Lot 4.1

Technologie de modélisation

Modélisation compositionnelle

Vérification compositionnelle des p-automates

Description :	Ce document a pour objectif de montrer comment appliquer les méthodes compositionnelles aux p-automates. Il présente d'abord le principe de la vérification compositionnelle, puis rappelle brièvement le modèle des p-automates, ainsi que celui des automates à chronomètres sur lequel l'outil HCMC s'applique. Il rappelle également les caractéristiques de la logique L_ν^h, qui permet essentiellement de spécifier des propriétés de sûreté pour des automates hybrides linéaires, donc en particulier pour des automates à chronomètres. Il montre enfin comment, à partir de résultats concernant la vérification compositionnelle des automates hybrides linéaires, on obtient une méthode qui s'applique aux p-automates. La principale difficulté concerne les paramètres. Elle est illustrée sur l'exemple de l'ABR, qui n'est pas concluant puisque HCMC effectue une surapproximation pour terminer. Cette étude de cas montre l'intérêt d'étendre HCMC par un calcul de point fixe sans approximation, qui donne une réponse exacte s'il termine.
Auteur(s) :	Béatrice Bérard, François Laroussinie
Référence :	Averroes / Lot 4.1 / Fourniture 1 / V1.0
Date :	20 novembre 2003
Statut :	validé
Version :	1.0

Réseau National des Technologies Logicielles
Projet subventionné par le Ministère de la Recherche et des Nouvelles Technologies

CRIL Technology, France Télécom R&D, INRIA-Futurs, LaBRI (Univ. de Bordeaux – CNRS), LIX (École Polytechnique, CNRS) LORIA, LRI (Univ. de Paris Sud – CNRS), LSV (ENS de Cachan – CNRS)

Historique

1 juillet 2003	V 0.1	version préliminaire
20 novembre 2003	V 1.0	mise au format averroes

Table des matières

1 Introduction

Les algorithmes usuels de model-checking, applicables aux automates finis, ont été étendus avec succès aux automates temporisés [AD94], malgré une complexité théorique importante : le problème du model-checking d'un automate temporisé est PSPACE-complet ou EXPTIME-complet selon la logique considérée. Les automates paramétrés ou p-automates forment une extension avec paramètres des automates temporisés. On peut également les considérer comme des cas particuliers d'automates hybrides linéaires [ACH+95, ACHH93, HHW95]. Bien que les questions de vérification deviennent alors indécidables, des semi-algorithmes ont été implantés dans des outils comme HYTECH, qui ont permis l'étude de nombreux systèmes industriels. Vue comme une alternative au model-checking standard, la méthode compositionnelle [And95, LL95] a également été étendue aux automates temporisés. Un semi-algorithme compositionnel a aussi été proposé pour les systèmes hybrides linéaires et implanté pour les automates à chronomètres [CL00].

La *méthode compositionnelle* part du principe qu'un système complet est généralement décrit par une composition de plusieurs éléments : $S = (H_1|\cdots|H_n)$. Étant donnée une propriété φ exprimée dans une logique adéquate, on chercher à vérifier que le système S satisfait cette propriété. La méthode consiste alors à construire une formule *quotient* φ/H_n telle que

$$(H_1|\cdots|H_n) \models \varphi \text{ ssi } (H_1|\cdots|H_{n-1}) \models \varphi/H_n.$$

La nouvelle formule φ/H_n est souvent plus grande que φ, puisqu'une partie du comportement de H_n y est codé. On applique alors des simplifications sur cette formule, afin de réduire sa taille. En répétant la transformation précédente, on se ramène à un problème de la forme $nil \models \varphi'$, où nil désigne un système vide, et φ' une formule. Il reste alors à résoudre un problème de model-checking simplifié. Il se peut aussi qu'après quotient par un H_i, la formule dérivée soit équivalente à vrai ou à faux, auquel cas on dispose directement de la réponse.

Nous allons rappeler les principales caractéristiques des p-automates, présenter la logique associée, et montrer brièvement comment adapter la méthode compositionnelle aux p-automates.

2 Les p-automates

2.1 Caractéristiques

Variables et contraintes. Les variables d'un p-automate sont : des *paramètres*, des *variables discrètes* et une *horloge* s. Ces variables à valeurs réelles diffèrent par leur évolution en fonction du temps. Les paramètres sont fixés par une contrainte initiale et n'évoluent jamais. Les variables discrètes n'évoluent pas non plus mais elles peuvent subir des mises à jour instantanées. L'horloge évolue de manière synchrone avec le temps. Une *valuation* est une application de l'ensemble V des variables dans $\mathbb{R}$.

Un *terme paramétré* est une expression de la forme $w + \Sigma_i \alpha_i p_i + \beta$, $\mathsf{s} + \Sigma_i \alpha_i p_i + \beta$ ou $\Sigma_i \alpha_i p_i + \beta$, où w est une variable discrète, les p_i sont des paramètres et α_i, β sont des constantes dans $\mathbb{Z}$. Avec la convention usuelle, un ensemble vide d'indices correspond à un terme sans paramètre.

Une *contrainte convexe* est une conjonction d'inégalités (strictes ou larges) entre des termes. Une *p-contrainte* est une disjonction de contraintes convexes. Une *mise à jour* est une conjonction d'inégalités entre une variable et un terme, notée $w' \bowtie term$, où w' est une copie primée de la variable discrète w, le symbole $\bowtie$ appartient à $\{<, \leq, =, \geq, >\}$ et $term$ est un terme paramétré. Par convention, $x' = x$ est implicite si x' n'apparaît pas dans la mise à jour.

On note $v \models \varphi$ lorsqu'une valuation v satisfait une p-contrainte φ et $(v, v') \models \theta$ lorsque les valuations v et v' satisfont une mise à jour θ.

États et transitions. Un p-automate comporte un ensemble L d'états (de contrôle), et un ensemble de transitions. Une contrainte convexe appelée *invariant* est associée à chaque état. La contrainte vaut vrai par défaut, et doit être satisfaite pour que, lors d'une exécution, il soit possible de rester dans l'état correspondant. Une transition est de la forme $\langle \ell, \varphi, a, \theta, \ell' \rangle$, où a est l'action étiquetant la transition, ℓ est l'état de départ, ℓ' l'état d'arrivée, φ est une contrainte convexe appelée *garde* et θ une mise à jour.

Exécutions. Une configuration est une paire (ℓ, v) comportant un état ℓ et une valuation v. Les exécutions d'un p-automate sont décrites par un système de transitions dont les états sont les configurations. Deux types de transitions sont possibles :

- Une *action* $(\ell, v) \xrightarrow{a} (\ell', v')$ utilise une transition de la forme $\langle \ell, \varphi, a, \theta, \ell' \rangle$, si v et v' satisfont les invariants de ℓ et de ℓ' respectivement, et si de plus $v \models \varphi$ et $(v, v') \models \theta$. Les variables discrètes sont modifiées et l'automate passe de l'état de contrôle ℓ à ℓ'. La transition d'action est instantanée : la valeur $\mathtt{s}$ de l'horloge est inchangée.
- Un *delai* $(\ell, v) \xrightarrow{t} (\ell, v')$ correspond au passage de t unités de temps dans un état de contrôle ℓ. Il est possible si l'invariant reste satisfait et v' est obtenu à partir de v en conservant les valeurs de toutes les variables sauf l'horloge : $\mathtt{s}' = \mathtt{s} + t$.

2.2 Les c-automates

Nous donnons une rapide présentation des automates à chronomètres, ou *c-automates*, utilisés dans le model-checker compositionnel HCMC [CL00], afin de pouvoir montrer par la suite comment s'applique la méthode du quotient. La principale différence entre les p-automates et les c-automates réside dans l'ensemble V des variables manipulées. Pour ces derniers, il n'y a qu'un type de variable, les chronomètres, dont l'évolution en fonction du temps se fait selon deux modes possibles : soit comme une variable discrète (pente 0), soit comme une horloge (pente 1). Cette évolution dépend de l'état de contrôle de l'automate : on dispose donc d'une fonction $Act : L \mapsto \{0,1\}^V$ qui associe à chaque état ℓ un vecteur $d \in \{0,1\}^V$, décrivant la pente des variables dans cet état. Le système de transition correspondant est obtenu en généralisant celui d'un p-automate :

- Les contraintes sont celles qui interviennent habituellement dans les automates temporisés : $x \bowtie m$ ou $x - y \bowtie m$, où x et y sont des chronomètres et m une constante dans $\mathbb{Z}$. Les chronomètres peuvent être mis à jour lors d'une transition d'action, mais ces mises à jour sont restreintes à des formules du type : $x' = y + m$ ou $x' = m$.
- un délai $(\ell, v) \xrightarrow{t} (\ell, v')$ est possible si l'invariant est satisfait pour v et v', avec $v' = v + Act(\ell) \cdot t$.

Les c-automates sont des cas particuliers d'automates hybrides linéaires, pour lesquels les pentes ne sont pas restreintes à $\{0,1\}$.

2.3 Synchronisation.

Nous présentons un modèle un peu plus général pour la synchronisation, nécessaire pour l'application du quotient. Pour n automates $H_1, \ldots, H_n$, sur un alphabet A, une fonction de synchronisation f est une fonction partielle de $(A \cup \{\tau\})^n \hookrightarrow A$ où τ est un symbole supplémentaire destiné à représenter l'absence d'action (ou une action interne) pour un composant dans un mouvement global du système. La fonction f utilise donc le renommage et on note $(H_1 | \ldots | H_n)_f$ la composition parallèle selon f. Les configurations de $(H_1 | \ldots | H_n)_f$ sont de la forme $(\overline{\ell}, v)$ où $\overline{\ell} = (\ell_1, \ldots, \ell_n)$ est un n-uplet d'états de contrôle (un par composant) et $v = (v_1, \ldots, v_n)$ une valuation de l'ensemble des variables : en supposant disjoints les ensembles de variables des composants, v_i peut être considérée comme la restriction de v aux variables de H_i. Les transitions globales sont alors définies par :

- $(\overline{\ell}, v) \xrightarrow{b} (\overline{\ell'}, v')$ s'il existe un n-uplet d'actions $(a_1, \ldots, a_n) \in (A \cup \{\tau\})^n$ tel que $f(a_1, \ldots, a_n) = b$ et, pour tout i,
  - si $a_i = \tau$, alors $\ell'_i = \ell_i$ et $v'_i = v_i$,
  - si $a_i \in A$, alors $(\ell_i, v_i) \xrightarrow{a_i} (\ell'_i, v'_i)$.
- $(\overline{\ell}, v) \xrightarrow{t} (\overline{\ell}, v')$ si pour tout i, $(\ell_i, v_i) \xrightarrow{t} (\ell_i, v'_i)$.

3 La logique L_ν^h

Ce formalisme [CL00], qui étend la logique L_ν définie dans [LL95], permet de spécifier des propriétés des automates hybrides linéaires. Elle comporte essentiellement un plus grand point fixe, ce qui restreint la spécification aux propriétés de sûreté. Elle permet toutefois d'exprimer des propriétés de vivacité bornée, comme par exemple des temps de réponse.

3.1 Définitions

Syntaxe. Pour un ensemble fini K de variables (appelées variables de formule), un ensemble Id d'identificateurs et un alphabet d'actions A, les formules de L_ν^h sont définies inductivement par :

$$\varphi, \psi \quad ::= \quad \varphi \vee \psi \mid \varphi \wedge \psi \mid \langle a\rangle\varphi \mid [a]\varphi \mid \langle\delta_d\rangle\varphi \mid [\delta_d]\varphi \mid \ \mathsf{Z} \mid \alpha \ \mathsf{in} \ \varphi \mid \gamma$$

où $a \in A$, α est une mise à jour de variables dans K, γ est une p-contrainte sur K, $d \in \mathbb{Z}^K$ et $Z \in \mathsf{Id}$.

Semantique. Les formules de L_ν^h sont interprétées sur des triplets (ℓ, v, u) où (ℓ, v) est une configuration d'un automate H (qui peut lui-même résulter d'une composition parallèle) et u est une valuation des variables de K.

La sémantique est définie de façon usuelle pour les opérateurs vrai et faux, ainsi que pour l'implication lorsqu'il n'y a pas d'identificateurs dans φ (par exemple, $c \Rightarrow \varphi$ est possible pour une contrainte convexe c sur les variables de K) puisque le fragment est clos par négation. La sémantique des identificateurs est donnée par une déclaration $\mathcal{D} : \mathsf{Id} \to L_\nu^h$.

Intuitivement, $\langle a\rangle$ (resp. $[a]$) représentent la quantification existentielle (resp. universelle) sur des transitions avec l'action a, $\langle\delta_d\rangle$ (resp. $[\delta_d]$) représentent la quantification existentielle (resp. universelle) sur des transitions de délai, pour une pente $d \in \mathbb{Z}^K$ des variables de K. Notons que pour l'action interne τ, on adopte la convention suivante : $\langle\tau\rangle\varphi \equiv [\tau]\varphi \equiv \varphi$. La formule $\alpha \ \mathsf{in} \ \varphi$ signifie qu'après une modification des valeurs selon la mise à jour α, l'état obtenu satisfait φ.

La contrainte γ sur des variables de K est satisfaite par (ℓ, v, u) si $u \models \gamma$. Finalement, Z est satisfait si (ℓ, v, u) appartient à la plus grande solution de l'équation $\mathsf{Z} = \mathcal{D}(\mathsf{Z})$. Les formules du type $x := 0 \ \mathsf{in} \ \varphi$ sont abrégées en $x \ \mathsf{in} \ \varphi$. De plus, lorsqu'une formule ne contient aucune variable de formule (dans K), on écrit $\langle\delta\rangle$ (resp. $[\delta]$) au lieu de $\langle\delta_\emptyset\rangle$ (resp. $[\delta_\emptyset]$). La sémantique est résumée dans la table ci-dessous.

$(\ell, v, u) \models_\mathcal{D} \varphi \wedge \phi$	ssi	$(\ell, v, u) \models_\mathcal{D} \varphi$ et $(\ell, v, u) \models_\mathcal{D} \phi$
$(\ell, v, u) \models_\mathcal{D} \varphi \vee \phi$	ssi	$(\ell, v, u) \models_\mathcal{D} \varphi$ ou $(\ell, v, u) \models_\mathcal{D} \phi$
$(\ell, v, u) \models_\mathcal{D} \langle a\rangle\varphi$	ssi	$\exists(\ell', v', u)\ t.q.\ (\ell, v) \xrightarrow{a} (\ell', v')$ et $(\ell', v', u) \models_\mathcal{D} \varphi$
$(\ell, v, u) \models_\mathcal{D} [a]\varphi$	ssi	$\forall(\ell', v', u), (\ell, v) \xrightarrow{a} (\ell', v')$ implies $(\ell', v', u) \models_\mathcal{D} \varphi$
$(\ell, v, u) \models_\mathcal{D} \langle\delta_d\rangle\varphi$	ssi	$\exists t \in \mathbb{R}^{\geq 0}\ t.q.\ (\ell, v) \xrightarrow{t} (\ell, v')$ et $(\ell, v', u + d.t) \models_\mathcal{D} \varphi$
$(\ell, v, u) \models_\mathcal{D} [\delta_d]\varphi$	ssi	$\forall t \in \mathbb{R}^{\geq 0}, (\ell, v) \xrightarrow{t} (\ell, v')$ implies $(\ell, v', u + d.t) \models_\mathcal{D} \varphi$
$(\ell, v, u) \models_\mathcal{D} \gamma$	ssi	$u \models \gamma$
$(\ell, v, u) \models_\mathcal{D} \alpha \ \mathsf{in} \ \varphi$	ssi	$(\ell, v, u') \models_\mathcal{D} \varphi$ avec $(u, u') \models \alpha$
$(\ell, v, u) \models_\mathcal{D} \mathsf{Z}$	ssi	(ℓ, v, u) appartient à une solution maximale de $\mathsf{Z} = \mathcal{D}(\mathsf{Z})$

TAB. 1 – Semantique de la logique modale L_ν^h

3.2 Exemples

Comme L_ν, la logique L_ν^h permet d'exprimer des opérateurs classiques. Nous en présentons quelques uns ci-dessous.

- Pour exprimer qu'une action `error` (dans A) n'est jamais exécutée, on peut utiliser l'équation suivante :

$$X \stackrel{def}{=} \bigwedge_{a \in A} [a]X \wedge [\mathtt{error}]\mathsf{faux} \wedge [\delta]X$$

souvent abrégée en $\mathsf{ALWAYS}_A([\mathtt{error}]\mathsf{faux})$. Dans cet exemple, il n'y a aucune variable de formule et l'opérateur $[\delta]$ concerne seulement les transitions de délais.

- Pour exprimer qu'il y a toujours moins de 100 transitions d'action a le long de tout chemin, on utilise la formule $x := 0$ in X où X est définie par :

$$X \stackrel{def}{=} (x \leq 100) \wedge \bigwedge_{b \in A \setminus \{a\}} [b]X \wedge [a](x := x{+}1 \ \mathsf{in}\ X) \wedge [\delta_{\{\dot{x}=0\}}]X$$

Dans ce cas, x est une variable de formule avec pente nulle, c'est-à-dire une variable discrète.

- Plus généralement, la formule $\mathsf{ALWAYS}_{B,E}(\varphi)$ avec $B \subseteq A$ et $E \subseteq \mathbb{Z}^K$ (un ensemble fini de pentes pour les variables de K), exprime que φ est satisfaite par tout état accessible par des transitions d'actions dans B et de délais avec des pentes dans E pour les variables de K. Cette formule est définie par :

$$X \stackrel{def}{=} \varphi \wedge \bigwedge_{b \in B} [b]X \wedge \bigwedge_{e \in E} ([\delta_e]X$$

- De la même façon, l'operateur *weak until* : $\varphi_1 \mathsf{Until}_{B,E}\, \varphi_2$ est défini comme plus grand point fixe de :

$$X \stackrel{def}{=} \varphi_2 \vee \big(\varphi_1 \wedge \bigwedge_{b \in B} [b]X \wedge \bigwedge_{e \in E} [\delta_e]X\big)$$

- Enfin, comme pour les automates temporisés, l'équation suivante interprétée sur un produit de deux automates (sans invariants [1]) H_1 et H_2 avec la fonction de synchronisation $f(a,\tau) = a_1$ et $f(\tau,a) = a_2$ pour tout $a \in A$, exprime la bisimilarité (forte) de H_1 et H_2 :

$$X \stackrel{def}{=} \bigwedge_{a \in A} [a_1]\langle a_2 \rangle X \wedge \bigwedge_{a \in A} [a_2]\langle a_1 \rangle X \wedge [\delta]X$$

4 Vérification compositionnelle pour L_ν^h

4.1 Cas des automates hybrides linéaires

Considérons maintenant un automate H muni d'un état initial ℓ_0 et d'une valuation initiale v_0 des variables de V. Pour une formule φ de L_ν^h avec des variables dans K, avec u_0 la valuation $\{0\}^K$ (toutes les variables de formule valent 0), et une déclaration $\mathcal{D}$, on définit :

$$H \models_{\mathcal{D}} \varphi \text{ si } (\ell_0, v_0, u_0) \models_{\mathcal{D}} \varphi.$$

Le résultat principal [CL00], obtenu pour des automates hybrides linéaires, peut alors s'énoncer de la façon suivante :

Theorème. Soient H et H' deux automates hybrides linéaires, f une fonction de synchronisation et φ une formule de L_ν^h avec des variables dans un ensemble K. Alors il existe une formule $\varphi/_f H$ et une déclaration $\mathcal{D}'$ telles que

$$(H \mid H')_f \models_{\mathcal{D}} \varphi \text{ ssi } H' \models_{\mathcal{D}'} (\varphi/_f H)$$

[1] Le cas d'automates avec invariants est obtenu par une formule un peu plus compliquée, suivant la même idée.

Ce résultat impose plusieurs remarques. Tout d'abord, lorsqu'on veut itérer l'opération à partir d'un système $S = (H_1 \mid \cdots \mid H_n)_f$, il est plus facile de transformer la synchronisation en utilisant une succession de fonctions binaires, pour obtenir le système sous la forme : $S = (\cdots(H_1|H_2)_{f_1} \cdots |H_n)_{f_{n-1}}$. On obtient alors une généralisation du théorème précédent à partir d'une formule φ et d'une déclaration $\mathcal{D}$ par :

$$S \models_{\mathcal{D}} \varphi \;\;\Leftrightarrow\;\; nil \models_{\mathcal{D}'} (\varphi/_{f_{n-1}} H_n \cdots /_{f_0} H_1)$$

La deuxième remarque concerne le fait que le quotient peut être calculé seulement à partir d'une configuration initiale du système S, ce qui pose un problème pour les paramètres d'un p-automate. En effet, la configuration initiale d'un p-automate est généralement donnée par une p-contrainte, qui fait intervenir les paramètres sans valeurs particulières. C'est principalement pour cette raison que le résultat ci-dessus ne peut pas être appliqué directement aux p-automates.

Enfin, dans la pratique, l'outil HCMC applique le résultat seulement sur des c-automates, ce qui impose des restrictions sur la forme des gardes et des mises à jour.

4.2 Application aux p-automates

Une première méthode possible consiste à transformer un p-automate en c-automate : le codage remplace les paramètres par des chronomètres, dont les valeurs initiales pourront être fixées, comme c'est le cas pour les autres variables. Pour cela, notons $\overline{p} = (p_1, \ldots, p_k)$ le k-uplet de chronomètres qui simulent les paramètres du système. Nous introduisons alors un module supplémentaire dans le système (fig. 1), où l'état initial original du système est $Init$, tandis que $\overline{Init}$ est l'état de calcul des valeurs des paramètres. Les pentes seront à 1 dans cet état et à 0 partout ailleurs. Les durées passées dans l'état entre les remises à 0 déterminent donc des valeurs arbitraires mais positives des paramètres. La transition vers $Init$ teste ensuite la condition initiale $\mathcal{C}(\overline{p})$ sur l'ensemble des paramètres. Cette transition réalise également les initialisations du système : elle doit par exemple remettre l'horloge s à zéro.

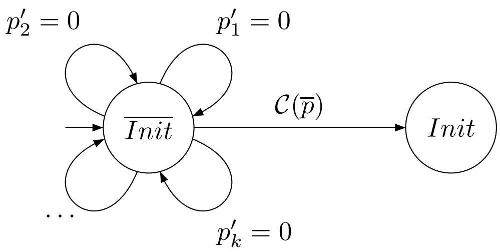

FIG. 1 – Initialisation des paramètres

Une seconde méthode considère les paramètres comme des variables de formule. Dans ce cas, on peut conserver le module ci-dessus en supprimant les boucles sur l'état $\overline{Init}$, et c'est la propriété originale à vérifier, φ, qui est transformée. En notant π_i le vecteur de pentes défini par $\dot{p_i} = 1$ et $\dot{p_j} = 0$ pour $j \neq i$, la nouvelle formule est :

$$[\delta_{\pi_1}][\delta_{\pi_2}] \ldots [\delta_{\pi_k}](\mathcal{C}(\overline{p}) \Rightarrow \varphi)$$

Elle exprime bien que pour toutes les valeurs possibles des paramètres, la contrainte initiale $\mathcal{C}(\overline{p})$ implique φ.

Notons que le problème des contraintes subsiste et nous allons voir dans l'exemple suivant que certains paramètres ne peuvent pas être conservés pour cette raison.

4.3 L'exemple de l'ABR

Rappelons que le système qui modélise l'ABR [BFKM03] contient trois automates : l'environnement, le calcul idéal paramétré et le calcul approché. Ce système comporte *a priori* trois paramètres : la date t d'arrivée d'une cellule de donnée, date à laquelle la propriété doit être vérifiée, et deux valeurs τ_2 et τ_3, qui reflètent des caractéristiques du réseau. De plus, de nombreuses variables discrètes interviennent pour décrire les calculs réalisés sur les débits. Ces variables doivent donc être décrites par des chronomètres dont les pentes sont nulles en permanence, sauf en ce qui concerne le taux $\mathtt{R}$ d'une nouvelle cellule RM. En effet, dans ce cas, la mise à jour $\mathtt{R}' > 0$ doit être réalisée comme pour un paramètre. Enfin, l'observation des gardes et des mises à jour fait apparaître des formules du type $t \leq \mathtt{s} + \tau_3$ ou bien $\mathtt{tfi}' = \mathtt{s} + \tau_2$ (où $\mathtt{s}$ est l'horloge et $\mathtt{tfi}$ une variable discrète), ce qui n'est pas possible dans un c-automate. Le paramètre t étant un élément crucial dans cette approche, la solution consiste à considérer τ_2 et τ_3 comme des constantes.

Le modèle de l'environnement devient donc le suivant :

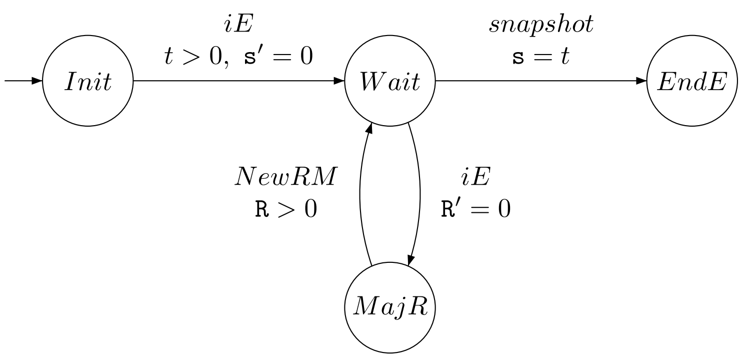

FIG. 2 – c-automate de l'environnement

L'action iE est une action interne de l'environnement, correspondant aux calculs de t et de $\mathtt{R}$. La pente est 1 pour t dans l'état $Init$ mais 0 partout ailleurs. Le dédoublement de l'état $Wait$ est nécessaire pour calculer une nouvelle valeur de $\mathtt{R}$. Dans cet état, il faut donc prendre une pente 1 pour $\mathtt{R}$ (qui vaudra 0 partout ailleurs) et 0 pour l'horloge $\mathtt{s}$, afin de simuler un calcul instantané. Ce dédoublement permet de simplifier les deux autres automates, qui se synchronisent avec l'environnement, en particulier sur l'action $NewRM$.

Ainsi, en notant [I1], [I2a], [I2b], [I3] les différentes opérations (dont le détail est donné en annexe), l'automate de l'algorithme idéal, qui calcule la valeur $\mathtt{E}$ du débit admissible, devient :

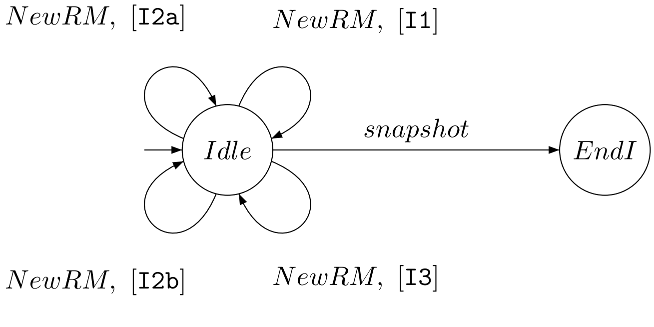

FIG. 3 – c-automate de l'algorithme idéal $\mathcal{I}$

De même, en notant [1], ..., [8], [9a], [9b] les opérations (données en annexe), et iB une action interne, l'algorithme de calcul d'une approximation A pour le débit admissible peut être modélisé par l'automate de la figure 4.

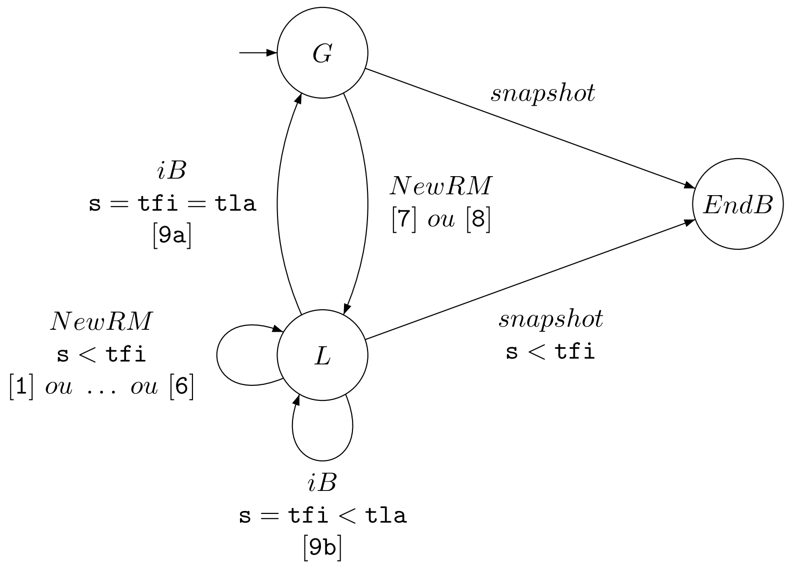

FIG. 4 – c-automate de l'approximation

La propriété cherchée exprime qu'un utilisateur n'est pas désavantagé, c'est-à-dire que, lorsqu'une donnée doit être envoyée, on a toujours $\mathtt{A} \geq \mathtt{E}$. Dans le modèle, la date d'émission d'une donnée est le paramètre t, ce qui signifie que la propriété doit être vérifiée lorsque l'horloge s atteint la valeur t. Ceci correspond à l'état global $(EndE, EndI, EndB)$, obtenu après une action *snapshot*. Dans la logique L_ν^h, cette propriété s'exprime par la formule $\varphi : [\delta][iE]X$, où X est définie par la déclaration $\mathcal{D}$:

$$X \stackrel{def}{=} [snapshot](A <= E) \wedge [iE]X \wedge [NewRM]X \wedge [iB]X \wedge [\delta]X$$

5 Conclusion

Le codage de l'ABR et de la propriété souhaitée dans HCMC sont donnés en annexe, de même que les résultats obtenus. Ces résultats ne sont pas satisfaisants actuellement car HCMC doit faire une surapproximation dans le calcul de point fixe La prochaine étape de ce travail consiste donc à étendre HCMC avec un calcul de point fixe utilisant une bibliothèque de polyèdres. Ceci fournira un semi-algorithme proche de celui qui est effectué par HYTECH, sans surapproximation. le calcul obtenu sera donc exact dans le cas où la terminaison se produit, le calcul obtenu sera exact. Ce calcul constituera en même temps une extension de HYTECH, dans la mesure où l'application à L_ν^h permettra de vérifier des propriétés comportementales plus intéressantes que les seules propriétés d'accessibilité disponibles dans HYTECH.

Références

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, Pei-Hsin Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1) :3–34, 1995.

[ACHH93] R. Alur, C. Courcoubetis, T. A. Henzinger, and Pei-Hsin Ho. Hybrid automata : an algorithmic approach to the specification and verification of hybrid systems. In *Proc. Workshop Hybrid Systems, Lyngby, DK, Oct. 1992*, volume 736 of *Lecture Notes in Computer Science*, pages 209–229. Springer, 1993.

[AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2) :183–235, 1994.

[And95] H. R. Andersen. Partial model checking (extended abstract). In *Proc. 10th IEEE Symp. Logic in Computer Science (LICS'95), San Diego, CA, USA, June 1995*, pages 398–407. IEEE Comp. Soc. Press, 1995.

[BFKM03] B. Bérard, L. Fribourg, F. Klay, and J.-F. Monin. A compared study of two correctness proofs for the standardized algorithm of ABR conformance. *Formal Methods in System Design*, 22(1) :59–86, 2003.

[CL00] F. Cassez and F. Laroussinie. Model-checking for hybrid systems by quotienting and constraints solving. In *Proc. 12th Int. Conf. Computer Aided Verification (CAV'2000), Chicago, IL, USA, July 2000*, volume 1855 of *Lecture Notes in Computer Science*, pages 373–388. Springer, 2000.

[HHW95] T. A. Henzinger, Pei-Hsin Ho, and H. Wong-Toi. A user guide to HyTech. In *Proc. 1st Int. Workshop Tools and Algorithms for the Construction and Analysis of Systems (TACAS'95), Aarhus, DK, May 1995*, volume 1019 of *Lecture Notes in Computer Science*, pages 41–71. Springer, 1995.

[LL95] F. Laroussinie and K. G. Larsen. Compositional model-checking of real time systems. In *Proc. 6th Int. Conf. Theory of Concurrency (CONCUR'95), Philadelphia, PA, USA, Aug. 1995*, volume 962 of *Lecture Notes in Computer Science*, pages 529–539. Springer, 1995.

A Annexe

Opérations associées à l'algorithme $\mathcal{I}$

[I1] si t >= s+τ_2 alors E'= R
[I2a] si s+τ_3 <= t < s+τ_2 et E < R alors E'= R
[I2b] si s+τ_3 <= t < s+τ_2 et E >= R alors E'= E
[I3] si t < s+τ_3 alors E'= E

Opérations associées à l'approximation

[1a] si s < tfi et Emx <= R et tfi < s+τ_3 et s+τ_3 < tla alors
    Emx' = R ; Ela' = R ; tla' = s+τ_3
[1b] si s < tfi et Emx <= R et tfi < s+τ_3 et tfi= tla alors
    Emx' = R ; Ela' = R ; tla' = s+τ_3
[2] si s < tfi et Emx <= R et tfi < tla et tla <= s+τ_3 alors
    Emx' = R ; Ela' = R
[3] si s < tfi et Emx <= R et tfi >= s+τ_3 et A <= R alors
    Emx' = R ; Ela' = R ; Efi' = R ; tfi' = s+τ_3 ; tla' = s+τ_3
[4] si s < tfi et Emx <= R et tfi >= s+τ_3 et A > R alors
    Emx' = R ; Ela' = R ; Efi' = R ; tla' = tfi
[5] si s < tfi et Emx > R et R < Ela alors
    Ela' = R ; Efi' = Emx ; tla' = s+τ_2
[6] si s < tfi et Emx > R et R >= Ela alors
    Ela' = R, Efi' = Emx
[7] si s >= tfi et A <= R alors
    Ela' = R, Efi' = R, Emx'= R, tfi'= s+τ_3, tla'= s+τ_3
[8] si s >= tfi et A > R alors
    Ela' = R, Efi' = R, Emx' = R, tfi' = s+τ_2, tla' = s+τ_2

Code HCMC

```
a := 10;
b := 20;

Automata : Env [
Clocks : s, R, t
Nodes : init, wait, maj, end
Transitions :
            init - ie, {s:=0} -> wait,
            wait - ie, {R:=0} -> maj,
            maj - NewRM, R>0 -> wait,
            wait - snaps, s=t -> end
Invariants :
wait |= s <= t
Null Activities :
wait : R t,
maj  : t s
];

Automata : AI [
Clocks : s, R, t, E
Nodes : idle, end
Transitions :
```

```
            idle  - NewRM, t<s+%a -> idle,
            idle  - NewRM, t>=s+%a ^ t< s+%b ^ E >= R -> idle,
            idle  - NewRM, t>=s+%a ^ t< s+%b ^ E < R, {E := R} -> idle,
            idle  - NewRM, t>=s+%b, {E := R} -> idle,
            idle - snaps -> end
Null Activities :
idle : E
];

Automata : B [
Clocks : s, R, t, A, tfi, tla, Efi, Ela, Emx
Nodes : greater, less, end
Transitions :
            greater - NewRM, s>=tfi ^ A <= R,
{Ela := R, Efi := R, Emx := R, tfi := s + %a, tla := s + %a} -> less,
            greater - NewRM, s>=tfi ^ A > R,
{Ela := R, Efi := R, Emx := R, tfi := s + %b, tla := s + %b} -> less,
            less - NewRM, s<tfi ^ Emx <= R ^ tfi < s+%a ^ (tfi = tla v tla > s+%a),
{Emx:=R, Ela := R, tla := s+%a} ->less,
            less - NewRM, s<tfi ^ Emx <= R ^ tfi < tla ^ tla <= s+%a,
{Emx:=R, Ela := R} -> less,
            less - NewRM, s<tfi ^ Emx <= R ^ tfi >= s+%a ^ A <= R,
{Emx := R, Ela :=R, Efi := R, tfi := s+%a, tla := s + %a} -> less,
            less - NewRM, s<tfi ^ Emx <= R ^ tfi >= s+%a ^ A > R,
{Emx := R, Ela :=R, Efi := R, tla := tfi} -> less,
            less - NewRM, s<tfi ^ Emx > R ^ R < Ela,
{Ela := R, Efi := Emx, tla := s+%b} -> less,
            less - NewRM, s<tfi ^ Emx > R ^ R >= Ela,
{Ela := R, Efi := Emx} -> less,
            less - i, s=tfi ^ tfi<tla,
{A:=Efi, tfi := tla, Efi:=Ela, Emx:=Ela} -> less,
            less - i, s=tfi^tfi=tla,
{A:=Efi, tfi := tla, Efi:=Ela, Emx:=Ela} -> greater,
            less - snaps, s<tfi -> end,
            greater - snaps -> end
Null Activities :
greater : A tfi tla Efi Ela Emx,
less : A tfi tla Efi Ela Emx
];

Table : T  {
        (Env,   AI,     B) ;
        (ie,    .,      .) -> ie;
        (snaps, snaps, snaps) -> snaps;
        (NewRM,NewRM,NewRM) -> NewRM;
        (.,     .,      i) -> iB
};

System : S (Env [1 1 0 1 0], AI, B [1 1 1 1 0]);

Declaration : Dsafe
 [ X = [snaps] (A <= E) ^ [ie] X ^ [NewRM] X ^ [iB] X ^ [@] X
];
```

```
  Fsafe := ([@][ie] X, Dsafe);
  Verify(Fsafe,S,T);
```

Résultats de la vérification

```
 Verification : S |= Fsafe ?

Quotient of Env (~SP0)
hitzone is put to 0
unguardedvar is put to 1
With the binary synchronization function :
Synchro. function :
iB -> {(iB,_) }
NewRM -> {(~a1,NewRM) }
snaps -> {(~a0,snaps) }
ie -> {(_,ie) }

With the synchronization table of the subsystem :
Synchronization table :  (2)
Parallel composition : ( AI | B )
.       i       --> iB
NewRM   NewRM   --> ~a1
snaps   snaps   --> ~a0
2 variables in the declaration before the simplifications.
---------------------------------------------------------

Specification :
[@] s in X0
 Declaration :     [
 X0 = ([iB]X0  ^ (s=t=>[~a0]A<=E) ^ R in X1  ^ [@\t,R] (s<=t=>X0 )) ;
 X1 = ([iB]X1  ^ ((s<=t ^ R>0)=>[~a1]X0 ) ^ [@\s,t] X1 )
];
2 identifiers.

0 equations have been reduced to tt or ff by boolean simplifications.
0 equivalent equations have been removed.
0 trivial equations have been removed.
0 tests have been removed by the Hzones.

2 in the final declaration.

Quotient of AI (~SP1)
With the binary synchronization function :
Synchro. function :
~a0 -> {(snaps,snaps) }
~a1 -> {(NewRM,NewRM) }
iB -> {(i,_) }

With the synchronization table of the subsystem :
Synchronization table :  (1)
Parallel composition : ( B )
snaps   --> snaps
NewRM   --> NewRM
i       --> i
```

```
3 variables in the declaration before the simplifications.
---------------------------------------------------------

Specification :
[@\E] s in X0
 Declaration :     [
 X0 = ([@\t,R,E] (s<=t=>X0 ) ^ R in X1  ^ (s=t=>[snaps]A<=E) ^ [i]X0 ) ;
 X1 = ([@\s,t,E] X1  ^ ((R>0 ^ s<=t)=>X2 ) ^ [i]X1 ) ;
 X2 = ((t<s+10=>[NewRM]X0 ) ^ ((t<s+20 ^ t>=s+10 ^ E>=R)=>[NewRM]X0 )
^ ((t<s+20 ^ t>=s+10 ^ E<R)=>[NewRM]E:=R in X0 )
^ (t>=s+20=>[NewRM]E:=R in X0 ))
];
3 identifiers.

0 equations have been reduced to tt or ff by boolean simplifications.
0 equivalent equations have been removed.
0 trivial equations have been removed.
0 tests have been removed by the Hzones.

3 in the final declaration.

Quotient of B (~SP2)
hitzone is put to 1
With the binary synchronization function :
Synchro. function :
i -> {(_,i) }
NewRM -> {(_,NewRM) }
snaps -> {(_,snaps) }

With the synchronization table of the subsystem :
Synchronization table :  (0)
Parallel composition : (  )
26 variables in the declaration before the simplifications.
---------------------------------------------------------

Specification :
[@\E,Emx,Ela,Efi,tla,tfi,A] s in X0
 Declaration :     [
 X0 = ((s=t=>A<=E) ^ R in X1  ^ [@\t,R,E,Emx,Ela,Efi,tla,tfi,A] (s<=t=>X0 )) ;
 X1 = ((R>0=>X2 ) ^ [@\s,t,E,Emx,Ela,Efi,tla,tfi,A] X1 ) ;
 X2 = ((t>=s+20=>X4 ) ^ ((t<s+20 ^ t>=s+10 ^ E<R)=>X4 )
^ ((t<s+20 ^ t>=s+10 ^ E>=R)=>X3 ) ^ (t<s+10=>X3 )) ;
 X3 = ((A<=R=>Ela:=R in Efi:=R in Emx:=R in tfi:=s+10 in tla:=s+10 in X5 )
^ (A>R=>Ela:=R in Efi:=R in Emx:=R in tfi:=s+20 in tla:=s+20 in X5 )) ;
 X4 = ((A<=R=>Ela:=R in Efi:=R in Emx:=R in tfi:=s+10 in tla:=s+10 in E:=R in X5 )
^ (A>R=>Ela:=R in Efi:=R in Emx:=R in tfi:=s+20 in tla:=s+20 in E:=R in X5 )) ;
 X5 = (((tfi=tla ^ s=tfi)=>A:=Efi in tfi:=tla in Efi:=Ela in Emx:=Ela in X0 )
^ ((tfi<tla ^ s=tfi)=>A:=Efi in tfi:=tla in Efi:=Ela in Emx:=Ela in X5 )
^ ((s=t ^ s<tfi)=>A<=E) ^ R in X6  ^ [@\t,R,E,Emx,Ela,Efi,tla,tfi,A] (s<=t=>X5 )) ;
 X6 = (((tfi=tla ^ s=tfi)=>A:=Efi in tfi:=tla in Efi:=Ela in Emx:=Ela in X1 )
^ ((tfi<tla ^ s=tfi)=>A:=Efi in tfi:=tla in Efi:=Ela in Emx:=Ela in X6 )
^ (R>0=>X7 ) ^ [@\s,t,E,Emx,Ela,Efi,tla,tfi,A] X6 ) ;
 X7 = ((t>=s+20=>X9 ) ^ ((t<s+20 ^ t>=s+10 ^ E<R)=>X9 )
^ ((t<s+20 ^ t>=s+10 ^ E>=R)=>X8 ) ^ (t<s+10=>X8 )) ;
```

```
 X8 = (X13  ^ ((tla<=s+10 ^ Emx<=R ^ s<tfi ^ tfi<tla)=>Emx:=R in Ela:=R in X5 )
^ ((A<=R ^ Emx<=R ^ s<tfi ^ tfi>=s+10)
=>Emx:=R in Ela:=R in Efi:=R in tfi:=s+10 in tla:=s+10 in X5 )
^ ((A>R ^ Emx<=R ^ s<tfi ^ tfi>=s+10)
=>Emx:=R in Ela:=R in Efi:=R in tla:=tfi in X5 )
^ ((R<Ela ^ s<tfi ^ Emx>R)=>Ela:=R in Efi:=Emx in tla:=s+20 in X5 )
^ ((R>=Ela ^ s<tfi ^ Emx>R)=>Ela:=R in Efi:=Emx in X5 )) ;
 X9 = (X10  ^ ((tla<=s+10 ^ Emx<=R ^ s<tfi ^ tfi<tla)
=>Emx:=R in Ela:=R in E:=R in X5 )
^ ((A<=R ^ Emx<=R ^ s<tfi ^ tfi>=s+10)
=>Emx:=R in Ela:=R in Efi:=R in tfi:=s+10 in tla:=s+10 in E:=R in X5 )
^ ((A>R ^ Emx<=R ^ s<tfi ^ tfi>=s+10)
=>Emx:=R in Ela:=R in Efi:=R in tla:=tfi in E:=R in X5 )
^ ((R<Ela ^ s<tfi ^ Emx>R)=>Ela:=R in Efi:=Emx in tla:=s+20 in E:=R in X5 )
^ ((R>=Ela ^ s<tfi ^ Emx>R)=>Ela:=R in Efi:=Emx in E:=R in X5 )) ;
 X10 = (tfi>=s+10 v s>=tfi v Emx>R v X11
v Emx:=R in Ela:=R in tla:=s+10 in E:=R in X5 ) ;
 X11 = (tla<=s+10 ^ X12 ) ;
 X12 = (tfi>tla v tfi<tla) ;
 X13 = (tfi>=s+10 v s>=tfi v Emx>R v X11  v Emx:=R in Ela:=R in tla:=s+10 in X5 )
];
14 identifiers.

0 equations have been reduced to tt or ff by boolean simplifications.
12 equivalent equations have been removed.
0 trivial equations have been removed.
6 tests have been removed by the Hzones.

14 in the final declaration.

Check procedure for not(spec)
 Fixpoint computation  (46 var.) :
it: 1-  (0)
it: 2-  (0) , Av. size : 0 , Max size : 1
it: 3-  (41) , Av. size : 0 , Max size : 2
it: 4-  (65) , Av. size : 0 , Max size : 2
it: 5-  (101) , Av. size : 0 , Max size : 7
it: 6-  (118) , Av. size : 1 , Max size : 7
it: 7-  (155) , Av. size : 1 , Max size : 7
it: 8-  (172) , Av. size : 2 , Max size : 7
it: 9-  (209) , Av. size : 2 , Max size : 16
it: 10-  (223) , Av. size : 3 , Max size : 16
it: 11-  (259) , Av. size : 4 , Max size : 31
it: 12-  (274) , Av. size : 4 , Max size : 31
it: 13-  (312) , Av. size : 5 , Max size : 31
it: 14-  (334) , Av. size : 5 , Max size : 34
it: 15-  (356) , Av. size : 6 , Max size : 47
it: 16-  (373) , Av. size : 6 , Max size : 37
it: 17-  (388) , Av. size : 7 , Max size : 36
it: 18-  (401) , Av. size : 7 , Max size : 41
it: 19-  (414) , Av. size : 8 , Max size : 42
it: 20-  (427) , Av. size : 8 , Max size : 44
it: 21-  (440) , Av. size : 8 , Max size : 48
it: 22-  (454) , Av. size : 10 , Max size : 58
```

```
it: 23-  (465) , Av. size : 10 , Max size : 55
it: 24-  (475) , Av. size : 11 , Max size : 57
it: 25-  (483) , Av. size : 11 , Max size : 65
it: 26-  (491) , Av. size : 11 , Max size : 71
it: 27-  (500) , Av. size : 12 , Max size : 62
it: 28-  (509) , Av. size : 13 , Max size : 64
it: 29-  (519) , Av. size : 13 , Max size : 66

 v0 belongs to the set [[- spec]] but an overapproximation
 has been done: no conclusion can be done.
 Approx:
 - under-approx -> 0
 - over-approx  -> 1
 - __abstraction-> 0
 __test = 281
t <=10  ;  v
0< A  ;  v
E-A<0  ;
 Results table:
2       3       26
2       3       14
 -
```