mathcal{W}$ *false* $]$.

La définition formelle des problèmes de contrôle sont basées sur les définitions de *stratégies* et d'*issue*. Dans une situation quelconque, les stratégies suggèrent une action particulière après un certain délai. Une stratégie [MAL 95] est décrite comme une fonction qui durant le jeu donne constamment l'information de que les joueurs veulent faire sous la forme d'une paire $(e, \delta) \in (E \times \mathbb{R}_{\geq 0}) \cup \{(\perp, \infty)\}$. $(\perp, \infty)$ signifie que la stratégie veut attendre indéfiniment.

L'environement a priorité quand il choisit ses actions. De plus, il peut décider de ne pas prendre d'action, sauf si un invariant requiert que l'état courant doit être quité et que le contrôleur ne peut pas le faire (dans ce cas seul l'environement peut le faire). Les actions incontrôlables peuvent ẗre forcées seulement dans les états q où un invariant requiert de prendre une action et qu'aucune transition contrôlable n'est possible et il y a une transition incontrôlable possible (à partir de la localité impliquant cet invariant). Pour plus de détails sur les différents cas où des actions "forcées" sont possibles, nous renvoyons le lecteur au manuel d'UPPAAL-TIGA disponible sur *http ://www.cs.aau.dk/˜adavid/tiga/*. La sémantique implémentée est légèrement différente que celle décrite dans [CAS 05] où le jeu ne peut être gagné que par des transitions contrôlables, ce qui impliquait qu'il n'y avait pas d'actions "forcées".

1.4.2. *Pipeline d'Atteignabilité*

Nous adaptons l'algorithme d'atteignabilité de [CAS 05] basé sur les transitions comme montré dans la figure 1.17 vers un algorithme basé maintenant sur les états pour le pipeline d'atteignabilité d'UPPAAL-TIGA comme montré dans la figure 1.18. En observant attentivement l'algorithme, il apparaît que nous n'avons besoin que de l'état de destination S' pour explorer en avant et de l'état source S pour explorer en arrière. Par conséquence, la queue d'attente dans le pipeline contient les états et leur direction d'exploration. Le graphe des états stocke en plus les sous-ensembles gagnants (et perdants). Bien que l'algorithme ne le montre pas, nous pouvons garder

Initialisation :

$Passed \leftarrow \{S_0\}$;
$Waiting \leftarrow \{(S_0, \alpha, S') \mid S' = Post_\alpha(S_0)^\nearrow\}$;
$Win[S_0] \leftarrow \emptyset$;
$Depend[S_0] \leftarrow \emptyset$;

Boucle principale :

while $((Waiting \neq \emptyset) \wedge (s_0 \notin Win[S_0]))$ **do**
  $e = (S, \alpha, S') \leftarrow pop(Waiting)$;
  if $S' \notin Passed$ **then**
    $Passed \leftarrow Passed \cup \{S'\}$;
    $Depend[S'] \leftarrow \{(S, \alpha, S')\}$;
    $Win[S'] \leftarrow S' \cap G$;
    $Waiting \leftarrow Waiting \cup \{(S', \alpha, S'') \mid S'' = Post_\alpha(S')^\nearrow\}$;
    if $Win[S'] \neq \emptyset$ **then** $Waiting \leftarrow Waiting \cup \{e\}$;
  else (* reevaluate *)
    $Win^* \leftarrow Pred_t(Win[S] \cup \bigcup_{S \xrightarrow{c} T} Pred_c(Win[T]), \bigcup_{S \xrightarrow{u} T} Pred_u(T \setminus Win[T])) \cap S$;
    if $(Win[S] \subsetneq Win^*)$ **then**
      $Waiting \leftarrow Waiting \cup Depend[S]$; $Win[S] \leftarrow Win^*$;
    if $Win[S'] \subsetneq S'$ **then** $Depend[S'] \leftarrow Depend[S'] \cup \{e\}$;
  endif
endwhile

Figure 1.17. *Algorithme symbolique à la volée pour jeux d'atteignabilité temporisés.*

les états perdants aussi puisque le problème est dual. La partie supérieure du pipeline calcule les successeurs de façcon similaire au pipeline d'atteignabilité d'UPPAAL. La partie inférieure propage en arrière l'information (états gagnants ou perdants).

L'implémentation de la figure 1.18 marche comme suit : quand un état s est retiré de la queue et qu'il doit être exploré en avant, les successeurs s' sont calculés et testés par rapport au graphe des états. Des successeurs supplémentaires seront calculés (s', F) si s' n'est pas inclus dans le graphe et que s' n'est pas gagnant (puisqu'alors le jeu se termine). De plus, nous avons besoin de propager en arrière des mises-à-jour à la source (s, B) si s' est en fait gagnant. Si l'état s' doit être exploré en arrière nous devons revenir à tous les états sources qui mènent à s'. Pour ce faire, nous calculons l'opération $pred_t$ [CAS 05] (des prédécesseurs temporels des états gagnants en évitant les états perdants) et nous propageons en arrière les sous-ensembles gagnants aux source $(s*, B)$ pour celles qui se retrouvent avec de nouveaux sous-ensembles gagnants.

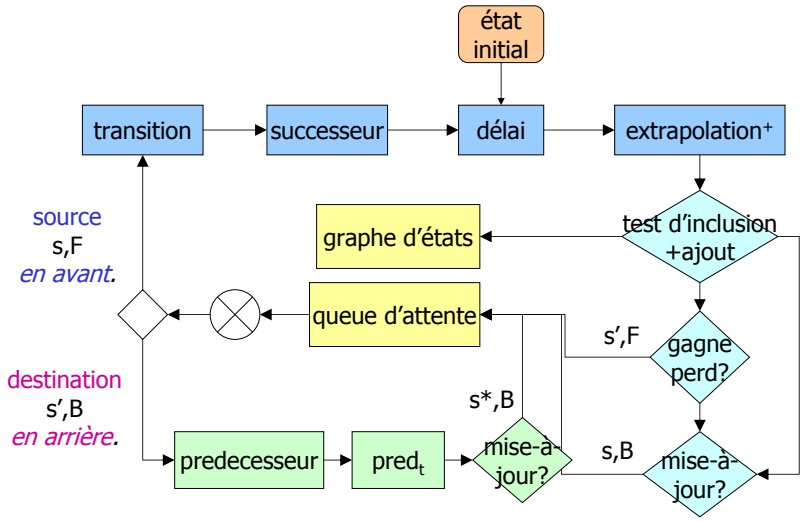

Figure 1.18. *Le pipeline d'atteignabilité d'*UPPAAL-TIGA.

Nous notons que l'algorithme donné ne calcule seulement que l'ensemble des états gagnants mais pas une *stratégie*. Les stratégies sont calculées à la volée en ajoutant la correspondance état-vers-action quand un état est gagnant et prendre cette action mène à un autre état gagnant. De plus, si la partie gagnante est obtenue par délai alors la correspondance se fait pour une action de délai. Le point important est de garder les correspondances précédantes des états-vers-action quand on ajoute de nouvelles correspondances alors que de nouveaux états gagnants sont découverts à la volée. En effet, cela permet de garantir le progrès sur un chemin déjà connu et d'éviter d'être pris dans une boucle.

1.4.3. *Optimalité Temporelle*

L'optimalité temporelle pour les jeux d'atteignabilité consiste à calculer le meilleur temps (optimal) pour qu'un contrôleur soit garanti d'atteindre un état gagnant. Si t^* est le temps optimal, le contrôleur a une stratégie qui garantit d'atteindre cet état en au plus t^* unités de temps quelles que soient les actions de l'adversaire et de plus, le controleur n'a pas une telle stratégie pour $t < t^*$. Ce problème est résolu dans [CAS 05] en rajoutant une nouvelle horloge z à l'automate de jeu temporisé original ainsi que l'invariant $Inv(\ell) \equiv z \leq T$ pour toutes les localités ℓ où T est une borne supérieure pour atteindre l'état gagnant. De plus, z n'est pas contraint dans l'état initial. L'algorithm est alors de calculer le point-fixe de tous les états gagnants et d'utiliser

z pour déduire le temps optimal. En pratique, nous calculons itérativement la borne supérieure à la volée pour couper l'espace des états quand une solution est trouvée et nous continuons d'exécuter l'algorithme pour affiner l'optimal courant. Le temps optimal est donné dans l'état initial par l'interval entre le maximum de z sur son axe à zéro et T. Quand nous mettons à jour T itérativement, l'algorithme converge en fait vers T qui devient la borne supérieure en question et le maximum de z devient zéro.

Les requêtes de temps optimal sont définies par les formules *control_t*(u,g) :* $A[\ \phi\ \mathcal{U}\ \psi\ ]$, valides que pour l'atteignabilité. L'expression supplémentaire u définit une borne supérieure pour couper la recherche, ce qui correspond a T dans l'algorithme. Cette borne supérieure est mise-à-jour à la volée. L'expression g donne une borne inférieure à partir de l'état courant dans la recherche vers l'état (but) gagnant. Les états qui sont au temps $t + g > u$ sont coupés de la recherche. Ici, t est le temps écoulé depuis l'état initial. En cas de doute, il est toujours possible d'assigner u à une grande valeur et g à zéro mais des valeurs (ou fonctions pour g) qui ont un sens aident la recherche de façon significative.

1.4.4. *Strategies Coopératives*

Pour les jeux où il n'y a pas de stratégie gagnante, il peut être utile de savoir quel est l'ensemble maximal des états pour lesquels il y a une stratégie gagnante et comment l'environement peut "aider" le contrôleur à atteindre un de ces états. Nous appelons de telles stratégies coopératives [DAV 08]. Par "aider" nous voulons dire que soit l'environement prend des actions incontrôlables favorables ou il laisse le contrôleur prendre ses actions au lieu de l'en empêcher. Le résultat de la recherche est alors une partition entre i) les états qui ont une stratégie gagnante, ii) ceux qui ont besoin de la coopération de l'environement et iii) ceux pour lesquels il est impossible de gagner.

L'algorithme utilise l'analyse précédante d'atteignabilité comme composant. L'algorithme est montré dans la figure 1.19. L'idée principale est de calculer le point-fixe du modèle original (et de construire la stratégie à la volée) et ensuite de recalculer le point-fixe du modèle modifié où nous considérons toutes les transitions contrôlables, c.a.d. l'environement aide le contrôleur. Le point le plus important ici est de compléter la stratégie précédemment obtenue avec les nouvelles actions qui sont alors coopératives. Par compléter nous voulons dire que nous rajoutons des nouvelles correspondances états-vers-action mais nous préservons les anciennes. Ces actions font parties de la stratégie coopérative, les actions originales de la stratégie gagnante pour les sous-ensemble des états gagnants et pour tous les autres états il n'est pas possible de gagner. L'algorithme calcule ces deux point-fixes bien que le premier peut se terminer prématurément s'il apparaît qu'il y a une stratégie gagnante à partir de l'état initial. Si l'ensemble des états gagnants n'est pas atteignable du tout alors il n'y a aucun espoir de stratégie. En fait il y a une stratégie coopérative *ssi* l'état gagnant est atteignable (dans le pire des cas l'environement coopére toujours).

fixpoint(G,l_0, Z_0*)*
if *win(*l_0, Z_0*)* **then return true**
if ¬*reached(Win)* **then return false**
fixpoint(G[c/u],l_0, Z_0*)*
return true

Figure 1.19. *L'algorithme dátteignabilité coopérative d'*UPPAAL-TIGA.

Les stratégies coopératives sont vérifiées par les formules de type $E <> \Phi$ où Φ est une formule d'UPPAAL-TIGA habituelle (avec *control :*). L'algorithme est généralisé à la sûreté aussi où l'environement doit éviter des états perdants.

1.5. ROMÉO : un outil pour l'analyse des extensions temporelles des réseaux de Petri

Dans cette section, nous présentons les fonctionnalités de l'outil ROMÉO dont l'objectif est l'analyse et la simulation de systèmes réactifs modélisés à l'aide d'extensions temporelles de réseaux de Petri. Il s'agit d'un logiciel libre, téléchargeable à l'adresse `http ://romeo.rts-software.org/`, disponible pour les systèmes d'exploitation Linux, Mac OS X et Windows.

ROMÉO traite les réseaux de Petri T-temporels, c'est-à-dire des réseaux pour lesquels des intervalles de temps $[a(t), b(t)]$ sont associés à chaque transition t du réseau (dans la suite de ce chapitre, nous appellerons de tels réseaux des réseaux de Petri temporels). Il permet non seulement de calculer l'espace d'états de ces modèles mais aussi de vérifier à la volée des propriétés temporelles quantitatives. Il est capable de traduire des réseaux de Petri temporels vers les automates temporisés en préservant la sémantique comportementale (bisimulation temporelle) du réseau. Le programme est en outre capable de modéliser et traiter une extension des réseaux de Petri temporels intégrant les mécanismes de préemption : les réseaux de Petri temporels étendus à l'ordonnancement à priorités fixes (*Scheduling-TPN*).

Les personnes contribuant, ou ayant contribué, au développement du logiciel sont les suivantes : Olivier (H.) Roux, Didier Lime, Guillaume Gardey, Charlotte Seidner, Louis-Marie Traonouez, Gilles Bénattar et moi-même.

1.5.1. *Modèles*

1.5.1.1. *Réseaux de Petri temporels*

Les réseaux de Petri temporels ont été définis par Merlin [**?**]. Le temps est pris en compte au niveau des transitions par l'ajout d'un intervalle temporel spécifiant les

instants de franchissement de la transition par rapport à l'instant où la transition a été nouvellement sensibilisée pour la dernière fois.

Définition 1.5.1 (Réseau de Petri temporel) *Un réseau de Petri temporel est un 7-uplet* $\mathcal{N} = \langle P, T, {}^\bullet(.), (.)^\bullet, a, b, M_0 \rangle$ *où :*

– $P = \{P_1, \ldots, P_m\}$ *est un ensemble fini et non vide de* places *;*

– $T = \{t_1, \ldots, t_n\}$ *est un ensemble fini et non vide de* transitions *;*

– ${}^\bullet(.) : T \to \mathbb{N}^P$ *est la* fonction d'incidence amont *;*

– $(.)^\bullet : T \to \mathbb{N}^P$ *est la* fonction d'incidence aval *;*

– $a : T \to \mathbb{N}$ *est la fonction donnant la* date de tir au plus tôt *d'une transition ;*

– $b : T \to \mathbb{N} \cup \{\infty\}$ *est la fonction donnant* la date de tir au plus tard *d'une transition ;*

– $M_0 \in \mathbb{N}^P$ *est le* marquage initial *du réseau.*

Un *marquage* du réseau de Petri est un vecteur de $\mathbb{N}^P$ tel que pour toute place $p \in P$, $M(p)$ est le nombre de *jetons* dans la place p.

Une transition t est dite *sensibilisée* par le marquage M si le nombre de jetons dans chaque place en amont de t est supérieur ou égal à la valuation de l'arc entre cette place et la transition ($M \geq^\bullet t$). Nous notons alors $t \in enabled\,(M)$..

Une transition t est dite *désensibilisée* par le tir de t' depuis M si elle est sensibilisée par M mais pas par $M - {}^\bullet t'$. Nous notons alors $t \in disabled(M, t')$.

Une transition t est dite *nouvellement sensibilisée* par le tir d'une transition t' si elle est sensibilisée par le marquage $M - {}^\bullet t' + t'^\bullet$ mais ne l'est pas par le marquage $M - {}^\bullet t'$. Si $t = t'$ alors t est nouvellement sensibilisée. Nous notons alors $t \in \uparrow enabled\,(M, t')$ où $\uparrow enabled\,(\cdot, \cdot)$ est défini par :

$$\uparrow enabled\,(M, t') = \{t \in T \,|\, M - {}^\bullet t' + t'^\bullet \geq^\bullet t \wedge (t = t' \vee \neg(M - {}^\bullet t' \geq^\bullet t)\}$$

Nous donnons la sémantique d'un réseau de Petri temporel en temps dense sous la forme d'un système de transitions temporisé.

Définition 1.5.2 (Sémantique d'un réseau de Petri temporel en temps dense) *La sémantique d'un réseau de Petri temporel* $\mathcal{N}$ *en temps dense est définie sous la forme d'un système de transitions temporisé* $\mathcal{S}_{\mathcal{N}}^{dense} = (Q, q_0, T, \to)$ *tel que :*

– $Q = \mathbb{N}^P \times (\mathbb{R}^+)^T$

– $q_0 = (M_0, \overline{0})$

– $\rightarrow \in Q \times (\mathbb{R}^+ \cup T) \times Q$ *est la relation de transition incluant des transitions continues et des transitions discrètes :*

- soient $q = (M, \nu) \in Q$ *et* $q' = (M, \nu') \in Q$ *deux états du réseau, la relation de transition continue est définie* $\forall d \in \mathbb{R}^+$ *par :*

$$(M, \nu) \xrightarrow{d} (M, \nu') \text{ ssi } \forall t_i \in T, \begin{cases} \nu'(t_i) = \nu(t_i) + d \\ M \geq^\bullet t_i \Rightarrow \nu'(t_i) \leq b(t_i) \end{cases}$$

- soient $q = (M, \nu) \in Q$ *et* $q' = (M', \nu') \in Q$ *deux états du réseau, la relation de transition discrète est définie* $\forall t_i \in T$ *par :*

$$(M, \nu) \xrightarrow{t_i} (M', \nu') \text{ ssi } \begin{cases} t_i \in enabled(M) \\ M' = M -^\bullet t_i + t_i^\bullet \\ a(t_i) \leq \nu(t_i) \leq b(t_i) \\ \forall t_k \in T, \nu'(t_k) = \begin{cases} 0 \text{ si } t_k \in \uparrow enabled(M, t_i) \\ \nu(t_k) \text{ sinon} \end{cases} \end{cases}$$

Dans l'approche en *temps discret*, le temps est vu comme une variable qui *« saute » d'un entier à l'autre* sans que l'on se préoccupe de ce qui peut survenir entre deux tics d'horloge. Bien entendu, les comportements d'un modèle exprimé en temps discret sont inclus dans ceux du modèle correspondant muni d'une sémantique de temps dense. Nous définissons ainsi la sémantique d'un réseau de Petri temporel en *temps discret* $\mathcal{N}$ sous la forme d'un système de transitions dans lequel nous distinguons deux types de relations discrètes : d'une part, une relation de transition discrète modifiant le marquage du réseau et, d'autre part, une relation de transition correspondant à un écoulement discret du temps (qui se caractérise par l'incrémentation d'une unité des horloges associées à chaque transition). Ainsi, nous choisissons d'exprimer ce système de transitions sous la forme d'un système de transitions temporisé $\mathcal{S}_{\mathcal{N}}^{discrete} = (Q, q_0, T, \rightarrow)$: partant de la définition que nous avons donnée pour le temps dense, nous remplaçons la relation de transition continue par une relation de transition de temps discret :

$$(M, \nu) \xrightarrow{1} (M, \nu') \text{ ssi } \forall t_i \in T, \begin{cases} \nu'(t_i) = \nu(t_i) + 1 \\ M \geq^\bullet t_i \Rightarrow \nu'(t_i) \leq b(t_i) \end{cases}$$

1.5.1.2. *Réseaux de Petri temporels étendus à l'ordonnancement*

Les réseaux de Petri temporels étendus à l'ordonnancement (ou *Scheduling time Petri nets*) constituent une sous-classe des réseaux de Petri à chronomètres. Ils ont été introduits par Roux et Déplanche dans [**?**]. Ils étendent les réseaux de Petri temporels

en incluant dans la sémantique du modèle la notion de chronomètre et la politique d'ordonnancement du système.

Le modèle intégré dans ROMÉO, les réseaux de Petri temporels étendus à l'ordonnancement à priorités fixes, se définit par l'ajout de deux nouveaux attributs associés aux places du réseau : l'un désigne l'identité du processeur auquel la place est affectée, l'autre la priorité, sur ce processeur, de la tâche dont le modèle contient cette place.

Définition 1.5.3 (Réseau de Petri temporel étendu à l'ordonnancement à priorités fixes) *Un réseau de Petri temporel étendu à l'ordonnancement à priorités fixes (Scheduling-TPN) est un 10-uplet* $\mathcal{N} = \langle P, T, {}^\bullet(.), (.)^\bullet, a, b, M_0, Proc, \gamma, \omega \rangle$ *où :*

– $P = \{p_1, \ldots, p_m\}$ *est un ensemble fini et non vide de* places *;*

– $T = \{t_1, \ldots, t_n\}$ *est un ensemble fini et non vide de* transitions *;*

– ${}^\bullet(.) : T \to \mathbb{N}^P$ *est la* fonction d'incidence amont *;*

– $(.)^\bullet : T \to \mathbb{N}^P$ *est la* fonction d'incidence aval *;*

– $a : T \to \mathbb{N}$ *est la fonction donnant la* date de tir au plus tôt *d'une transition ;*

– $b : T \to \mathbb{N} \cup \{\infty\}$ *est la fonction donnant* la date de tir au plus tard *d'une transition ;*

– $M_0 \in \mathbb{N}^P$ *est le* marquage initial *du réseau.*

– $Proc = \{\emptyset, proc_1, proc_2, \ldots\}$ *est un ensemble fini de processeurs (incluant* $\emptyset$ *pour signifier qu'une place n'est assignée à aucun processeur de l'architecture matérielle) ;*

– $\gamma \in Proc^P$ *est la fonction d'allocation d'un processeur à une place ;*

– $\omega \in \mathbb{N}^P$ *est la fonction d'assignation de priorité.*

En plus des notions définies pour les réseaux de Petri temporels classiques, nous dirons qu'une transition t est *active* pour le marquage M lorsqu'elle est sensibilisée et que toutes ses places amont sont assignées à des processeurs pour lesquels aucune place de priorité supérieure n'est marquée. Nous le notons $t \in active(M)$. Les transitions sensibilisées mais non actives sont dites suspendues.

Une transition t est dite *tirable* (ou *franchissable*) lorsqu'elle est sensibilisée, qu'elle l'a été sans être suspendues pendant au moins $a(t)$ unités de temps et qu'elle n'est pas suspendue.

Définition 1.5.4 (Sémantique d'un *Scheduling-TPN* en temps dense) *La sémantique d'un réseau de Petri temporel étendu à l'ordonnancement à priorités fixes* $\mathcal{N}$ *en temps dense est définie sous la forme d'un système de transitions temporisé* $\mathcal{S}_\mathcal{N}^{dense} = (Q, q_0, T, \to)$ *tel que :*

– $Q = \mathbb{N}^P \times (\mathbb{R}^+)^T$

– $q_0 = (M_0, \overline{0})$

– $\rightarrow \in\ Q \times (T \cup \mathbb{R}) \times Q$ *est la relation de transition incluant des transitions continues et des transitions discrètes :*

- la relation de transition continue est définie $\forall d \in \mathbb{R}^+$ *par :*

$$(M,\nu) \xrightarrow{d} (M,\nu') \text{ ssi } \forall t_i \in T,$$
$$\begin{cases} \nu'(t_i) = \begin{cases} \nu(t_i) + d \text{ si } t_i \in enabled(M) \text{ et } t_i \in active(M) \\ \nu(t_i) \text{ sinon,} \end{cases} \\ M \geq^{\bullet} t_i \Rightarrow \nu'(t_i) \leq b(t_i) \end{cases}$$

- la relation de transition discrète est définie $\forall t_i \in T$ *par :*

$$(M,\nu) \xrightarrow{t_i} (M',\nu') \text{ ssi },$$
$$\begin{cases} t_i \in enabled(M) \text{ et } t_i \in active(M), \\ M' = M - max(^{\circ}t_i \times M^t, ^{\bullet}t_i) + t_i^{\bullet}, \\ a(t_i) \leq \nu(t_i) \leq b(t_i), \\ \forall t_k \in T, \nu'(t_k) = \begin{cases} 0 \text{ si } t_k \in \uparrow enabled(M, t_i) \\ \nu(t_k)\ sinon \end{cases} \end{cases}$$

Partant de la définition de la sémantique que nous avons données pour le temps dense, nous obtenons la sémantique des réseaux en temps discret en remplaçant simplement la relation de transition continue par une relation de transition de temps discret :

$$(M,\nu) \xrightarrow{1} (M,\nu') \text{ ssi } \forall t_i \in T,$$
$$\begin{cases} \nu'(t_i) = \begin{cases} \nu(t_i) + 1 \text{ si } t_i \in enabled(M) \text{ et } t_i \in active(M) \\ \nu(t_i) \text{ sinon,} \end{cases} \\ M \geq^{\bullet} t_i \Rightarrow \nu'(t_i) \leq b(t_i) \end{cases}$$

ROMÉO implante également, tant pour les réseaux de Petri temporels que pour leur extension à l'ordonnancement, les arcs de *reset* et les arcs inhibiteurs logiques. Les premiers permettent de vider l'intégralité des jetons contenus dans une place lors du tir d'une transition (facilitant ainsi la modélisation des systèmes possédant des fonctions de réinitialisation) tandis que les seconds permettent d'empêcher le tir de certaines transitions tant que certaines places sont marquées.

1.5.2. *Architecture générale*

ROMÉO se compose d'une interface graphique programmée en Tcl/Tk, d'une bibliothèque pour la simulation des réseaux et d'un module de calcul, MERCUTION,

programmé en C++. Il est dédié à la saisie, la simulation, le calcul de l'espace d'états, la vérification et le contrôle des réseaux de Petri temporels et de leur extension à chronomètres dédiée à la modélisation d'ordonnancements préemptifs, le tout en temps dense. L'outil implante des traductions des extensions temporelles des réseaux de Petri en temps discret vers les réseaux de Petri non temporisés et les systèmes à compteurs ; il offre également la possibilité de calculer symboliquement l'espace d'états de réseaux de Petri étendus à l'ordonnancement en temps discret.

Nous allons revenir plus précisément sur ces fonctionnalités dans les paragraphes suivants.

1.5.3. *Modélisation de systèmes*

Dans le cadre de l'étude de systèmes complexes, l'interface graphique de ROMÉO permet de modéliser des systèmes réactifs ou préemptifs en utilisant des réseaux de Petri temporels ou des réseaux de Petri étendus à l'ordonnancement. Ces modèles bénéficient d'une représentation graphique simple et pour laquelle il est facile de mettre en œuvre les principales fonctions du temps réel (le parallélisme, la synchronisation, la gestion de ressources, les chiens de garde, etc.).

En tant qu'aide à la modélisation, Roméo implante des outils de simulation pour les RdPT et les *Scheduling-TPN* et de vérification formelle sur les RdPT. Il concourt ainsi à la détection de problèmes de modélisation très en amont du processus de conception.

1.5.4. *Vérification de propriétés*

Une fois le système décrit sous la forme d'un modèle (réseaux de Petri temporels ou réseaux de Petri à chronomètres), il importe de formaliser les spécifications de sa correction. L'expression de propriétés sous la forme d'observateurs ou d'une logique dédiée constituent les deux voies privilégiées permettant de s'acquitter de cette tâche.

1.5.4.1. *Model-checking en ligne*

ROMÉO offre une implantation pratique d'algorithmes pour la vérification de propriétés d'un fragment de la logique TCTL (*Time Computation Tree Logic*) dédiée aux réseaux de Petri temporels (baptisée TPN-TCTL [**?**]) en temps dense. Il est ainsi possible de vérifier des propriétés temporelles quantitatives à la volée sur ce modèle.

1.5.4.2. *Model-checking hors ligne*

1.5.4.2.1. Vérification à l'aide d'observateurs

Un observateur est un réseau de Petri temporel qui est ajouté au réseau initial *de manière non intrusive* (c'est-à-dire que l'observateur ne doit pas modifier le comportement du système initial) et qui permet de déterminer la valeur de vérité d'une propriété

[**?**]. Celle-ci est alors donnée par l'occurrence ou non d'un marquage ou du tir d'une transition dans l'espace d'états du système « réseau de Petri temporel (à chronomètres) observé + observateur ».

Cette démarche présente l'avantage de transformer la propriété à vérifier en un problème d'accessibilité de marquage ou d'exécution de trace. Elle souffre néanmoins de plusieurs limitations. D'une part, il n'existe pas de techniques automatiques de génération d'observateurs. Or il est parfois délicat de ramener le problème de vérification d'une propriété à un problème d'accessibilité. D'autre part, chaque propriété requiert un observateur spécifique et donc un nouveau calcul de l'espace d'états. Enfin, l'observateur est souvent de taille aussi importante que le modèle initial, accroissant ainsi de façon non négligeable le coût de la vérification de la propriété.

1.5.4.2.2. Vérification basée sur des traductions vers d'autres modèles

Une alternative intéressante pour vérifier des propriétés temporelles quantitatives sur des réseaux de Petri temporels consiste à établir une traduction des réseaux de Petri vers les automates temporisés. En effet, il existe, sur ces derniers, de nombreux outils efficaces pour la vérification de telles propriétés. Les traductions se divisent en deux catégories : d'une part, les *traductions structurelles* (telle la traduction proposée dans [**?**] d'un réseau de Petri temporel non nécessairement borné à dates de tir au plus tard finies ou infinies vers un automate temporisé), d'autre part les *traductions avec calcul de l'espace d'états* (à l'instar de la traduction présentée dans [**?**, **?**] qui consiste à obtenir l'espace d'états d'un réseau de Petri temporel sous la forme d'un automate temporisé).

Roméo implante différentes méthodes théoriques permettant de traduire les modèles analyses en automates, automates temporisés (TA) ou automates à chronomètres (*SWA*). Cette approche a l'avantage que plusieurs outils efficaces de *model-checking* existent sur ces modèles (MEC, ALDEBARAN, UPPAAL, KRONOS, HYTECH). Ces traductions étendent la classe de propriétés qui peuvent être vérifiées à l'aide d'observateurs à des logiques temporelles (LTL, CTL) et temporelles quantitatives (TCTL).

Une première traduction consiste à calculer le graphe des classes d'états qui fournit une représentation finie du comportement de réseaux bornés tout en préservant les propriétés LTL [**?**]. Dans le cadre des réseaux de Petri temporels bornés, l'algorithme se base sur la structure de données DBM tandis que pour les réseaux de Petri temporels étendus, le semi-algorithme manipule des polyèdres généraux (par l'intermédiaire de la bibliothèque *Parma Polyhedra Library* [**?**]).

Deux méthodes différentes sont proposées pour générer un automate temporisé à partir d'un réseau de Petri temporel en préservant sa sémantique (au sens de la bisimulation temporelle) : la première est inspirée de la théorie sur les automates temporisés

[**?**], la seconde est dérivée de l'approche classique du graphe des classes d'états [**?**]. Pour cette dernière, le logiciel implémente une méthode de réduction du nombre d'horloges telle que la vérification sur l'automate temporisé résultant soit plus efficace. Pour les deux procédures, les automates sont générés de manière à être conforme au format d'UPPAAL ou de KRONOS.

Sur les réseaux de Petri temporels étendus à l'ordonnancement sont implantées les méthodes approchées et exactes introduites dans [**?**, **?**]. La première permet d'effectuer une traduction rapide en un automate à chronomètres en utilisant un algorithme sur-approximant (basé sur les DBM). Même s'il s'agit d'une surapproximation, il a été montré que l'automate à chronomètres est bisimilaire au réseau de Petri original. L'automate à chronomètres est généré au format d'entrée de HYTECH et est calculé avec un faible nombre de chronomètres. Puisque le nombre de chronomètres est critique pour la complexité de la vérification, la méthode accroît ainsi l'efficacité de l'analyse temporisée du système. Avantage supplémentaire : dans certains cas, cette méthode rend l'analyse possible alors même qu'elle aurait conduit à une impasse si le système avait été modélisé directement avec HYTECH.

1.5.4.2.3. Vérification d'un fragment de TCTL pour les réseaux de Petri temporels

Les auteurs de [**?**, **?**] sont allés plus loin en définissant une logique TCTL « native » pour les réseaux de Petri temporels en temps dense, appelée *TPN-TCTL*. Ils ont prouvé la décidabilité du *model-checking* de *TPN-TCTL* sur les réseaux de Petri temporels bornés et ont montré que sa complexité est PSPACE. Ils ont également introduit un fragment de *TPN-TCTL* pour lequel ils proposent des algorithmes de vérification à la volée et une implantation pratique dans le logiciel ROMÉO.

1.5.4.2.4. Vérification d'un fragment de TCTL pour les réseaux de Petri à chronomètres

Il a été montré qu'en temps dense, l'accessibilité d'état et de marquage ne sont pas décidables sur les réseaux de Petri à chronomètres, même bornés [**?**]. Ces problèmes deviennent toutefois décidables lorsqu'une sémantique de temps discret est considérée [**?**]. Une méthode efficace de calcul symbolique de l'espace d'états a alors été proposée sur les réseaux de Petri à chronomètres [**?**].

La démarche consiste à étendre les représentations symboliques classiques du temps dense (assurées via des polyèdres convexes) au temps discret. Pour ce faire, une procédure envisagée est de calculer l'espace d'états des réseaux en temps discret comme la discrétisation de l'espace d'états du modèle associé en temps dense. Cette démarche, correcte pour les réseaux de Petri temporels, ne l'est pas pour les réseaux à chronomètres : en effet, pour ces derniers, une telle procédure peut conduire à ajouter de faux comportements discrets, c'est-à-dire des comportements qui ne sont pas

accessibles avec la sémantique en temps discret. Une solution a été proposée pour surmonter ce problème : elle consiste à décomposer les polyèdres généraux représentant l'information temporelle du réseau en une union de polyèdres plus simples garantissant la validité du calcul du successeur symbolique.

Il est dès lors possible de vérifier des propriétés temps réel exprimées à l'aide de TCTL sur les réseaux de Petri à chronomètres bornés en temps discret via une adaptation très simple de l'outil ROMÉO [**?**].

Donnons-en une appréciation intuitive. Dans [**?**, **?**], les auteurs proposent une démarche pour vérifier des propriétés exprimées dans la logique TCTL (ou dans une sous-classe de la logique TCTL) sur les réseaux de Petri temporels via le graphe des zones. Cette méthode est naturellement étendue du graphe des classes [**?**]. En fait, la démarche suivie est générale : elle s'applique à toutes les extensions temporisées des réseaux de Petri telles que les domaines de tir de toutes les classes d'états du modèle sont des DBM. D'autre part, les auteurs de [**?**] proposent un algorithme pour calculer l'espace d'états de réseaux de Petri à chronomètres en temps discret en n'utilisant que des DBM. La combinaison des deux procédures permet d'obtenir une méthode élégante pour vérifier des formules TCTL sur les réseaux de Petri à chronomètres en temps discret.

Grâce à l'implantation pratique de ces algorithmes dans Roméo, le logiciel est capable de vérifier des propriétés temporelles quantitatives sur les réseaux de Petri à chronomètres dotés d'une sémantique de temps discret.

1.5.5. *Comparaisons avec d'autres outils*

Les tableaux 1.5.5 et 1.5.5 comparent les fonctionnalités de Roméo à celles des deux autres principaux outils permettant l'analyse des réseaux de Petri temporels et d'extension prenant en compte la préemption. Ils mettent en parallèle les capacités respectives de chacun des outils en termes de classes de propriétés qui peuvent être vérifiées.

TINA [**?**] est un outil pour l'analyse des réseaux de Petri temporels ; il repose principalement sur des techniques de calcul du graphe des classes d'états. ORIS [**?**] est un programme qui analyse une extension des réseaux de Petri temporel prenant en compte la préemption, extension qui est équivalente aux *Scheduling-TPNs*.

Le principal avantage de Roméo est d'adapter les méthodes développées sur les réseaux de Petri temporel aux *Scheduling-TPNs*. Cela nous permet ensuite de vérifier des propriétés exprimées en logique TCTL.

	Accessibilité	LTL	CTL	Quantitatif Vivacité [a]	TCTL
TINA	Graphe des marquages	SCG[b] + MC[c]	SCG atomique[b] + MC[c]	–	–
Roméo	Vérification à la volée ou Graphe des marquages	SCG + MC[c] ou ZFG[e] + MC[c]	Traduction en Automate Temporisé + UPPAAL [d] ou KRONOS Ou vérif. d'un fragment de TCTL		

Tableau 1.1. *Fonctionnalités de Roméo sur les RdPTs*

0. Inclut les propriétés de réponse telle que $\forall\Box(\varphi \implies \forall\Diamond\Psi)$ où φ ou Ψ peuvent être des contraintes sur les horloges.

0. SCG = Calcul du graphe des classes d'états (*State Class Graph*).

0. MC = Requiert l'utilisation d'un *model-checker* sur le SCG ou le ZFG.

0. Sous-ensemble de TCTL.

0. ZFG = Calcul en avant du graphe des zones (*Zone-based Forward Graph*).

	Calcul de l'espace d'états		Analyse temporisée	
	Surapproximation	Abstraction exacte	RTTL	TCTL
ORIS	DBM	–	DBM-SCG[a] + MC[b]	–
Roméo	DBM	SCG[c]	Traduction efficace en automates à chronomètres temporellement bisimilaire + HYTECH Ou fragment de TCTL en temps discret	

Tableau 1.2. *Fonctionnalités de Roméo sur les Scheduling-TPNs*

0. Calcul d'une surapproximation du graphe des classes d'états via DBM.

0. Oris permet l'analyse ponctuelle exacte de traces (en regard d'une variante temporellement linéaire de *Real-Time Temporal Logic* (RTTL).

0. Comme sur les RdPTs, le SCG préserve les propriétés LTL.

1.5.6. *Cas d'étude*

1.5.6.1. *Description*

Dans cette sous-partie, nous travaillons sur le modèle partiel d'un système de contrôle d'un compensateur à oscillations (absorbeur de chocs hydraulique) et d'un frein différentiel sur un tracteur. Le système partiel est composé de processeurs qui exécutent un système d'exploitation temps réel, l'ensemble étant relié avec un bus CAN.

Pour cet exemple, nous avons utilisé la traduction d'un *Scheduling-TPN* en un *SWA* dont l'espace d'états est calculé avec HYTECH.

L'efficacité de cette méthode a été comparée avec une modélisation directe réalisée avec HYTECH Ḋes systèmes plus ou moins complexes, obtenus en ajoutant ou retirant des tâches et/ou des processeurs, ont également été testés. Le tableau 1.3 synthétise les résultats obtenus.

Les colonnes 2 et 3 donnent le nombre de processeurs et de tâches du système. Les colonnes 4, 5 et 6 décrivent les résultats d'une modélisation directe avec HYTECH : le nombre d'automates à chronomètres utilisés pour modéliser le système, le nombre de chronomètres et le temps nécessaire à HYTECH pour calculer l'espace d'états. Pour cette modélisation générique, nous avons utiliser le produit d'un automate à chronomètres par tâche et d'un automate à chronomètres par ordonnanceur. Les colonnes 7 à 11 donnent les résultats pour la méthodes consistant à traduire un *Scheduling-TPN* en *SWAs* : le nombre de localités, de transitions, d'horloges générées et le temps pris pour la construction de l'automate. Enfin, la dernière colonne donne le temps pris par HYTECH pour calculer l'espace d'états de l'automate à chronomètres généré. Les temps sont donnés en secondes et NA signifie que le calcul avec HYTECH n'a pas terminé sur la machine utilisée pour les tests.

	Description		Modélisation directe via SWA			Méthode via une traduction [1]				
Ex.	Proc.	Tâches	SWA	Chrono.	t_{HYTECH}	Loc.	Trans.	Chrono.	$t_{\text{Roméo}}$	t_{HYTECH}
1	2	4	8	7	77.8	20	29	3	$\leq$0.1	0.2
2	3	6	11	9	590.3	40	58	4	$\leq$0.1	0.5
3	3	7	12	10	NA	52	84	4	$\leq$0.1	0.7
4	3+CAN	7	13	11	NA	297	575	7	0.3	5.3
5	4+CAN	9	15	13	NA	761	1 677	8	0.9	29.8
6	5+CAN	11	17	15	NA	1 141	2 626	9	6	60.1
7	6+CAN	14	.	.	NA	4 587	12 777	10	59.7	438.8
8	7+CAN	18	.	.	NA	8 817	25 874	12	1 146.7	NA

Tableau 1.3. *Résultats expérimentaux*

0. *Scheduling-TPN* $\xrightarrow{\text{Roméo}}$ SWA $\xrightarrow{\text{HYTECH}}$ espace d'états

Ces calculs ont été effectués sur un POWERPC G4 1.25GHz équipé de 512Mo de mémoire vive.

Nous constatons que le calcul sur la modélisation directe à base de produits d'automates à chronomètres conduit rapidement à des systèmes trop gros pour être calculés (cf. exemple 3). *A contrario*, avec la méthode basée sur une traduction, il est possible de vérifier des systèmes de taille bien plus importante.

1.6. Bibliographie

[ABA 92] ABADI M., LAMPORT L., « An Old-Fashioned Recipe for Real Time », *Proc. of REX Workshop "Real-Time : Theory in Practice"*, n° 600LNCS, Springer, p. 1–27, 1992.

[ABD 01a] ABDULLA P. A., « Using (Timed) Petri Nets for Verification of Parameterized (Timed) systems », *VEPAS'2001, Verification of Parameterized Systems, ICALP'2001 satellite workshop*, 2001.

[ABD 01b] ABDULLA P. A., NYLÉN A., « Timed Petri Nets and BQQs », *Proceedings of ICATPN'2001, 22nd International Conference on application and theory of Petri nets*, 2001.

[AHU 93] AHUJA R. K., MAGNANTI T. L., ORLIN J. B., *Network Flows - Theory, Algorithms, and Applications*, Prentice Hall, 1993.

[ALU 90a] ALUR R., COURCOUBETIS C., DILL D. L., « Model-checking for Real-time Systems », *5th Symposium on Logic in Computer Science (LICS'90)*, p. 414–425, 1990.

[ALU 90b] ALUR R., DILL D. L., « Automata for Modeling Real-Time Systems », *Proc. of Int. Colloquium on Algorithms, Languages, and Programming*, vol. 443 de *LNCS*, p. 322–335, 1990.

[ALU 94] ALUR R., DILL D. L., « A theory of timed automata », *Theoretical Computer Science*, vol. 126, n°2, p. 183–235, 1994.

[ALU 01] ALUR R., LA TORRE S., PAPPAS G. J., « Optimal Paths in Weighted Timed Automata », *Fourth International Workshop on Hybrid Systems : Computation and Control*, vol. 2034 de *Lecture Notes in Computer Science*, Springer, p. 49–62, 2001.

[AMN 01] AMNELL T., BEHRMANN G., BENGTSSON J., D'ARGENIO P. R., DAVID A., FEHNKER A., HUNE T., JEANNET B., LARSEN K. G., MÖLLER M. O., PETTERSSON P., WEISE C., YI W., « UPPAAL - Now, Next, and Future », CASSEZ F., JARD C., ROZOY B., RYAN M., Eds., *Modelling and Verification of Parallel Processes*, n° 2067Lecture Notes in Computer Science Tutorial, Springer–Verlag, p. 100–125, 2001.

[ASA 98] ASARIN E., MALER O., PNUELI A., SIFAKIS J., « Controller Synthesis for Timed Automata », *Proc. IFAC Symp. on System Structure & Control*, Elsevier Science, p. 469-474, 1998.

[BAI 08] BAIER C., KATOEN J.-P., *Principles of Model Checking*, MIT Press, 2008.

[BAL 96] BALARIN F., « Approximate reachability analysis of timed automata », *17th IEEE Real-Time Systems Symposium*, IEEE Computer Society Press, 1996.

[BEH 00] BEHRMANN G., HUNE T., VAANDRAGER F., « Distributed Timed Model Checking - How the Search Order Matters », *Proc. of 12th International Conference on Computer Aided Verification*, Lecture Notes in Computer Science, Chicago, Springer, Juli 2000.

[BEH 01a] BEHRMANN G., DAVID A., LARSEN K. G., MÖLLER M. O., PETTERSSON P., YI W., « UPPAAL - Present and Future », *Proc. of 40*th *IEEE Conference on Decision and Control*, IEEE Computer Society Press, 2001.

[BEH 01b] BEHRMANN G., FEHNKER A., HUNE T., LARSEN K. G., PETTERSSON P., ROMIJN J., « Efficient Guiding Towards Cost-Optimality in UPPAAL », MARGARIA T., YI W., Eds., *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, n° 2031Lecture Notes in Computer Science, Springer, p. 174–188, 2001.

[BEH 01c] BEHRMANN G., FEHNKER A., HUNE T., LARSEN K. G., PETTERSSON P., ROMIJN J., VAANDRAGER F., « Minimum-Cost Reachability for Priced Timed Automata »,

BENEDETTO M. D. D., SANGIOVANNI-VINCENTELLI A., Eds., *Proceedings of the 4th International Workshop on Hybris Systems : Computation and Control*, n° 2034Lecture Notes in Computer Sciences, Springer, p. 147–161, 2001.

[BEH 02] BEHRMANN G., BENGTSSON J., DAVID A., LARSEN K. G., PETTERSSON P., YI W., « UPPAAL Implementation Secrets », *Proc. of 7*th *International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems*, 2002.

[BEH 03a] BEHRMANN G., LARSEN K. G., PELANEK R., « To store or not to store », *Proceedings of the 15th International Conference on Computer Aided Verification*, vol. 2725 de *LNCS*, Springer Verlag, p. 433–445, 2003.

[BEH 03b] BEHRMANN G., DAVID A., LARSEN K. G., YI W., « Unification & Sharing in Timed Automata Verification », *SPIN Workshop 03*, vol. 2648 de *LNCS*, p. 225–229, 2003.

[BEH 04a] BEHRMANN G., BOUYER P., LARSEN K., PELNEK R., « Lower and upper bounds in zone based abstractions of timed automata », *TACAS 2004*, vol. 2988 de *LNCS*, Springer–Verlag, p. 312–326, 2004.

[BEH 04b] BEHRMANN G., DAVID A., LARSEN K. G., « A Tutorial on UPPAAL », BERNARDO M., CORRADINI F., Eds., *Formal Methods for the Design of Real-Time Systems : 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, n° 3185LNCS, Springer–Verlag, p. 200–236, September 2004.

[BEH 05a] BEHRMANN G., BRINKSMA E., HENDRIKS M., MADER A., « Scheduling Lacquer Production by Reachability Analysis – A Case Study », *Workshop on Parallel and Distributed Real-Time Systems 2005*, IEEE Computer Society, p. 140-, 2005.

[BEH 05b] BEHRMANN G., LARSEN K. G., RASMUSSEN J. I., « Optimal scheduling using priced timed automata », *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 32, n°4, p. 34–40, ACM Press, 2005.

[BEH 07] BEHRMANN G., COUGNARD A., DAVID A., FLEURY E., LARSEN K. G., LIME D., « UPPAAL-TIGA : Time for Playing Games ! », *Proceedings of the 19th International Conference on Computer Aided Verification*, n° 4590LNCS, Springer, p. 121–125, 2007.

[BEN 98] BENGTSSON J., LARSEN K. G., LARSSON F., PETTERSSON P., WANG Y., WEISE C., « New Generation of UPPAAL », *Int. Workshop on Software Tools for Technology Transfer*, juin 1998.

[BEN 02] BENGTSSON J., Clocks, DBMs and States in Timed Systems, PhD thesis, Uppsala University, 2002.

[BER 91] BERTHOMIEU B., DIAZ M., « Modeling and verification of time dependent systems using time Petri nets », *IEEE Trans. on Software Engineering*, vol. 17, n°3, p. 259–273, 1991.

[BOW 98] BOWMAN H., FACONTI G. P., KATOEN J.-P., LATELLA D., MASSINK M., « Automatic Verification of a Lip Synchronisation Algorithm using UPPAAL », JAN FRISO GROOTE B. L., VAN WAMEL J., Eds., *In Proceedings of the 3rd International Workshop on Formal Methods for Industrial Critical Systems. Amsterdam , The Netherlands*, 1998.

[CAS 00] CASSEZ F., LARSEN K. G., « The Impressive Power of Stopwatches », *CONCUR 2000*, vol. 1877 de *LNCS*, Springer-Verlag, p. 138–152, 2000.

[CAS 05] CASSEZ F., DAVID A., FLEURY E., LARSEN K. G., LIME D., « Efficient On-the-fly Algorithms for the Analysis of Timed Games », *CONCUR'05*, vol. 3653 de *LNCS*, Springer–Verlag, p. 66–80, August 2005.

[D'A 97] D'ARGENIO P. R., KATOEN J.-P., RUYS T. C., TRETMANS J., « The bounded retransmission protocol must be on time ! », *In Proceedings of the 3rd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, vol. 1217 de *LNCS*, Springer–Verlag, p. 416–431, April 1997.

[DAV 00] DAVID A., YI W., « Modelling and Analysis of a Commercial Field Bus Protocol », *Proceedings of the 12th Euromicro Conference on Real Time Systems*, IEEE Computer Society, p. 165–172, 2000.

[DAV 02] DAVID A., BEHRMANN G., LARSEN K. G., YI W., « New UPPAAL Architecture », PETTERSSON P., YI W., Eds., *Workshop on Real-Time Tools*, Uppsala University Technical Report Series, 2002.

[DAV 03] DAVID A., BEHRMANN G., LARSEN K. G., YI W., « A Tool Architecture for the Next Generation of UPPAAL », *10th Anniversary Colloquium. Formal Methods at the Cross Roads : From Panacea to Foundational Support*, LNCS, 2003.

[DAV 05] DAVID A., « Merging DBMs Efficiently », *17th Nordic Workshop on Programming Theory*, DIKU, University of Copenhagen, p. 54–56, October 2005.

[DAV 06] DAVID A., HÅKANSSON J., LARSEN K. G., PETTERSSON P., « Model Checking Timed Automata with Priorities using DBM Subtraction », *Proceedings of the 4th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'06)*, vol. 4202 de *LNCS*, p. 128–142, 2006.

[DAV 08] DAVID A., LARSEN K. G., LI S., NIELSEN B., « Cooperative Testing of Uncontrollable Timed Systems », Fourth Workshop on Model-Based Testing MBT'08, March 2008.

[DAW 06] DAWS C., KORDY P., « Symbolic Robustness Analysis of Timed Automata. », *FORMATS*, vol. 4202 de *Lecture Notes in Computer Science*, Springer, p. 143-155, 2006.

[DEA 01] DE ALFARO L., HENZINGER T. A., MAJUMDAR R., « Symbolic Algorithms for Infinite-State Games », *Proc. 12^{th} Conf. on Concurrency Theory (CONCUR'01)*, vol. 2154 de *LNCS*, Springer, p. 536-550, 2001.

[FLO 62] FLOYD R. W., « Acm algorithm 97 : Shortest Path », *Communications of the ACM*, vol. 5, n°6, page345, 1962.

[GAR 05] GARDEY G., LIME D., MAGNIN M., ROUX O. H., « Roméo : A tool for analyzing time Petri nets », *Proceedings of the 17th International Conference on Computer Aided Verification*, vol. 3576 de *LNCS*, Springer Berlin, p. 418-423, 2005.

[HAV 97] HAVELUND K., SKOU A., LARSEN K. G., LUND K., « Formal Modelling and Analysis of an Audio/Video Protocol : An Industrial Case Study Using UPPAAL », *Proceedings of the 18th IEEE Real-Time Systems Symposium*, p. 2–13, December 1997.

[HEN 92] HENZINGER T. A., NICOLLIN X., SIFAKIS J., YOVINE S., « Symbolic Model Checking for Real-Time Systems », *Proc. of IEEE Symposium on Logic in Computer Science*, 1992.

[HEN 94] HENZINGER T. A., « Symbolic Model Checking for Real-time Systems », *Information and Computation*, vol. 111, p. 193–244, 1994.

[HEN 02] HENDRIKS M., LARSEN K. G., « Exact Acceleration of Real-Time Model Checking », ASARIN E., MALER O., YOVINE S., Eds., *Electronic Notes in Theoretical Computer Science*, vol. 65, Elsevier Science Publishers, April 2002.

[HEN 03] HENDRIKS M., BEHRMANN G., LARSEN K., NIEBERT P., VAANDRAGER F., « Adding Symmetry Reduction to Uppaal », LARSEN K., NIEBERT P., Eds., *Proceedings of the First International Workshop on Formal Modeling and Analysis of Timed Systems (FORMATS 2003)*, vol. 2791 de *LNCS*, Springer Verlag, p. 46-49, 2003.

[HOL 91] HOLZMANN G. J., *Design and Validation of Computer Protocols*, Prentice-Hall, 1991.

[HOL 98] HOLZMANN G. J., « An Analysis of Bitstate Hashing », *Formal Methods in System Design*, vol. 13, p. 289–307, 1998.

[HUN 00] HUNE T., LARSEN K. G., PETTERSSON P., « Guided Synthesis of Control Programs Using UPPAAL », LAI T. H., Ed., *Proc. of the IEEE ICDCS International Workshop on Distributed Systems Verification and Validation*, IEEE Computer Society Press, p. E15–E22, avril 2000.

[IVE 00] IVERSEN T. K., KRISTOFFERSEN K. J., LARSEN K. G., LAURSEN M., MADSEN R. G., MORTENSEN S. K., PETTERSSON P., THOMASEN C. B., « Model-Checking Real-Time Control Programs — Verifying LEGO Mindstorms Systems Using UPPAAL », *Proc. of 12*th *Euromicro Conference on Real-Time Systems*, IEEE Computer Society Press, p. 147–155, juin 2000.

[JES 07] JESSEN J. J., RASMUSSEN J. I., LARSEN K. G., DAVID A., « Guided Controller Synthesis for Climate Controller Using UPPAAL-TIGA », *Proceedings of the 19th International Conference on Formal Modeling and Analysis of Timed Systems*, n° 4763LNCS, Springer, p. 227–240, 2007.

[KRI 96] KRISTOFFERSON K. J., LAROUSSINIE F., LARSEN K. G., PETTERSSON P., YI W., A Compositional Proof of a Real-Time Mutual Exclusion Protocol, Rapport n°RS-96-55, BRICS, December 1996.

[KRI 02] KRISTENSEN L., MAILUND T., « A Generalised Sweep-Line Method for Safety Properties », *Proc. of FME'02*, vol. 2391 de *LNCS*, Springer-Verlag, p. 549–567, 2002.

[LAR 95] LARSEN K. G., PETTERSSON P., YI W., « Model-Checking for Real-Time Systems », *Proc. of Fundamentals of Computation Theory*, n° 965Lecture Notes in Computer Science, p. 62–88, août 1995.

[LAR 97a] LARSEN K. G., PETTERSSON P., YI W., « UPPAAL in a Nutshell », *Int. Journal on Software Tools for Technology Transfer*, vol. 1, n°1–2, p. 134-152, Springer–Verlag, octobre 1997.

[LAR 97b] LARSSON F., LARSEN K. G., PETTERSSON P., YI W., « Efficient Verification of Real-Time Systems : Compact Data Structures and State-Space Reduction », *Proc. of the 18*th *IEEE Real-Time Systems Symposium*, IEEE Computer Society Press, p. 14–24, décembre 1997.

[LAR 01] LARSEN K. G., BEHRMANN G., BRINKSMA E., FEHNKER A., HUNE T., PETTERSSON P., ROMIJN J., « As Cheap as Possible : Efficient Cost-Optimal Reachability for Priced Timed Automata », BERRY G., COMON H., FINKEL A., Eds., *Proceedings of CAV 2001*, n° 2102Lecture Notes in Computer Science, Springer, p. 493–505, 2001.

[LIN 01] LINDAHL M., PETTERSSON P., YI W., « Formal Design and Analysis of a Gearbox Controller », *Springer International Journal of Software Tools for Technology Transfer (STTT)*, vol. 3, n°3, p. 353–368, 2001.

[LIU 98] LIU X., SMOLKA S., « Simple Linear-Time Algorithm for Minimal Fixed Points », *Proc. 26th Conf. on Automata, Languages and Programming (ICALP'98)*, vol. 1443 de *LNCS*, Springer, p. 53-66, 1998.

[LÖN 97] LÖNN H., PETTERSSON P., « Formal Verification of a TDMA Protocol Startup Mechanism », *Proc. of the Pacific Rim Int. Symp. on Fault-Tolerant Systems*, p. 235–242, décembre 1997.

[MAL 95] MALER O., PNUELI A., SIFAKIS J., « On the Synthesis of Discrete Controllers for Timed Systems », *Proc. 12th Symp. on Theoretical Aspects of Computer Science (STACS'95)*, vol. 900, Springer, p. 229-242, 1995.

[RAS 06] RASMUSSEN J. I., LARSEN K. G., SUBRAMANI K., « On using priced timed automata to achieve optimal scheduling », *Form. Methods Syst. Des.*, vol. 29, n°1, p. 97–114, Kluwer Academic Publishers, 2006.

[RAZ 85] RAZOUK R. R., PHELPS C. V., « Performance analysis using timed Petri nets », *Protocol Testing, Specification, and Verification*, p. 561–576, 1985.

[ROK 93] ROKICKI T. G., Representing and Modeling Digital Circuits, PhD thesis, Stanford University, 1993.

[SWA 07] SWAMINATHAN M., FRANZLE M., « A Symbolic Decision Procedure for Robust Safety of Timed Systems », *Proceedings of the 14th International Symposium on Temporal Representation and Reasoning*, IEEE Computer Society, page192, 2007.

[TRI 99] TRIPAKIS S., ALTISEN K., « Controller Synthesis for Discrete and Dense-Time Systems », *Proc. World Congress on Formal Methods in the Development of Computing Systems (FM'99)*, vol. 1708 de *LNCS*, Springer, p. 233-252, 1999.

[WON 94] WONG-TOI H., Symbolic Approximations for Verifying Real-time Systems, PhD thesis, Stanford University, 1994.

[YI 94] YI W., PETTERSSON P., DANIELS M., « Automatic Verification of Real-Time Communicating Systems By Constraint-Solving », HOGREFE D., LEUE S., Eds., *Proc. of the 7*th *Int. Conf. on Formal Description Techniques*, North–Holland, p. 223–238, 1994.

[ZUB 80] ZUBEREK W. M., « Timed Petri nets and preliminary performance evaluation », *Proceedings of the 7th anual symposium on Computer Architecture*, ACM Press, p. 88–96, 1980.

[ZUB 85] ZUBEREK W. M., « Extended D-timed Petri nets, timeouts, and analysis of communication protocols », *Proceedings of the 1985 ACM annual conference on the range of computing : mid-80's perspective*, ACM Press, p. 10–15, 1985.

Fiche pour le service de fabrication

1.7. *

Auteurs :

AFSEC

1.8. *

Titre du livre :

Systèmes embarqués communicants
Approches formelles

1.9. *

Titre abrégé :

Syst. embarqués Approches formelles

1.10. *

Date de cette version :

15 décembre 2008

1.11. *

Contact :

– téléphone : 04 92 94 27 48
– télécopie : 04 92 94 28 96
– Mél : rr@unice.fr

1.12. *

Logiciel pour la composition :

– LaTeX, avec la classe `ouvrage-hermes.cls`,
– version 1.3, 2004/07/24.

– traité (option treatise) : Non *(par défaut, livre, mêmes auteurs de chapitres)*
– livre en anglais (option english) : Non *(par défaut en français)*
– tracé des limites de page (option cropmarks) : Non *(par défaut)*
– suppression des en-têtes de page (option empty) : Non *(par défaut)*
– impression des pages blanches (option allpages) : Oui
– césures actives : voir la coupure du mot signal dans le fichier .log