Vérification de modèles temporisés

(mémoire d'habilitation à diriger les recherches)

B. Bérard

Laboratoire Spécification et Vérification
ENS de Cachan & CNRS UMR 8643

Table des matières

1 Introduction

Ce mémoire présente brièvement une partie de mes travaux les plus récents, consacrés à la vérification des systèmes réactifs. De tels systèmes sont constitués de programmes destinés à maintenir une interaction avec un environnement et qui, de ce fait, revêtent un aspect dynamique important. Pour un programme fermé, contenant explicitement son environnement, on cherche à vérifier que, sous certaines conditions initiales, il se termine en produisant un résultat correct. Au contraire, un système réactif est soumis à des contraintes concernant le déroulement de ses exécutions possibles.

Cette présentation est centrée sur les modèles dits *temporisés* : ces modèles décrivent une sous-classe des systèmes réactifs, dans laquelle la notion de temps est manipulée de manière explicite, de sorte qu'il soit possible d'exprimer des conditions critiques sous forme de contraintes quantitatives sur des délais. Par exemple, on souhaite vérifier que, chaque fois qu'une situation dangereuse se produit, une alarme se déclenche moins de trente secondes plus tard. Nous avons choisi de ne pas parler des travaux réalisés dans [BLS00, BBFM01] qui, bien qu'ils soient également liés à la vérification de systèmes réactifs, n'entrent pas directement dans le cadre temporisé. Ce document comporte trois parties.

1. La première (chapitres 2 et 3) expose de manière succincte la problématique de modélisation de systèmes temporisés. Nous rappelons rapidement les différentes approches qui ont permis de construire de tels modèles à partir des modèles classiques où le temps est abstrait. Nous proposons ensuite un modèle général, fondé sur les systèmes hybrides et esquissé dans [Bér95], qui permet de présenter dans un cadre unifié une grande partie des travaux existants sur les systèmes temporisés. Nous indiquons finalement les questions qui se posent pour ces modèles ainsi que certains résultats maintenant bien connus.
2. Dans la deuxième partie (chapitres 4 et 5), nous décrivons nos contributions, tirées de [BDGP98, BD00, BP00], aux aspects théoriques des modèles temporisés. Nous nous intéressons essentiellement à des questions de décidabilité et d'expressivité pour des classes particulières de tels modèles. En effet, dans le cadre des systèmes hybrides, la plupart des problèmes liés à la vérification sont indécidables. A partir de modèles plus simples, pour lesquels certains résultats de décidabilité existent, nous examinons la persistance de ces propriétés pour différents types d'extensions.
3. La troisième partie (chapitres 6 et 7) concerne les travaux exposés dans [BFKM01, BF99], qui sont davantage orientés vers les applications. Nous présentons en particulier des études de cas qui, par les résultats expérimentaux obtenus, mettent en évidence la puissance des méthodes de vérification destinées spécifiquement aux modèles temporisés. Ces méthodes portent sur deux catégories de modèles, les p-automates et les réseaux de Petri, ce dernier point illustrant une utilisation remarquable d'un outil «temps-réel» pour la vérification de propriétés sur des entiers.

2 Les modèles temporisés

Une modélisation est qualifiée de *temporisée* lorsqu'elle fait intervenir une notion explicite de temps, ce qui permet d'exprimer des contraintes quantitatives. Avant de décrire les différentes approches de construction des modèles temporisés, nous examinons les représentations du temps.

2.1 Domaines de temps et sémantiques temporisées

La représentation du temps. Les domaines traditionnellement utilisés pour le temps sont les ensembles $\mathbb{N}$, $\mathbb{Q}_+$ et $\mathbb{R}_+$. Le premier est un cas particulier de domaine *discret*, parfois modélisé aussi par des séquences d'événements appelés *ticks*, lors desquels il est possible d'observer les états du système. Les deux autres représentent un temps *dense*. De façon plus générale, [NSY93] propose un modèle unifié du domaine de temps sous la forme d'un monoïde commutatif $(\mathbb{T}, +)$, où l'élément neutre est noté 0, vérifiant les deux propriétés suivantes :

- $\forall t, t' \in \mathbb{T},\ t + t' = t \Leftrightarrow t' = 0$
- la relation $\leq$ définie par $t \leq t'$ si $\exists t'' \in \mathbb{T}$ tel que $t' = t + t''$, est un ordre total sur $\mathbb{T}$.

Cette définition implique que 0 est le plus petit élément de $\mathbb{T}$ et que si $t \leq t'$, alors il existe un unique $t'' \in \mathbb{T}$, noté $t' - t$, tel que $t' = t + t''$.
Ainsi, les éléments de $\mathbb{T}$ représentent des dates et la différence entre deux dates correspond naturellement à une durée, également dans $\mathbb{T}$. Il est possible dans ce cadre de formaliser les notions de domaine dense ou discret.
Une *suite de dates* est une suite croissante d'éléments de $\mathbb{T}$. Lorsque le domaine est $\mathbb{Q}_+$ ou $\mathbb{R}_+$, une telle suite peut être convergente. Les comportements associés sont alors qualifiés de convergents ou *zénoniens*, en référence au fameux paradoxe de Zénon d'Élée. Dans la suite, pour des raisons de simplicité, le domaine de temps $\mathbb{T}$ est plongé dans $\mathbb{R}$.

Systèmes de transitions temporisés. Les modèles temporisés ont une sémantique à base de systèmes de transitions étiquetés. Rappelons qu'un tel système est défini par un triplet $\mathcal{T} = (L, S, E)$, où L est l'ensemble des étiquettes, S est l'ensemble des états et la relation de transition E est un sous-ensemble de $S \times L \times S$. On note généralement $\xrightarrow{\ell}$ l'ensemble des transitions d'étiquette ℓ et $s \xrightarrow{\ell} s'$ une transition $(s, \ell, s') \in E$. Pour un sous-ensemble R de S et un sous-ensemble K de L on définit :

- $Post_K(R) = \{s' \in S \mid \exists s \in R, \exists \ell \in K\ s \xrightarrow{\ell} s'\}$,
- les itérés pour $n \in \mathbb{N}$
  $Post_K^n(R) = \{s' \in S \mid$ il existe un chemin de longueur n d'un $s \in R$ à s' avec des étiquettes dans $K\}$,
- et $Post_K^*(R) = \cup_{n \geq 0} Post_K^n(R)$.

Un *système de transitions temporisé* est un système de transitions $\mathcal{T}$ étiqueté par $\Sigma \cup \mathbb{T}$, où Σ est un alphabet fini et $\mathbb{T}$ un domaine de temps. Les transitions $\xrightarrow{a}$, $a \in \Sigma$, correspondent à des actions au sens usuel et sont considérées comme instantanées. Dans certaines variantes, on utilise $\Sigma \cup \{\varepsilon\}$ au lieu de Σ, le mot vide ε représentant une action non observable. Les autres transitions $\xrightarrow{d}$, $d \in \mathbb{T}$, expriment l'écoulement d'une durée et doivent de ce fait vérifier des conditions particulières, qui traduisent la compatibilité du système par rapport aux opérations sur le temps :

- $s \xrightarrow{0} s$
- si $s \xrightarrow{d} s'$ et $s' \xrightarrow{d'} s''$, alors $s \xrightarrow{d+d'} s''$

Le système $\mathcal{T}$ est *déterministe par rapport au temps* si $s \xrightarrow{d} s_1$ et $s \xrightarrow{d} s_2$ implique $s_1 = s_2$.

Pour un système de transitions temporisé, les notions de successeurs par rapport au temps prennent un sens particulier. Pour cette raison, on définit la relation plus abstraite $\leq$, appelée *t-successeur*, par $s \leq s'$ s'il existe $d \in \mathbb{T}$ tel que $s \xrightarrow{d} s'$, et on étend la fonction $Post$ par : $Post_t(R) = \{s' \mid \exists s \in R, s \leq s'\}$.

Une autre sémantique possible consiste à considérer un seul type de transitions, avec pour étiquettes des paires $(a, d) \in \Sigma \times \mathbb{T}$. Cette approche permet alors d'interpréter d comme la durée de a. La relation correspondante peut être définie à partir de $\mathcal{T}$ en combinant une transition de délai $\xrightarrow{d}$ et une transition d'action $\xrightarrow{a}$:

$s \overset{a,d}{\Rightarrow} s'$ s'il existe $s'' \in S$ tel que $s \xrightarrow{d} s''$ et $s'' \xrightarrow{a} s'$.

L'ordre des deux transitions $\xrightarrow{d}$ et $\xrightarrow{a}$ est parfois interverti. Le système de transitions étiqueté $\mathcal{T}_m = (\Sigma \times \mathbb{T}, S, \Rightarrow)$ est appelé *système de transitions mixte* associé à $\mathcal{T}$, comme dans [Lab98]. C'est cette sémantique qui est utilisée dans [AD90], où elle est définie directement à partir du modèle de machine.

Notre travail est orienté vers les relations entre construction et vérification des modèles. Nous utilisons donc les systèmes de transitions temporisés sans aborder en détail les questions de sémantique, qui sont largement étudiées dans le cadre des algèbres de processus. On pourra consulter [BM01] pour une présentation des algèbres de processus temporisés.

Langages temporisés. Un *mot temporisé* est une paire (σ, τ), où σ est le mot $a_1 a_2 \dots$ sur l'alphabet Σ et $\tau = t_1 t_2 \dots$ est une suite de dates de même longueur que σ.
A partir de l'étiquette $w = (a_i, d_i)_{i \geq 1}$ dans $(\Sigma \times \mathbb{T})^\infty$ d'un chemin dans un système de transitions mixte $\mathcal{T}_m$, on obtient un mot temporisé $obs(w) = (a_i, t_i)_{i \geq 1}$ en cumulant les délais : $t_i = \sum_{j \leq i} d_j$, pour $i \geq 1$. Dans ce cadre, une paire (a_i, t_i) est appelée une *action temporisée*, puisqu'elle peut représenter une observation du système à la date t_i, date à laquelle se produit instantanément l'action a_i. Réciproquement, à partir d'un mot temporisé, on obtient une séquence de paires $(a_i, d_i) \in \Sigma \times \mathbb{T}$, où $d_i = t_i - t_{i-1}$ peut être considéré comme la durée maximale de a_i.

Lorsque les étiquettes sont dans $\Sigma \cup \{\varepsilon\} \cup \mathbb{T}$, il faut de plus supprimer les actions (a_i, t_i) non observables, c'est-à-dire celles pour lesquelles $a_i = \varepsilon$). Notons que, dans ce dernier cas, l'opération de calcul des dates à partir des délais doit nécessairement précéder l'effacement des actions non observables, pour que les délais associés à ces dernières soient pris en compte.

Un *langage temporisé* est un ensemble de mots temporisés. On peut se ramener au cadre classique des mots non temporisés par la projection $Untime(L) = \{\sigma \mid \exists \tau\ (\sigma, \tau) \in L\}$ d'un langage temporisé L. Les langages temporisés ont été peu étudiés indépendamment des systèmes de transitions mixtes. Quelques travaux [ACM97, HRS98, BP99, BP01] ont néanmoins abordé la question des expressions régulières dans le cadre temporisé, et des approches algébriques ont été proposées dans [GL95, Dim00, BPT01].

2.2 Construction de modèles temporisés

Temporiser un modèle consiste à utiliser une représentation explicite du temps pour lui associer des contraintes temporelles quantitatives. Un premier type d'approche est fondé sur la remarque que les observations d'un système ont toujours lieu selon des séquences discrètes. Avec une granularité assez fine, il suffirait alors de déplier le modèle initial pour en obtenir un autre, dans lequel toutes les unités de temps sont apparentes. L'exemple ci-dessous montre une utilisation du domaine de temps $\mathbb{N}$ qui correspond à un prolongement des modèles classiques.

Un exemple avec des dates entières. Considérons la spécification suivante d'un système où trois actions a, b et c sont possibles : «toute occurrence de l'action a est suivie dans le futur par une occurrence de l'action b». Nous obtenons dans ce cas très simple l'automate fini $\mathcal{A}_1$ de la figure 1.

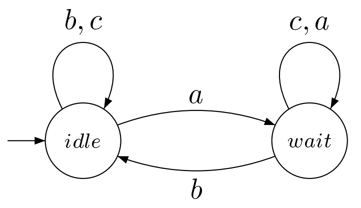


FIG. 1 – $\mathcal{A}_1$: toute action a sera suivie d'un b

Ajoutons maintenant une contrainte de temps, par exemple : «toute occurrence de l'action a est suivie d'une occurrence de l'action b dans un délai inférieur ou égal à 3 unités de temps». En introduisant dans le système l'action supplémentaire #, qui représente l'action vide (ou une action interne), déplier l'automate précédent fournit l'automate «temporisé» $\mathcal{A}_2$ de la figure 2 dans lequel, implicitement, l'intervalle entre deux actions (instantanées) est d'une unité de temps.

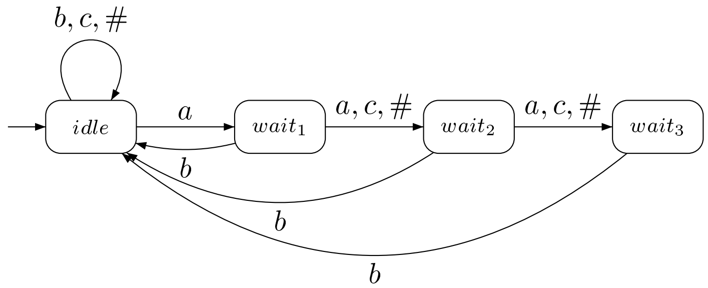


FIG. 2 – $\mathcal{A}_2$: toute action a est suivie d'un b après au plus 3 unités de temps

Dans ce cadre, les suites de dates sont de la forme $(t_i)_{i\geq 0}$, avec $t_i = i$ pour tout i. On peut aussi modifier légèrement les modèles de façon à autoriser plusieurs actions (généralement un nombre fini) simultanées. On obtiendrait alors des suites de dates «avec répétitions».
Un inconvénient de ce type de modèles est la difficulté d'y associer un traitement symbolique du temps et des contraintes.

Temporisation des transitions. Une autre approche consiste à *temporiser les transitions* : à partir d'un modèle non temporisé avec un ensemble Q d'états, appelés *modes de contrôle*, on munit d'une contrainte chacune des transitions ou *changements de mode* $e : q \xrightarrow{a} q'$, pour $q, q' \in Q$, et on définit ensuite un système de transitions temporisé. Deux méthodes ont principalement été proposées pour ce type de construction.

- Historiquement, la première a consisté à représenter une contrainte temporelle élémentaire comme un intervalle du domaine de temps. On associe donc à $e : q \xrightarrow{a} q'$ un intervalle (min_e, max_e), où $min_e \leq max_e$ dans $\mathbb{T}$ représentent respectivement le délai minimal et le délai maximal autorisés pour exécuter l'action a dans le mode q. C'est cette approche qui a été utilisée pour la temporisation des réseaux de Petri dans [JLL77], ou pour les systèmes temporisés de [MMP91, HMP94]. En particulier, [HMP92] applique cette méthode avec le domaine de temps $\mathbb{N}$ et propose une discussion sur les mérites comparés des modèles avec temps dense ou temps discret.
- La seconde méthode, plus récente, utilise des variables pour décrire certains aspects du système [Dil89, AH97]. Ces variables sont considérées comme des fonctions du temps dans les modes de contrôle, et elles peuvent être modifiées de manière discrète lors des changements de modes. La classe très générale des systèmes hybrides est obtenue de cette façon, avec des variables à valeurs réelles. On en trouve une présentation synthétique pour les cas où le contrôle est fini dans [Hen96].

2.3 Les systèmes hybrides linéaires

Nous décrivons maintenant une sous-classe des systèmes hybrides que nous qualifions ici de «linéaire» : elle contient les systèmes qui manipulent des variables à valeurs réelles, dont la variation est linéaire par morceaux en fonction du temps. Notons que cette appellation concerne usuellement une classe plus large, dans laquelle l'évolution des variables est caractérisée par une équation différentielle linéaire [AHLP00]. L'exposé qui suit reprend des éléments maintenant classiques, mais en utilisant toutefois un cadre un peu plus abstrait, que nous avons esquissé dans [Bér95]. L'originalité de cette approche réside dans une distinction explicite entre les notions liées au temps et celles qui ont trait au contrôle, le système complet étant obtenu comme un produit synchronisé de deux systèmes de transitions :

- le premier décrit uniquement les aspects temporisés du système, c'est-à-dire les évolutions possibles des variables en fonction de temps,
- le second consiste en un système non temporisé classique, qui décrit le contrôle des variables.

Contraintes linéaires. Soit X un ensemble fini de variables à valeurs dans (un sous-ensemble de) $\mathbb{R}$. Un *terme linéaire* est une expression de la forme $k_0 + k_1x_1 + \cdots + k_mx_m$, où les x_i sont des variables de X et les k_i des constantes dans (un sous-ensemble de) $\mathbb{R}$. Si ℓ_1 et ℓ_2 sont des termes linéaires, $\ell_1 \leq \ell_2$ est une *inégalité linéaire*. Une *contrainte linéaire* est une combinaison booléenne d'inégalités linéaires. L'ensemble des contraintes linéaires sur X est noté $\mathcal{C}(X)$.
Une *valuation* est une application $v : X \to \mathbb{R}$, qui peut aussi être vue comme un n-uplet $\mathbf{x}$ de valeurs réelles, où $n = |X|$ est le cardinal de X. Une valuation $\mathbf{x}$ satisfait une contrainte $\varphi \in \mathcal{C}(X)$, noté $\mathbf{x} \models \varphi$, si le prédicat fermé $\varphi[X := \mathbf{x}]$ est vrai. On note $[\![\varphi]\!]$ l'ensemble des valuations satisfaisant φ, les deux notations étant parfois confondues.

Systèmes de transitions linéaires. Nous construisons un système de transitions dit *linéaire* $\mathcal{L} = (M, V, \Delta)$, dont les états sont précisément les valuations des variables de X, c'est-à-dire $V = \mathbb{R}^X$. Le cardinal n de X est appelé la *dimension* du système. L'ensemble des étiquettes est de la forme $M = J \cup (\Pi \times \mathbb{T})$, où J et Π sont deux ensembles finis :

- un élément j de J caractérise un *saut* (ou *jump*), autrement dit un changement discret de valuation. Il est décrit par une contrainte linéaire dans $\mathcal{C}(X \cup X')$, où $X' = \{x' \mid x \in X\}$ est une copie de X qui désigne les nouvelles valeurs des variables après ce changement.
- un élément π de Π caractérise l'évolution des variables en fonction du temps. C'est donc cette fois une contrainte linéaire dans $\mathcal{C}(\dot{X})$, où $\dot{X} = \{\dot{x} \mid x \in X\}$ est une autre copie de X représentant les *pentes* des variables de X.

Les transitions de Δ sont des deux types suivants :

- celles qui décrivent un changement discret produit par $j \in J$:
  $\mathbf{x} \xrightarrow{j} \mathbf{x}'$ si le prédicat fermé $j[X, X' := \mathbf{x}, \mathbf{x}']$ est vrai.
- celles qui représentent l'écoulement d'une durée $d \in \mathbb{T}$ avec une pente $\mathbf{p}$ produite par $\pi \in \Pi$:
  $\mathbf{x} \xrightarrow{\pi,d} \mathbf{x}'$ s'il existe un vecteur $\mathbf{p} \in \mathbb{R}^n$ tel que $\pi[\dot{X} := \mathbf{p}]$ est vrai et $\mathbf{x}' = \mathbf{x} + \mathbf{p}.d$, où $+$ et $.$ désignent respectivement l'addition de vecteurs et la multiplication par un réel (composante par composante).

En posant $\mathbf{x} \xrightarrow{d} \mathbf{x}'$ s'il existe $\pi \in \Pi$ tel que $\mathbf{x} \xrightarrow{\pi,d} \mathbf{x}'$, on définit sur le même ensemble d'états V, avec $J \cup \mathbb{T}$ comme ensemble d'étiquettes, un système de transitions temporisé (au sens du paragraphe 2.1), généralement non déterministe.

Remarquons que la notation $j[X, X' := \mathbf{x}, \mathbf{x}']$ signale explicitement le fait que le remplacement des variables par leurs valeurs dans le prédicat j s'effectue de façon instantanée. Une écriture de la forme $j[X := \mathbf{x}, X' := \mathbf{x}']$ pourrait laisser supposer que les deux groupes d'affectations sont séquentiels.

Contrôle. Un système hybride linéaire est un ensemble de variables linéaires, manipulées par un contrôle qui définit les sauts et les pentes, c'est-à-dire les étiquettes du système. Ce contrôle est généralement modélisé par un graphe $\mathcal{G} = (X, \Sigma, Q, E, \pi)$ dont à la fois les sommets et les arêtes sont étiquetés :

- X est un ensemble de variables linéaires et Σ un alphabet fini d'actions,
- Q est l'ensemble des sommets, appelés *modes de contrôle*,
- $E \subseteq Q \times (\Sigma \times \mathcal{C}(X \cup X')) \times Q$ est un ensemble fini de changements de mode,
- π est une application de Q dans $\mathcal{C}(\dot{X})$.

Dans certains contextes, les sommets de $\mathcal{G}$ peuvent être munis d'autres étiquettes comme des propositions atomiques ou des contraintes, souvent dans un sous-ensemble de $\mathcal{C}(X)$.

Notons que rien n'impose *a priori* le choix d'un contrôle fini, même si une telle hypothèse laisse espérer de bons résultats pour l'analyse des systèmes. Dans certains travaux [BER94, GS94, Bér95], le contrôle est réalisé par le système de transition d'une grammaire algébrique ou d'un réseau de Petri. On parlera d'automate dans le cas d'un contrôle fini. Ainsi, un automate hybride linéaire est précisément un système de transitions linéaire couplé avec un contrôle fini.

Sémantique des systèmes hybrides linéaires. On considère maintenant le produit synchronisé de $\mathcal{G}$ avec $\mathcal{L}$: les états sont des paires de $Q \times V$ et les transitions sont de la forme

- $(q, \mathbf{x}) \xrightarrow{a,j} (q', \mathbf{x}')$ s'il existe un changement de modes $(q, a, j, q') \in E$ avec $\mathbf{x} \xrightarrow{j} \mathbf{x}'$,

- $(q, \mathbf{x}) \xrightarrow{\pi(q),d} (q, \mathbf{x}')$ si $\mathbf{x} \xrightarrow{\pi(q),d} \mathbf{x}'$.

La sémantique d'un système hybride linéaire est alors simplement le système de transitions temporisé obtenu par projection :

- $(q, \mathbf{x}) \xrightarrow{a} (q', \mathbf{x}')$ s'il existe un changement de modes $(q, a, j, q') \in E$ avec $\mathbf{x} \xrightarrow{j} \mathbf{x}'$,
- $(q, \mathbf{x}) \xrightarrow{d} (q, \mathbf{x}')$ s'il existe $\pi \in \Pi$ tel que $\mathbf{x} \xrightarrow{\pi,d} \mathbf{x}'$.

Le système de transitions mixte associé est à nouveau obtenu par composition d'une transition de délai $\xrightarrow{d}$ suivie d'une transition d'action $\xrightarrow{a}$:

$(q, \mathbf{x}) \overset{a,d}{\Rightarrow} (q', \mathbf{x}')$ s'il existe $\mathbf{x}''$ tel que $(q, \mathbf{x}) \xrightarrow{d} (q, \mathbf{x}'')$ et $(q, \mathbf{x}'') \xrightarrow{a} (q', \mathbf{x}')$.

2.4 Exemples et extensions

En raison de leur simplicité, les systèmes linéaires au sens ci-dessus sont implicitement présents dans de nombreuses modélisations temporisées.

Automates temporisés. Pour les automates temporisés, les variables sont toutes des *horloges*, c'est-à-dire des variables de pente 1. Une restriction supplémentaire provient de la forme des sauts : un élément $j \in J$ est de la forme $\varphi \wedge \psi$, où

- la contrainte φ est une *garde* dans un sous-ensemble $C \subseteq \mathcal{C}(X)$ de contraintes linéaires sur X,
- la contrainte ψ est une *remise à zéro* (ou *reset*) : une conjonction de contraintes atomiques $x' = 0$, pour une horloge x.

Les ensembles (finis) C de gardes sont généralement choisis dans des sous-classes de $\mathcal{C}(X)$ obtenues par des combinaisons booléennes de contraintes atomiques particulières.

- $\mathcal{C}_0(X)$ correspond à des contraintes atomiques *rectangulaires* : $x \bowtie c$, pour une horloge x, une constante rationnelle c et des opérateurs $\bowtie \in \{<, =, >\}$.
- $\mathcal{C}_1(X)$ est la sous classe associée à des contraintes atomiques qui sont soit rectangulaires, soit *triangulaires* : $x - y \bowtie c$, pour deux horloges x, y, une constante c et $\bowtie \in \{<, =, >\}$. Notons que l'appellation originale *automate temporisé* introduite dans [AD90, AD94] correspond à cette sous classe.
- Les classes $\mathcal{C}_{i+}(X)$ sont obtenues à partir des $\mathcal{C}_i(X)$ en ajoutant des contraintes atomiques additives : $x + y \bowtie c$, pour deux horloges x, y, une constante c et $\bowtie \in \{<, =, >\}$.

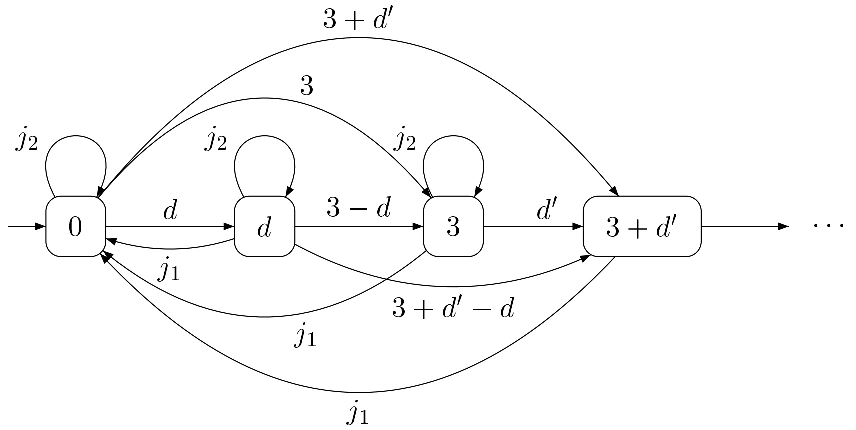


FIG. 3 – Un système de transitions linéaire $\mathcal{L}$ pour une horloge et deux sauts

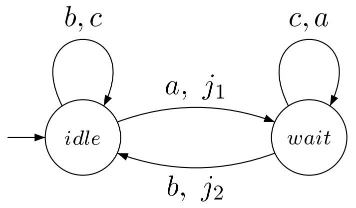


FIG. 4 – $\mathcal{A}_3$: toute action a est suivie d'un b après au plus 3 unités de temps

Ce modèle permet par exemple d'exprimer de façon naturelle la spécification «toute occurrence de l'action a est suivie d'une occurrence de l'action b dans un délai inférieur ou égal à 3 unités de temps». On introduit pour cela une horloge x qui peut être remise à zéro par le saut $j_1 :\ x' = 0$, et comparée à 3 par $j_2 :\ x \leq 3$. Le système linéaire associé est représenté schématiquement sur la figure 3 : il contient une infinité non dénombrable d'états lorsque le domaine de temps est $\mathbb{R}$. De plus, l'unique prédicat $\pi : \dot{x} = 1$ a été omis sur les transitions de temps, où ne figurent que les délais. Le contrôle est défini par l'automate temporisé $\mathcal{A}_3$ de la figure 4, qui est simplement $\mathcal{A}_1$, décoré par les sauts. Les systèmes $\mathcal{T}$ et $\mathcal{T}_m$ associés au produit $\mathcal{A}_3 \times \mathcal{L}$ sont représentés, à nouveau de manière très abrégée, dans les figures 5 et 6. Par exemple, tous les états de la forme $(idle, d)$, $d \geq 0$ ont été fusionnés et toutes les transitions de temps sont étiquetées par d. Ces systèmes peuvent être comparés à l'automate $\mathcal{A}_2$ de la figure 2 obtenu avec un temps discret.

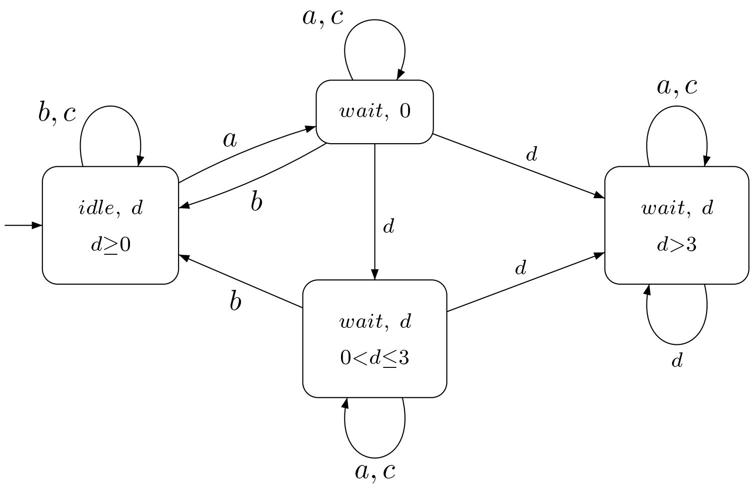


FIG. 5 – Une représentation simplifiée du système temporisé $\mathcal{T}$

Dans ces deux systèmes, rien n'empêche *a priori* de rester indéfiniment dans une suite d'états $(wait,\ t_i)$, pour une suite quelconque $(t_i)_{i \geq 1}$. Ce problème a conduit à incorporer dans les modèles des conditions pour assurer la progression du temps ou l'absence de blocage. Différentes solutions ont été proposées, parmi lesquelles on peut citer l'introduction de conditions de Büchi, présentes dans le modèle original de [AD90], mais aussi l'ajout d'*invariants*, qui sont des contraintes linéaires associées aux modes de contrôle [NSY92]. Les relations entre diverses conditions de vivacité sont examinées en détail dans [BS97, Lab98].

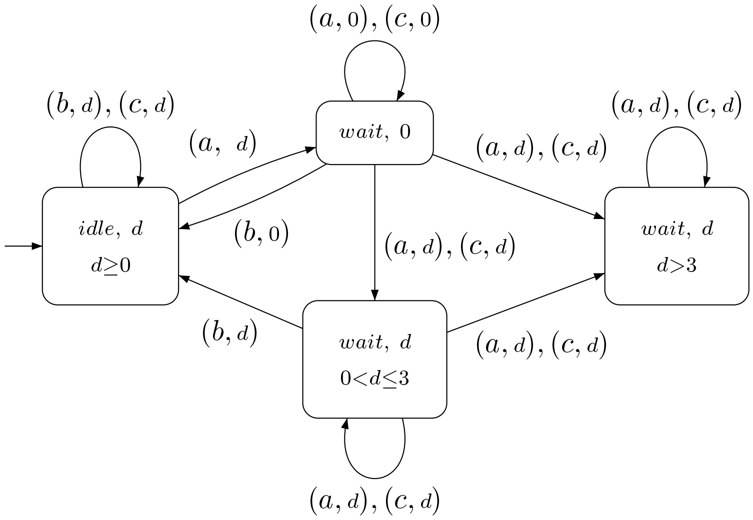


FIG. 6 – Une représentation simplifiée du système mixte $\mathcal{T}_m$

Extensions pour le domaine de temps $\mathbb{R}$. A partir des travaux d'Alur et Dill, plusieurs extensions sont apparues.

Les *automates rectangulaires* [HK97] utilisent des ensembles $\mathcal{I}$ d'intervalles de $\mathbb{R}$ dont les bornes sont rationnelles ou infinies. L'ensemble Π décrivant les pentes des variables contient un unique prédicat de la forme $\pi = \wedge_{x \in X}(\dot{x} \in I_x)$, pour des intervalles I_x de $\mathcal{I}$. D'autre part, un saut $j \in J$ est encore de la forme $\varphi \wedge \psi$, où

- La garde $\varphi \in \mathcal{C}_0(X)$ est exprimée comme une conjonction de contraintes atomiques $x \in I$, pour une horloge x et un intervalle I de $\mathcal{I}$,
- La contrainte ψ généralise les opérations de *reset* par une conjonction de contraintes atomiques $x' \in I$ ou $x' = x$, pour une horloge x et un intervalle I de $\mathcal{I}$.

D'autres généralisations des remises à zéro ont été proposées dans [BDFP00a, BDFP00b], avec une contrainte ψ de la forme $x' \bowtie x + c$ ou $x' \bowtie y + c$.

Des systèmes linéaires particuliers sont étudiés dans [DZ98, KPSY99]. Le mécanisme introduit dans [DZ98] est celui des *cadres* (*frames*). De manière schématique, un cadre produit des pentes qui vérifient : pour chaque vecteur $\mathbf{p} = (p_1, \ldots, p_n)$, il existe un rationnel c tel que pour tout $i = 1, \ldots, n$, $p_i = 0$ ou $p_i = c$.

Nous terminons par un exemple «non linéaire», extrait de [JLHM91] (et simplifié), qui permet de situer les limites du modèle choisi, en terme de pouvoir d'expression.

L'automate hybride de la figure 7 décrit le fonctionnement d'un thermostat, qui déclenche ou arrête un appareil de chauffage suivant les informations provenant d'un capteur de température. Il comporte deux modes de contrôle : `ON` et `OFF`, qui manipulent une variable θ représentant la température. Dans chacun des deux modes, la loi l'évolution de θ en fonction du temps est caractérisée par une équation différentielle. On suppose pour simplifier que le système se trouve initialement dans le mode `ON` avec une température nulle et que les constantes K et θ_h dépendent de l'appareil de chauffage. Dans ce mode, l'appareil est en marche et la température augmente en suivant la loi spécifiée, jusqu'à atteindre une valeur maximale, θ_{max}.

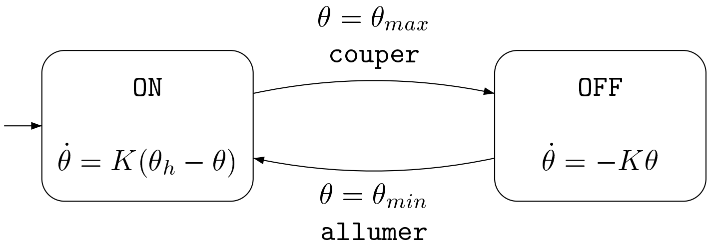


FIG. 7 – modélisation temporisée d'un thermostat

Le système passe alors en mode `OFF`, où l'appareil s'éteint et la température décroît. Lorsque la température vaut θ_{min}, l'appareil se remet en marche. Notons que la variation de θ en fonction du temps correspond à des contraintes dans $\mathcal{C}(X \cup \dot{X})$, c'est-à-dire à des équations différentielles linéaires, comme dans [AHLP00].

2.5 Réseaux de Petri et variables discrètes.

D'autres modèles manipulent uniquement des variables *discrètes*, c'est-à-dire de pente nulle. Dans ce cas, il n'y a pas de transitions décrivant l'écoulement du temps et l'ensemble des étiquettes peut être réduit à $M = J$.

Réseaux de Petri. Nous rappelons brièvement le cadre formel des réseaux de Petri, en montrant comment ils peuvent être interprétés en termes de systèmes linéaires.
Un réseau de Petri est un n-uplet $\mathcal{N} = (W, \Sigma, (\mathbf{g}_a, \mathbf{m}_a)_{a \in \Sigma})$, où W est un ensemble fini de variables entières (les places) et Σ un alphabet fini (d'étiquettes de transitions). À chaque $a \in \Sigma$ sont associés deux vecteurs $\mathbf{g}_a$ et $\mathbf{m}_a$ de $\mathbb{Z}^W$. Une valuation est appelée un *marquage* et identifiée à un vecteur d'entiers naturels. Ainsi, en notant $S = \mathbb{N}^W$ l'ensemble des états, le système de transitions $\mathcal{T} = (\Sigma, S, \Delta)$ associé à un réseau de Petri est appelé *graphe des marquages*, la relation d'accessibilité Δ étant définie par $\bigcup_{a \in \Sigma} \xrightarrow{a}$ où

$$\mathbf{w} \xrightarrow{a} \mathbf{w}' \text{ si } \mathbf{w} \geq \mathbf{g}_a \wedge \mathbf{w}' = \mathbf{w} + \mathbf{m}_a$$

pour les opérations usuelles sur les vecteurs.

Le réseau de Petri représenté sur la figure 8 possède une seule transition a, avec deux places d'entrée w_1, w_2 et trois places de sortie w_2, w_3, w_4. La transition a est franchissable si $w_1 \geq 3$ et $w_2 \geq 1$, avec comme résultat $w'_1 = w_1 - 3$, $w'_3 = w_3 + 2$ et $w'_4 = w_4 + 1$. En considérant les contraintes implicites de la forme $w'_i = w_i$ et $w_i \geq 0$, on a :

$$\mathbf{w} \xrightarrow{a} \mathbf{w}' \text{ si } \mathbf{w} \geq (3, 1, 0, 0) \wedge \mathbf{w}' = \mathbf{w} + (-3, 0, 2, 1).$$

La description ci-dessus évoque naturellement une sous-classe des systèmes linéaires avec des variables de pentes nulles, à valeurs entières. Pour un ensemble W de variables discrètes, les sauts sont des contraintes de la forme $\varphi \wedge \psi$, où φ est une conjonction de contraintes atomiques $w \geq c$, pour $w \in W$ et $c \in \mathbb{N}$, et la mise à jour ψ est une conjonction de contraintes

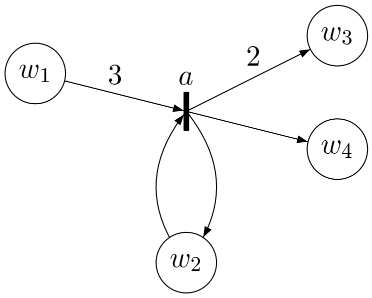


FIG. 8 – Un réseau de Petri à une transition

atomiques $w' = w + c$, où $c \in \mathbb{Z}$. Dans ce contexte, le contrôle peut être considéré comme trivial.

Automates de données. Nous mentionnons pour conclure une extension non linéaire des systèmes à variables discrètes : les *automates de données*, introduits dans [BPT01], manipulent des registres qui sont des variables à valeurs dans un domaine quelconque $\mathbb{D}$, muni d'un élément minimal noté $\perp$. Comme pour les réseaux de Petri, le système correspondant possède un seul type de transitions, étiquetées par des sauts de la forme $\varphi \wedge \psi$, où φ est une garde et ψ une mise à jour. Ces systèmes n'entrent pas non plus dans la catégorie linéaire en raison du type des contraintes autorisées pour les gardes.

3 Problèmes d'analyse

La précision des modèles temporisés, tels qu'ils apparaissent dans le chapitre qui précède, a longtemps semblé un obstacle sérieux lorsqu'il s'agissait de la vérification de propriétés, en raison du nombre infini éventuellement non dénombrable d'états des systèmes associés. Les réponses classiques, dans la ligne décrite au paragraphe 2.2, ont consisté à maintenir une vision abstraite du temps, à la fois en considérant des systèmes essentiellement discrets et en restreignant les contraintes temporelles à des opérateurs concernant l'ordonnancement des événements, dans une distinction du type avant/après. Depuis les travaux d'Alur et Dill [AD90, AD94], les modèles temporisés avec variables ont été largement étudiés. Des outils permettant l'analyse de tels modèles ont été développés, illustrant leur efficacité pratique sur de nombreuses études de cas industrielles. Finalement, des résultats récents montrent qu'intrinsèquement, la complexité théorique de la vérification pour ces modèles n'est pas pire que celle qui résulte de la composition dans le contexte habituel [LS00]. Bien entendu, dans la pratique, l'explosion liée au produit se combine à celle qui résulte de l'existence de variables, ce qui conduit à des difficultés spécifiquement liées aux questions algorithmiques. Nous examinons maintenant différents aspects de la vérification de systèmes temporisés.

3.1 Contexte général

Composition. Remarquons tout d'abord qu'un système de transitions résulte souvent, dans la pratique, d'un produit de plusieurs modèles sous-jacents. C'est évidemment le cas aussi pour les systèmes temporisés. Toutefois, l'introduction de contraintes quantitatives fait apparaître des problèmes spécifiquement liés à la composition. L'utilisation de modèles manipulant des variables a permis de définir (par exemple dans [ACH94, SY96, BS98]) des méthodes robustes pour cette opération.

Dans cette présentation, nous considérons un système donné $\mathcal{T}$, sans faire intervenir la manière dont il a été obtenu. Différentes approches sont possibles pour l'analyse d'un tel système, suivant la nature des propriétés souhaitées et la façon dont celles-ci sont exprimées.

Model-checking. Lorsqu'une propriété est représentée par une formule φ d'une logique, le problème de savoir si $\mathcal{T}$ satisfait φ est celui du model checking. Plusieurs logiques ont été proposées pour l'étude des modèles temporisés ; elles consistent généralement en des extensions de logiques temporelles incorporant des paramètres quantitatifs. Plusieurs équipes ont consacré des efforts importants au model-checking de ces logiques, qui ont conduit, dans les années 1995-1996, au développement d'outils, des «model-checkers» temporisés. Notre exposé n'aborde pas cet aspect algorithmique mais on peut citer [LL98, AK96, HHW97, BDM$^+$98, BLL$^+$98] pour les rapports les plus récents sur CMC,Cospan, HyTech, Kronos et Uppaal. Dès leur apparition comme prototypes, ces outils ont permis de réaliser avec succès des études expérimentales sur des exemples académiques mais aussi sur de très nombreuses applications industrielles.

Comparaison de systèmes. La spécification d'une propriété p peut aussi prendre la forme d'un système de transitions $\mathcal{T}_p$. Dans ce cas, savoir si le système de départ $\mathcal{T}$ satisfait la propriété s'exprime comme une *simulation* : «tout comportement de $\mathcal{T}$ est-il aussi un comportement de $\mathcal{T}_p$?». De multiples relations d'équivalence entre systèmes temporisés ont été examinées dans le cadre des algèbres de processus temporisés [BM01].

Plus faiblement, la satisfaction d'une propriété peut être vue comme une inclusion de langages. En considérant des systèmes mixtes et les mots étiquetant des chemins, on cherche à décider si «tout mot de $\mathcal{T}$ est aussi un mot de $\mathcal{T}_p$». Une question plus simple concerne l'existence d'un chemin dans $\mathcal{T}$ satisfaisant une propriété p. Lorsqu'on peut construire un système $\mathcal{T}_{\neg p}$ contenant exactement les comportements de $\mathcal{T}$ qui ne satisfont pas p, la vérification se ramène à tester le vide de $\mathcal{T}_{\neg p}$. Enfin, les problèmes les plus élémentaires sont liés à l'accessibilité, et s'expriment alors par des propriétés des états. Par exemple, pour un système de transitions $\mathcal{T} = (L, S, E)$, où un sous-ensemble S_0 d'états initiaux est fixé, on cherche à savoir si un état donné s est atteint depuis S_0, c'est-à-dire si $s \in Post^*(S_0)$. À nouveau, si on peut décrire le sous-ensemble N_p de S des états atteints par un chemin ne satisfaisant pas une propriété p, la vérification se ramène à tester le vide de $Post^*(S_0) \cap N_p$. Dans le cas d'un modèle temporisé, on peut aussi chercher si un mode de contrôle q donné est atteint depuis un sous-ensemble S_0 de $Q \times V$, autrement dit s'il existe une valuation $\mathbf{x}$ telle que $(q, \mathbf{x}) \in Post^*(S_0)$.

Pour les systèmes temporisés, ces problèmes ont été étudiés tout au long des dix dernières années, notamment dans [AH92a, AH92b, CY92, OSY94, ACH+95, TY96, Hen96, ABBL98, DT98, HKPV98, KPSY99, Lar00]. Dans ce qui suit, nous nous intéressons plus particulièrement aux questions d'accessibilité et de décidabilité du vide.

3.2 Construction d'un quotient

Une méthode classique pour analyser les systèmes de transitions infinis consiste à en obtenir un quotient que l'on peut traiter, et qui conserve néanmoins de bonnes propriétés. C'est notamment le cas lorsqu'on peut construire un quotient relié par une équivalence au système de départ. Nous allons rappeler rapidement ce principe, qui a permis d'obtenir le premier résultat de décidabilité du vide pour les automates temporisés [AD90]. Toutefois, nous nous plaçons dans le cadre plus général exposé ci-dessus qui, en séparant les hypothèses selon qu'elles concernent le temps ou le contrôle, permet de mettre en évidence des conditions générales d'application.

Considérons un système linéaire $\mathcal{L} = (M, V, \Delta)$, défini comme précédemment. On cherche à construire une partition de l'ensemble V des valuations, telle que la relation d'équivalence $\sim$ associée soit une bisimulation, compatible avec les sauts de J et avec la relation de t-successeur :

- pour tout saut $j \in J$, si $\mathbf{x} \sim \mathbf{y}$ et $\mathbf{x} \xrightarrow{j} \mathbf{x}'$, alors il existe une transition $\mathbf{y} \xrightarrow{j} \mathbf{y}'$ telle que $\mathbf{x}' \sim \mathbf{y}'$.
- si $\mathbf{x} \sim \mathbf{y}$ et $\mathbf{x} \leq \mathbf{x}'$ alors il existe une valuation $\mathbf{y}'$ telle que $\mathbf{y} \leq \mathbf{y}'$ et $\mathbf{x}' \sim \mathbf{y}'$.

Dans le cas particulier où toutes les pentes sont nulles, seule la première condition est pertinente. Supposons qu'un tel quotient $\mathcal{R} = V/\sim$ puisse être construit, nous appelons *région* un élément de $\mathcal{R}$. Les opérations $\xrightarrow{j}$ et $\leq$ (ainsi que les ensembles $Post_j$ et $Post_t$) peuvent alors être définies sur $\mathcal{R}$, de façon à fournir un système $\mathcal{K} = (M, \mathcal{R}, D)$ bisimilaire à $\mathcal{L}$. On obtient à nouveau un système avec contrôle par un produit synchronisé de $\mathcal{K}$ avec le graphe de contrôle $\mathcal{G}$. Les états sont des paires de $Q \times \mathcal{R}$ et les transitions de D sont de la forme :

- $(q, \rho) \xrightarrow{a,j} (q', \rho')$ s'il existe un changement de modes $(q, a, j, q') \in E$ avec $\rho \xrightarrow{j} \rho'$,
- $(q, \rho) \xrightarrow{\varepsilon} (q, \rho')$ si $\rho \leq \rho'$.

La plupart des travaux définissent plutôt des relations d'équivalence directement sur le quotient $Q \times \mathcal{R}$. Par exemple dans [ACH94, Hen95, TY96], des études systématiques d'équivalences sont proposées, pour de larges classes de systèmes hybrides. Toutefois, dans la plupart des cas, ces relations s'expriment naturellement comme un produit, d'une façon similaire à ce qui vient d'être décrit.

En adaptant la définition précédente à la relation $\Rightarrow$ des systèmes de transitions mixtes, on obtient un autre produit avec un seul type de transitions :

$(q, \rho) \xrightarrow{a} (q', \rho')$ s'il existe un t-successeur ρ'' de ρ et un changement de modes $(q, a, j, q') \in E$ avec $\rho'' \xrightarrow{j} \rho'$.

Ce dernier point de vue permet d'obtenir une relation simple entre les langages L et $Untime(L)$, lorsque L est un langage temporisé associé, dans certaines conditions que nous préciserons plus tard, au système de transitions temporisé de $\mathcal{G} \times \mathcal{L}$:

Proposition 3.1 ([Bér95]). *$Untime(L) = pr(G \cap K)$, où G et K sont des langages sur l'alphabet $\Sigma \times J$, étiquetant des chemins dans $\mathcal{G}$ et $\mathcal{K}$ respectivement, et pr est la projection de $(\Sigma \times J)^\infty$ sur Σ^∞.*

On obtient ainsi un schéma général permettant d'obtenir la décidabilité du vide, reposant sur l'équivalence $L = \emptyset$ ssi $Untime(L) = \emptyset$. La proposition 3.1 permet d'obtenir la décidabilité du vide dès que celle-ci peut être déduite des propriétés de $\mathcal{K}$ (s'il existe) et de $\mathcal{G}$. Par exemple, lorsqu'on peut construire un quotient fini $\mathcal{K}$, le système linéaire de départ est bisimilaire à un automate fini. Dans ce cas, on a :

Théorème 3.2 ([Bér95]). *Soit $\mathcal{S}$ une classe de systèmes de transitions temporisés, $\mathcal{F}_\mathcal{S}$ une famille de langages temporisés associée à $\mathcal{S}$ et $\mathcal{U}_\mathcal{S}$ la famille non temporisée correspondante des langages de chemins. Si $\mathcal{U}_\mathcal{S}$ est fermée par projection et intersection rationnelle, alors $Untime(\mathcal{F}_\mathcal{S}) = \mathcal{U}_\mathcal{S}$.*

En conséquence, pour de tels systèmes linéaires, la décidabilité du vide se ramène à des propriétés de la classe de systèmes de transitions à laquelle appartient le contrôle $\mathcal{G}$. Si celui-ci est fini, le produit synchronisé est encore un automate fini, ce qui résoud immédiatement le problème. Le résultat est conservé si $\mathcal{G}$ est dans une classe décidable, possédant des propriétés constructives de fermeture, comme celle des automates à pile ou des réseaux de Petri.

Le cas des automates temporisés de [AD90] mérite une attention particulière, puisqu'il est à l'origine de l'étude des modèles temporisés. Nous rappelons d'abord ses particularités et les principaux résultats obtenus dans ce contexte, puis nous passons en revue quelques extensions intéressantes.

3.3 Cas des automates temporisés

Régions. Nous supposons que le domaine $\mathbb{T}$ est $\mathbb{N}$ ou $\mathbb{Q}$ (ou $\mathbb{R}$). Comme nous l'avons déjà mentionné au chapitre précédent (paragraphe 2.4), les automates temporisés manipulent un ensemble X d'horloges, avec des gardes appartenant généralement aux sous-ensembles $\mathcal{C}_0(X)$ ou $\mathcal{C}_1(X)$ de $\mathcal{C}(X)$, et des remises à zéro.

Dans [AD90], Alur et Dill considèrent des contraintes de $\mathcal{C}_0(X)$, avec des constantes entières. En fixant la constante maximale m apparaissant dans les gardes, ils construisent une partition finie compatible, fondée sur la relation d'équivalence suivante : $v \sim v'$ si

- pour toute horloge x, ou bien les parties entières de $v(x)$ et de $v'(x)$ sont égales, ou bien $v(x) > m$ et $v'(x) > m$
- pour toute horloge x telle que $v(x) \leq m$, $frac(v(x)) = 0$ si et seulement si $frac(v'(x)) = 0$, où $frac(t)$ désigne la partie fractionnaire du réel t,
- pour tout couple (x, y) d'horloges telles que $v(x) \leq m$ et $v'(x) \leq m$, $frac(v(x)) \leq frac(v(y))$ si et seulement si $frac(v'(x)) \leq frac(v'(y))$.

Pour un système de dimension 2, avec $m = 3$, on obtient la partition représentée sur la figure 9. Les régions sont les points à coordonnées entières, les segments ouverts entre deux tels

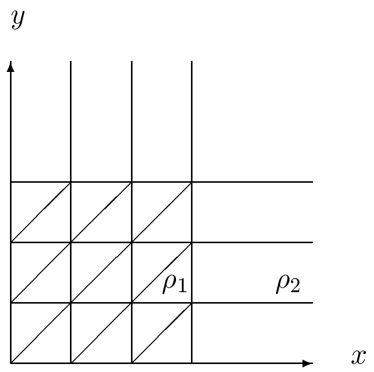


FIG. 9 – Partition de $\mathbb{R}^2_+$ pour $m = 3$

points, les intérieurs des triangles délimités par les segments, les demi-droites ouvertes et les régions ouvertes délimitées par ces demi-droites. Ainsi, les régions notées ρ_1 et ρ_2 sur cette figure peuvent être décrites par : $\rho_1 : (2 < x < 3) \wedge (1 < y < 2) \wedge (0 < frac(y) < frac(x))$ et $\rho_2 : (x > 3) \wedge (1 < y < 2)$.

De façon générale, pour un ensemble C de contraintes, on notera dans la suite $C^m(X)$ la sous-classe de C qui n'utilise que des constantes entières de C bornées par m. La construction ci-dessus s'étend facilement à $\mathcal{C}_1^m(X)$, la partition associée à un système de dimension 2 avec $m = 2$ étant représentée sur la figure 10.

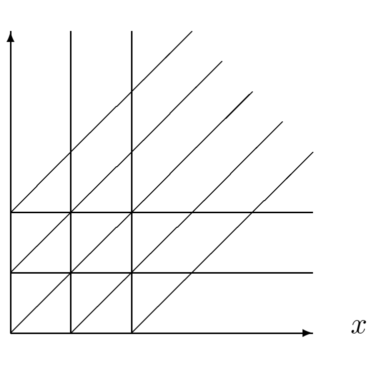


FIG. 10 – Partition de $\mathbb{R}^2$ pour $\mathcal{C}_1^2(X)$

Remarquons que dans le premier cas, certaines régions ne peuvent pas être décrites par une contrainte de $\mathcal{C}_0(X)$. Nous dirons que la partition $\mathcal{R}$ est *fine* si $\mathcal{R} \subseteq C$, c'est-à-dire que toute région s'exprime par une contrainte. La partition obtenue pour $\mathcal{C}_1^m(X)$ est donc fine.

Classes de langages temporisés. Les automates temporisés de [AD90, AD94] ont les caractéristiques suivantes : le contrôle fini Q manipule un ensemble X d'horloges, avec un ensemble C de contraintes dans $\mathcal{C}_1(X)$, et des actions dans $\Sigma \cup \{\varepsilon\}$, où Σ contient les actions visibles, tandis que le mot vide ε désigne une action non observable. En raison de la forme particulière des sauts, l'étiquette d'une transition est notée comme un triplet (φ, a, λ), où φ est la garde, a est l'action et $\lambda \in \mathcal{P}(X)$ est le sous-ensemble de X des horloges qui doivent être remises à zéro.
La sémantique est celle du système de transitions mixte $\mathcal{T}_m$, auquel peuvent être ajoutées des conditions d'acceptation, spécifiées par des sous-ensembles de Q :
- $I \subseteq Q$ décrit les états de contrôle initiaux,
- $F \subseteq Q$ est un ensemble d'états finals,
- $R \subseteq Q$ représente une condition de Büchi.

Rappelons qu'un chemin infini P : $(q_0, v_0) \xrightarrow{(a_1,d_1)} (q_1, v_1) \xrightarrow{(a_2,d_2)} \ldots$ dans $\mathcal{T}_m$ est accepté avec la condition de Büchi R si au moins un des états de R est répété une infinité de fois dans P.

Un tel automate est donc un n-uplet $\mathcal{A} = (X, \Sigma, Q, E, I, F, R)$, où $E \subseteq Q \times (C \times (\Sigma \cup \{\varepsilon\}) \times \mathcal{P}(X)) \times Q$ est l'ensemble des transitions. Considérons la séquence d'étiquettes $w = (a_i, d_i)_{i \geq 1}$ d'un chemin de $\mathcal{T}_m$, commençant dans un état initial avec des valuations nulles et vérifiant une des conditions d'acceptation. Comme expliqué dans la première partie (paragraphe 2.1), on construit une séquence de temps en cumulant les délais, et le mot temporisé correspondant $obs(w)$ est obtenu en supprimant ensuite les paires pour lesquelles $a_i = \varepsilon$. Pour des horloges à valeurs dans $\mathbb{T}$, l'ensemble des mots acceptés ayant des dates dans $\mathbb{T}$ est le langage accepté par $\mathcal{A}$, noté $L(\mathcal{A}, \mathbb{T})$.

On note TA_ε l'ensemble des automates dont les étiquettes sont dans $\mathcal{C}_1(X) \times (\Sigma \cup \{\varepsilon\}) \times \mathcal{P}(X)$, TA^0_ε le sous-ensemble de ceux dont les contraintes sont restreintes à $\mathcal{C}_0(X)$ et TA le sous-ensemble de ceux qui n'ont pas d'ε-transitions. On peut aussi définir $TA^0 = TA \cap TA^0_\varepsilon$. Les classes de langages correspondantes associées au domaine de temps $\mathbb{T}$ sont désignées respectivement par $TL_\varepsilon(\mathbb{T})$, $TL^0_\varepsilon(\mathbb{T})$, $TL(\mathbb{T})$ et $TL^0(\mathbb{T})$. Rappelons les propriétés concernant ces langages :

Théorème 3.3 ([AD90, AD94]).

1. *les classes TL et TL_ε sont fermées par union et intersection (finies),*
2. *le vide est décidable dans TL_ε,*
3. *$TL_\varepsilon = TL^0_\varepsilon$ et $TL = TL^0$.*

Pour des constantes entières, le résultat de décidabilité est une conséquence immédiate de la construction d'une partition, combinée à la propriété de contrôle fini (théorème 3.2). Lorsque les constantes sont rationnelles (mais toujours en nombre fini), on se ramène facilement au cas précédent en les multipliant par leur dénominateur commun. Alur et Dill ont par ailleurs montré que la décidabilité du vide est un problème PSPACE-complet, ainsi que l'accessibilité. Un panorama plus complet sur ces questions apparaît dans [CY92]. Malgré ces résultats de complexité, les aspects pratiques de la vérification ont conduit à chercher des constructions plus efficaces pour les quotients [ACH+92, YL93, BLP+99]. D'autre part, des études théoriques ont été menées pour obtenir, à l'intérieur même de la classe des automates temporisés, des familles possédant de bonnes propriétés. Ainsi, [AFH94] étudie le problème de la déterminisation, [HKWT95] propose plusieurs hiérarchies classant les automates temporisés suivant leur pouvoir d'expression et [GHJ97] examine une notion de robustesse.

Extensions. Parmi les extensions mentionnées au paragraphe 2.4, le vide reste décidable dans les classes de [DZ98] et [KPSY99], et dans celle des automates rectangulaires. Ces classes semblent maximales puisque des généralisations très faibles rendent l'accessibilité indécidable [Hen96]. En particulier, les extensions des remises à zéro étudiées dans [BDFP00a, BDFP00b] font apparaître un phénomène intéressant en regard du point 3 du théorème 3.3 : alors que les ensembles de contraintes $\mathcal{C}_0(X)$ et $\mathcal{C}_1(X)$ fournissent les mêmes classes pour les automates temporisés, les résultats de décidabilité obtenus dans ces travaux sont liés à la présence ou non de contraintes atomiques triangulaires.

Enfin, certains résultats de décidabilité ont été obtenus récemment pour des classes situées au-delà des systèmes linéaires. On peut citer [BPT01] pour les automates de données et [LPY99, LPY01] pour des systèmes associés à des équations différentielles linéaires.

4 Expressivité et décidabilité

Les résultats de ce chapitre sont tirés de [BDGP98, BD00, BP00]. Ils se situent dans la ligne qui vient d'être décrite, avec pour objectif d'examiner les limites des propriétés de décidabilité du vide, pour des extensions des automates temporisés. Nous commençons par une preuve d'indécidabilité, pour un modèle d'automates temporisés étendus avec des contraintes additives sur les horloges. Puis, nous présentons une étude détaillée des langages temporisés acceptés par des automates avec ε-transitions. Enfin, le dernier paragraphe examine la question des suites convergentes de temps. Pour ces deux paragraphes, les démonstrations ne sont exposées que lorsqu'elles donnent lieu à une interprétation géométrique intuitive, en termes de régions.

4.1 Contraintes additives

Lorsqu'on décide d'ajouter des contraintes additives aux classes usuelles $\mathcal{C}_0(X)$ et $\mathcal{C}_1(X)$, on obtient un modèle pour lequel le vide est indécidable [AD94]. Cependant, une étude plus fine de ces contraintes additives s'est avérée utile, en raison d'un cas particulier où le problème est décidable. Nous avons donc proposé dans [BD00] une construction «minimale» pour la preuve d'indécidabilité, utilisant seulement quatre horloges et n'ajoutant à $\mathcal{C}_0(X)$ qu'une contrainte additive de la forme $x + y = 1$, pour deux horloges x et y.

Décidabilité en dimension 2.
Le cas particulier mentionné ci-dessus concerne la classe TL^2_+ des langages acceptés par des automates temporisés avec deux horloges et des contraintes atomiques dans $\{x \bowtie c,\ x - y \bowtie c,\ x + y \bowtie c\}$, c'est-à-dire $\mathcal{C}_{1+}(\{x, y\})$. Nous avons :

Proposition 4.1.

1. *Le vide est décidable dans TL^2_+.*
2. *TL^2_+ n'est pas contenue dans TL.*

En effet, pour l'ensemble $\mathcal{C}^m_{1+}(\{x, y\})$ correspondant à des constantes entières bornées par m, on construit facilement une partition compatible de $\mathbb{R}^2_+$, illustrée dans la figure 11 pour $m = 3$. D'autre part, le langage $L = \{(a^n, t_1 \dots t_n) \mid n \geq 1 \text{ et } t_i = 1 - 1/2^i,\ 1 \leq i \leq n\}$ appartient à TL^2_+, puisqu'il est accepté par l'automate de la figure 11. Un résultat de [DGP97] permet ensuite de montrer que L n'est pas accepté par un automate temporisé avec contraintes dans $\mathcal{C}_1(X)$.

Indécidabilité en dimension 4.
Nous considérons maintenant la classe TL^4_+ des langages acceptés par des automates temporisés avec au plus quatre horloges et des contraintes dans $\mathcal{C}_{0+}(X)$, où les seules contraintes atomiques additives sont de la forme $x + y = 1$, pour deux horloges x et y. Nous montrons alors le résultat suivant :

Théorème 4.2 ([BD00]). *Le vide n'est pas décidable dans la classe TL^4_+.*

La démonstration repose sur la construction d'un automate, acceptant un langage de la classe TL^4_+, et qui simule une machine à deux compteurs. Rappelons qu'une telle machine $\mathcal{M}$ consiste en une séquence finie d'instructions étiquetées, qui manipulent deux compteurs c et d, de valeurs initiales nulles. Cette séquence se termine par une instruction spéciale d'étiquette

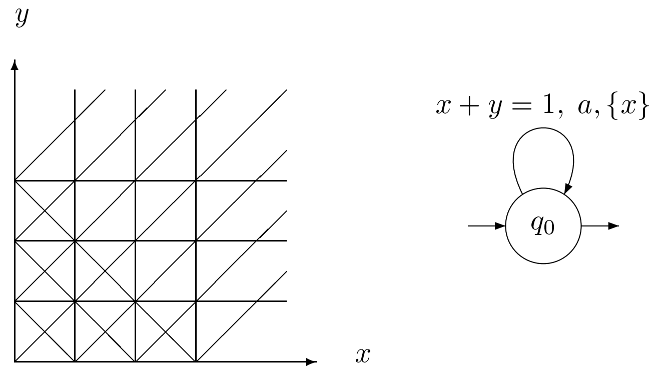


FIG. 11 – Une partition de $\mathbb{R}^2$ pour $\mathcal{C}^3_{1+}(\{x,y\})$ et un automate de TL^2_+ pour L

Halt. Les autres instructions ont l'une des deux formes suivantes, où $z \in \{c,d\}$ représente un compteur :

1. ℓ : z :=z+1 ; goto ℓ' ;
2. ℓ : if z=0 goto ℓ' else z :=z-1 ; goto ℓ'' ;

Le comportement de $\mathcal{M}$ est décrit par une suite (éventuellement infinie) de configurations : $< \ell_0, 0, 0 >< \ell_1, n_1, m_1 > \cdots < \ell_i, n_i, m_i > \cdots$, où n_i et m_i sont les valeurs respectives des compteurs et ℓ_i est l'étiquette, après la $i$ème instruction. Le problème de l'arrêt d'une telle machine est alors indécidable [Min67].
Nous obtenons une réduction de ce problème en construisant un automate temporisé $\mathcal{A}_\mathcal{M}$ qui atteint un état final si et seulement si $\mathcal{M}$ s'arrête.

L'automate $\mathcal{A}_\mathcal{M}$ a pour alphabet $\Sigma = \{c, d, \#\}$, où $\#$ désigne une action interne visible. La valeur n d'un compteur $z \in \{c,d\}$ est codée par un facteur $w = (x^n, t_1 \dots t_n)$, où la séquence de temps est de la forme $p-1/2, p-1/2^2, \dots, p-1/2^n$, dans un intervalle $[p-1,p]$ de longueur 1. Une telle séquence peut être vue comme une copie (avec décalage) de celles qui sont produites par le langage L utilisé pour la démonstration de la proposition 4.1. Les valeurs des compteurs c and d alternent : pour $i \geq 0$, n_i lettres c sont lues dans l'intervalle $[2i, 2i+1]$ et m_i lettres d dans $[2i+1, 2(i+1)]$. De plus, des actions temporisées de la forme $(\#, t)$ peuvent apparaître avant ou après ces facteurs.

L'ensemble des quatre horloges est $X = \{u, x_0, x_1, x_2\}$. L'horloge u permet de calculer des intervalles de longueur 1 et chacune des horloges x_i joue un rôle particulier à l'intérieur d'un intervalle. Cependant, ces rôles sont échangés à la fin de chaque intervalle.

- L'une des x_i calcule les dates associées au compteur courant, comme dans l'automate de la figure 11.
- Une autre x_i, appelée horloge de référence du compteur courant, est utilisée pour arrêter le calcul.
- La dernière horloge prendra le rôle de référence dans l'intervalle suivant.

Les états de $\mathcal{A}_\mathcal{M}$ sont : $Q = \{(\ell, x_i) \mid i \in \{0,1,2\}\} \cup \{loop(\ell, x_i) \mid i \in \{0,1,2\}\} \cup \{q_0\}$, où q_0 est l'état initial et l'ensemble d'états finals est $F = \{(Halt, x_i) \mid i \in \{0,1,2\}\}$.

Les transitions de $\mathcal{A}_\mathcal{M}$ sont décrites dans des *modules* de l'automate qui exécutent les instructions de $\mathcal{M}$. L'initialisation est représentée sur la figure 12 et les autres modules sont utilisés pour incrémenter, décrémenter ou conserver la valeur d'un compteur.

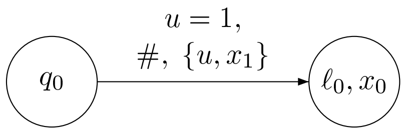


FIG. 12 – Initialisation

Chaque opération est réalisée par trois copies d'un même module, associées respectivement à chacune des trois horloges x_i, $i \in \{0, 1, 2\}$. Un module qui commence dans l'état (ℓ, x_i) correspond à l'instruction d'étiquette ℓ avec x_i comme horloge de référence. Dans les figures, les indices ℓ et x_i associés aux états *loop* ne sont pas représentés. La figure 13 décrit un module qui laisse le compteur c inchangé. Dans ce module, l'horloge de référence est x_0. L'horloge x_2 est remise à zéro par la dernière transition et servira de référence à c deux intervalles plus loin. L'horloge x_1 n'est pas utilisée mais sera la référence pour d dans l'intervalle suivant.

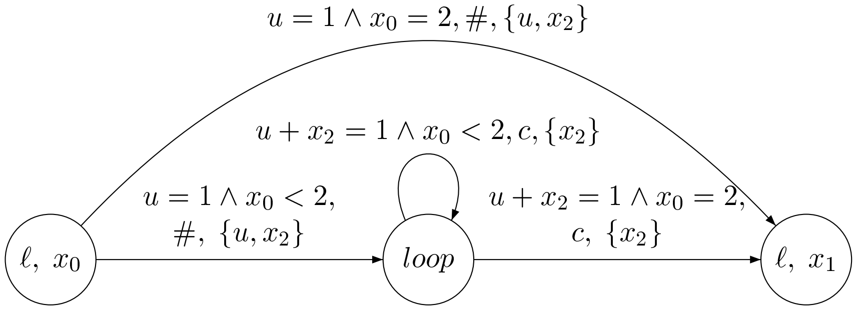


FIG. 13 – c inchangé, avec référence x_0

La figure 14 est le module qui teste et décrémente le compteur c, avec x_0 comme horloge de référence.

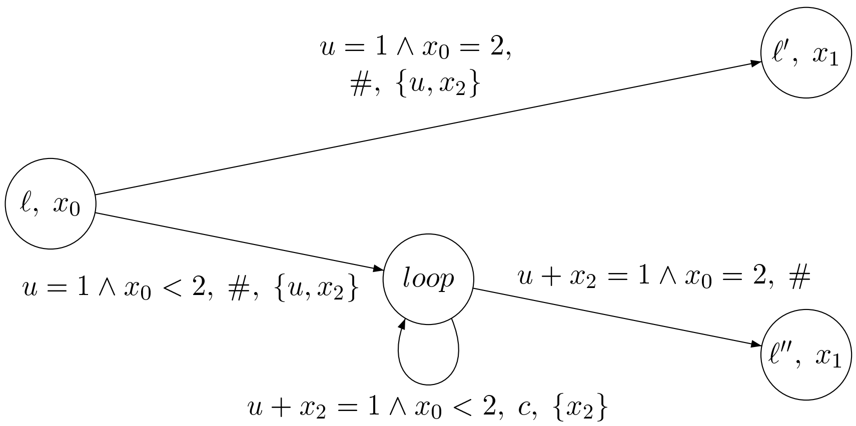


FIG. 14 – décrémentation de c, si $c > 0$, avec référence x_0

Remarquons que le fait de changer d'horloge de référence est imposé par la construction du module de décrémentation. En effet, lorsque x_0 atteint la valeur 2, elle ne peut plus être utilisée comme référence dans l'intervalle suivant, ce qui conduit à échanger ce rôle avec x_2.

Pour simuler $\mathcal{M}$, l'automate $\mathcal{A}_{\mathcal{M}}$ part dans l'état q_0 et, pour chaque instruction, réalise une opération pour c et une pour d. Par exemple, une incrémentation de d est effectuée dans $\mathcal{A}_{\mathcal{M}}$ en reliant un module laissant c inchangé, suivi d'un module incrémentant d. Supposons que $\mathcal{M}$ contienne n instructions. Alors $\mathcal{A}_{\mathcal{M}}$ contient $6n + 1$ modules et au plus $6n + 1$ états de contrôle. Pour l'ensemble $F = \{(Halt, x_i) \mid i \in \{0, 1, 2\}\}$ d'états finals, le langage accepté par $\mathcal{A}_{\mathcal{M}}$ est vide si et seulement si $\mathcal{M}$ ne s'arrête pas.

Signalons pour conclure que le problème reste ouvert pour trois horloges.

4.2 Transitions silencieuses

Dans les travaux fondateurs d'Alur et Dill [AD90] (mais pas dans les suivants), les automates temporisés peuvent contenir des ε-transitions ou transitions «silencieuses», qui représentent des actions internes non observables du système. Pour les automates finis non temporisés, il est bien connu que l'existence de telles transitions ne modifie pas le pouvoir d'expression du modèle. Au contraire, plusieurs exemples ont conduit à penser qu'elles jouaient un rôle différent lorsqu'elles étaient accompagnées d'opérations sur les horloges. Nous avons donc mené dans [BDGP98] une étude précise du modèle original et comparé la classe TL_ε des langages ainsi obtenus avec la classe TL utilisée dans les travaux ultérieurs. Nous avons montré que, pour un automate temporisé où les transitions silencieuses ne remettent jamais d'horloges à zéro, on peut construire un automate sans ε-transition acceptant le même langage.

Remarques préliminaires.
Rappelons qu'un automate temporisé comme nous l'avons défini précédemment est un n-uplet $\mathcal{A} = (X, \Sigma, Q, E, I, F, R)$ tel que :

- X est l'ensemble des horloges et Σ l'alphabet des actions visibles,
- Q est l'ensemble des modes de contrôle, $I \subseteq Q$ est le sous-ensemble d'états initiaux, $F \subseteq Q$ est l'ensemble d'états finals et $R \subseteq Q$ est l'ensemble d'états répétés,
- $E \subseteq Q \times (\mathcal{C}_1(X) \times \Sigma \cup \{\varepsilon\} \times \mathcal{P}(X)) \times Q$ est l'ensemble des transitions.

Deux automates sont dits *L-équivalents* s'ils acceptent le même langage temporisé.

Dans la définition des langages acceptés, nous n'avons pas exclu les mots associés à des suites convergentes de temps, contrairement à l'usage. Toutefois, ce problème est abordé en détails dans la suite. De plus, il suffit de remarquer que le langage :

$$Div = \{(a_i, t_i)_{i \geq 1} \in (\Sigma \times \mathbb{Q})^\omega \mid 0 \leq t_1 \leq t_2 \leq \cdots \text{ et } \forall t \in \mathbb{Q}, \exists i : t < t_i\}$$

qui contient les mots temporisés infinis dont les suites de temps divergent, est accepté par l'automate de la figure 15. Une simple opération d'intersection, autorisée par les propriétés de fermeture de la classe TL_ε, permettrait donc d'exclure les mots convergents.

Remarquons enfin qu'un automate temporisé de TA_ε peut accepter des mots finis avec un chemin infini contenant seulement des ε-transitions à partir d'un certain rang. La situation est illustrée avec $I = \{1\}$ et $R = \{2\}$ dans la figure 16.

Comparaisons des classes TL et TL_ε.
Les propriétés comparatives de ces classes peuvent être résumées par la proposition suivante, qui confirme l'intérêt de la classe TL_ε du point de vue de la robustesse et de l'expressivité :

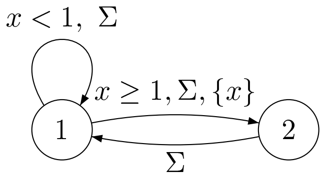


FIG. 15 – Automate acceptant *Div*

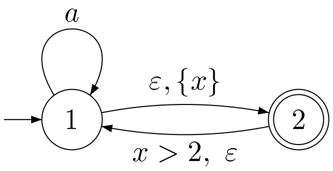


FIG. 16 – Accepter un mot fini par un chemin infini

Proposition 4.3 ([BDGP98]).

1. *la classe* $TL(\mathbb{T})$ *est strictement contenue dans* $TL_\varepsilon(\mathbb{T})$
2. *la classe* $TL(\mathbb{N})$ *n'est pas contenue dans* $TL(\mathbb{Q})$
3. $TL_\varepsilon(\mathbb{N}) \subseteq TL_\varepsilon(\mathbb{Q})$

Illustrons les points 1 et 2 en considérant les langages

$$\begin{array}{rcl} Pair & = & \{(a_i, t_i)_{i \geq 1} \in (\Sigma \times \mathbb{T})^\infty \mid t_i \in 2\mathbb{N} \text{ et } t_i \leq t_{i+1} \text{ pour tout } i\} \\ \text{et } Int & = & \{(a_i, t_i)_{i \geq 1} \in (\Sigma \times \mathbb{N})^\infty \mid t_i \leq t_{i+1} \text{ pour tout } i\} \end{array}$$

On peut vérifier que le langage $Pair$ est accepté par l'automate $\mathcal{A}_{Pair}$ de la figure 17, mais nous montrons qu'il ne peut pas être accepté sans ε-transitions. D'autre part, Int est accepté par l'automate $\mathcal{A}_{Int_1}$ sur le domaine de temps $\mathbb{N}$ et par $\mathcal{A}_{Int_2}$ sur $\mathbb{Q}$, mais comme $Pair$, il ne peut pas être accepté sur $\mathbb{Q}$ sans ε-transitions.

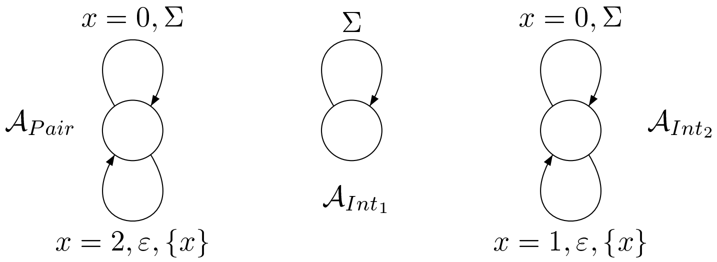


FIG. 17 – Automates temporisés pour *Pair* et *Int*.

L'inclusion stricte de TL dans TL_ε conduit à se demander où réside la puissance des transitions silencieuses. Nous examinons le cas où ces transitions ne remettent pas d'horloges à zéro. Franchir une séquence de telles transitions revient à laisser passer du temps à partir d'une configuration donnée, ce qui se traduit par une opération de *futur* sur les contraintes d'horloges. Un tel mécanisme est utilisé en général pour le calcul des successeurs d'un ensemble d'états du système (par exemple dans [ACH+95, HNSY94]).

Stabilité des contraintes d'horloges par futur.

Soit X un ensemble d'horloges et $C \subseteq \mathcal{C}(X)$, un ensemble de contraintes linéaires sur X. Pour $\varphi \in C$, on définit $\overrightarrow{\varphi}$ par :

$$v' \models \overrightarrow{\varphi} \text{ si } \exists v \leq v',\ v \models \varphi$$

Construire un automate qui combine des transitions silencieuses nécessite en particulier de pouvoir exprimer $\overrightarrow{\varphi}$ par une contrainte. Le problème est donc de savoir si l'ensemble C est fermé par l'opération de passage au futur et la solution de ce problème est étroitement liée aux propriétés de la partition. Considérons par exemple l'ensemble $C = \mathcal{C}_0^m(X)$ des combinaisons booléennes de contraintes rectangulaires, avec des constantes entières bornées par m, déjà utilisé dans la construction classique du quotient.

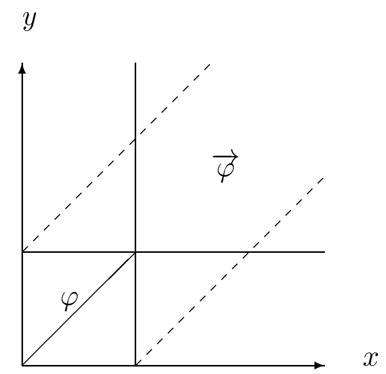


FIG. 18 – Futur d'une contrainte

Cet ensemble n'est pas fermé par futur, comme le montre la figure 18 dans le cas de deux horloges avec $m = 1$. En effet, dans ce cas, le futur de la contrainte $\varphi = [0 \leq x < 1 \wedge 0 \leq y < 1]$ est strictement contenu dans la réunion des t-successeurs de φ.
En revanche, nous avons :

Théorème 4.4 ([BDGP98]). $\mathcal{C}_1(X)$ *est fermé par futur.*

Bien qu'une démonstration directe, applicable avec un choix plus large de constantes, soit donnée dans [BDGP98], nous préférons mettre cette propriété de fermeture en relation avec des propriétés de la partition associée aux contraintes d'horloges. Dans le contexte général de gardes dans $\mathcal{C}(X)$, où les seules opérations de mise à jour sont des remises à zéro, nous définissons, pour une valuation v et un sous-ensemble λ de $\mathcal{P}(X)$, la valuation $v[\lambda]$ par : $v[\lambda](x) = 0$ si $x \in \lambda$ et $v[\lambda](x) = v(x)$ sinon.
Considérons maintenant un ensemble C de contraintes et une partition compatible $\mathcal{R}$ de l'ensemble V des valuations, en notant $[v]$ la classe d'équivalence d'une valuation v. L'opération de passage au futur est naturellement définie pour une région par : $\overrightarrow{\rho} = \{v' \mid \exists v \in \rho,\ v \leq v'\}$ et la compatibilité de $\mathcal{R}$ s'exprime par :

- pour toute région $\rho \in \mathcal{R}$, pour toute garde $\varphi \in C$, ou bien $\rho \subseteq \varphi$, ou bien $\rho \subseteq \neg\varphi$,
- pour toute région ρ, $\{v[\lambda] \mid v \in \rho\}$ est contenu dans une région, auquel cas on peut définir $\rho[\lambda] = [v[\lambda]]$,
- pour toute région ρ,

$$\overrightarrow{\rho} \subseteq \bigcup_{\rho' \in Post_t(\rho)} \rho'$$

Nous dirons que $\mathcal{R}$ est *fermée par futur* si pour toute région ρ,

$$\overrightarrow{\rho} = \bigcup_{\rho' \in Post_t(\rho)} \rho'$$

On voit alors que si C admet une partition fine (au sens du paragraphe 3.3) et fermée par futur, alors C est fermé par futur.
Il reste à remarquer qu'une partition construite pour $\mathcal{C}_1^m(X)$ sur le modèle de la figure 10 est fine et fermée par futur.

Une construction similaire permet d'associer à une contrainte φ son passé modulo une remise à zéro $\overleftarrow{\varphi}^{\lambda}$, défini par :

$$v \models \overleftarrow{\varphi}^{\lambda} \text{ si } \exists v',\ v[\lambda] \leq v' \text{ et } v' \models \varphi$$

et donc pour une région

$$\overleftarrow{\rho}^{\lambda} = \{v \in V \mid [v[\lambda]] \leq \rho\} = \bigcup_{\rho'[\lambda] \leq \rho} \rho'$$

Autrement dit, la simple compatibilité d'une partition fine $\mathcal{R}$ suffit pour obtenir sa fermeture par passé modulo reset. On en déduit donc la fermeture de $\mathcal{C}_1(X)$ par cette opération.

Égalité des classes TL et RTL_ε.
Notons maintenant RTA_ε le sous-ensemble de TA_ε des automates temporisés dont les ε-transitions ne remettent pas d'horloges à zéro et par RTL_ε la classe de langages correspondante. Le résultat annoncé s'exprime alors par :
Théorème 4.5 ([BDGP98]). $TL = RTL_\varepsilon$

Nous résumons brièvement les étapes successives permettant, à partir d'un automate temporisé $\mathcal{A}$ de la classe RTA_ε, de construire un automate de TA acceptant le même langage.

1. La première phase consiste à résoudre le problème posé dans la figure 16, en construisant, pour un automate de la classe RTA_ε, un automate temporisé L-équivalent pour lequel un mot fini peut être accepté par un chemin fini.
2. Dans une deuxième étape, on peut alors modifier à nouveau l'automate pour retirer les cycles formés uniquement d'ε-transitions. La possibilité de réaliser une telle transformation résulte du fait qu'on peut maintenant borner le nombre d'ε-transitions consécutives.
3. Un troisième problème est dû aux ε-transitions qui atteignent des états répétés. En dupliquant les états de l'automate, on peut obtenir une partition des états $Q = Q_1 \cup Q_2$, telle que (i) les états de Q_1 soient exactement les états d'arrivée des ε-transitions et (ii) l'ensemble R des états répétés soit contenu dans Q_2.
4. La quatrième étape consiste à déplacer les contraintes situées sur des séquences d'ε-transitions de la forme $e : q \xrightarrow{\varphi_1, \varepsilon} q_1 \xrightarrow{\varphi_2, \varepsilon} q_2 \cdots q_{k-1} \xrightarrow{\varphi_k, \varepsilon} q'$, sur la transition visible qui suit immédiatement ou, si l'état q' est final, sur la transition visible qui précède immédiatement la séquence. Dans tous les cas, on ajoute une horloge x_q qui est remise à zéro en arrivant dans l'état q. On traite d'abord toutes les séquences relatives au premier cas, en considérant la disjonction des contraintes

$$\varphi(e) = \overrightarrow{\overrightarrow{\overrightarrow{x_q = 0} \wedge \varphi_1} \wedge \cdots \wedge \varphi_k}$$

pour l'ensemble des séquences d'ε-transitions de q à q'.

On traite ensuite le deuxième cas, en introduisant de façon similaire des contraintes de la forme

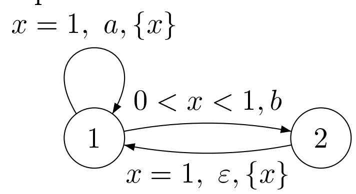
$$\psi(e,\lambda) = \overleftarrow{\overrightarrow{\overrightarrow{\overrightarrow{x_q = 0} \wedge \varphi_1} \wedge \cdots \wedge \varphi_{k-1}}}^{\lambda}$$

5. Finalement, après ajout éventuel d'un état puits pour accepter le mot vide, il reste à supprimer toutes les ε-transitions.

Transitions silencieuses avec remise à zéro.

Poursuivant ce travail, Diekert, Gastin et Petit ont montré dans [DGP97] les résultats suivants :

1. Soit $\mathcal{A}$ un automate temporisé dans lequel aucun cycle ne contient d'ε-transition remettant des horloges à zéro. Alors on peut construire un automate $\mathcal{A}'$ de TA tel que :
   $L(\mathcal{A}) \subseteq L(\mathcal{A}')$ et
   $L(\mathcal{A}) \cap Div = L(\mathcal{A}') \cap Div$
   où Div est le langage des mots divergents, accepté par l'automate de la figure 15.
2. Le langage temporisé accepté par l'automate ci-dessous n'appartient pas à la classe TL.



L'exemple du point 2 montre que la construction obtenue au point 1 ne peut pas être étendue et permet donc de cerner précisément la puissance d'expression des ε-transitions.

4.3 Exécutions convergentes

Nous avons déjà dit qu'en l'absence de restrictions supplémentaires, des suites convergentes de dates sont parfois produites par des exécutions d'automates temporisés sur des domaines de temps denses. Certains travaux comme [HPZ95] ont examiné de près ces phénomènes de convergence, avec en particulier l'exemple de la balle qui rebondit en perdant de l'altitude. Lorsque ces comportements peuvent être suivis par d'autres séquences, on obtient des suites dites *transfinies*, indexées par des ordinaux. Dès les années 70, des automates (non temporisés) acceptant des mots transfinis ont été étudiés [Cho78, Woj85, HW92, Bed96], après leur introduction par Büchi [Büc73] pour montrer la décidabilité de la logique monadique du second ordre. Nous avons donc dans [BP00] complété les modèles existants en introduisant une notion explicite de temps, d'une façon similaire aux méthodes employées pour la construction de systèmes temporisés : les automates transfinis contrôlent un ensemble fini X d'horloges à valeurs dans $\mathbb{R}$, par des contraintes de $\mathcal{C}_0(X)$ et des remises à zéro. Nous avons montré que la classe de langages TTL ainsi obtenue (sans ε-transitions) étend la classe TL, tout en conservant de bonnes propriétés de fermeture, ainsi que le résultat de décidabilité du vide.

Mots temporisés indexés par des ordinaux.

Dans ce qui suit, les ordinaux (voir [Ros82]) sont utilisés principalement pour l'indexation des séquences. Rappelons que les ordinaux finis sont les entiers naturels, le premier ordinal infini étant désigné par ω. En général, un ordinal α représente un ensemble bien ordonné (à isomorphisme près) et on peut additionner deux ordinaux $\alpha+\beta$ en considérant la classe (pour les isomorphismes) de la réunion $\alpha \cup \beta$, où tous les éléments de α sont strictement inférieurs à ceux de β. L'ordinal $\alpha+1$ est appelé le *successeur* de α. Cette addition définit naturellement un produit : par exemple, $\omega.2 = \omega + \omega$ et $\omega^2 = \omega + \omega + ...$, ω fois. On utilise la notation habituelle $\omega^0 = 1$. Nous considérons ici seulement les ordinaux inférieurs à ω^ω, qui possèdent une décomposition polynômiale $\alpha = \sum_{k=p}^{0} \omega^k.n_k$, où $p, n_0, n_1 \cdots$ sont des entiers naturels et le *type* de α est le plus petit entier k tel que $n_k \neq 0$. Si α est de type strictement positif, il n'a pas de plus grand élément et il est appelé un ordinal *limite*. Un ordinal limite de type k peut s'écrire $\alpha = \beta + \omega^k$, où soit $\beta = 0$, soit β est un ordinal de type supérieur ou égal à k.

Un mot temporisé de longueur α sur un alphabet Σ est simplement une suite $w = ((a_i, t_i))_{i<\alpha}$, indexée par α, où les a_i sont dans Σ et la suite de dates $(t_i)_{i<\alpha}$ est croissante. La date $t_\alpha = sup(t_i, i < \alpha)$ est appelée la *date de fin* de w et notée $\tau(w)$. Un mot est donc convergent s'il se termine à une date finie. Dans ce contexte, on peut définir la concaténation de deux mots w et w' lorsque w se termine à une date finie, l'étoile et la puissance ω, et étendre ces opérations aux langages.

Automates transfinis.

Plusieurs modèles peuvent être définis et nous avons montré dans [BP00] qu'ils définissent une même classe de langages. Nous choisissons une version adaptée à cette présentation, sans faire intervenir de transitions silencieuses et avec des contraintes d'horloges dans $\mathcal{C}_0(X)$.
Un automate temporisé transfini ou tt-automate est un n-uplet $\mathcal{A} = (X, \Sigma, Q, E, I, F, \gamma)$, où

- X est l'ensemble fini d'horloges et Σ l'alphabet d'actions,
- Q est un ensemble fini de modes de contrôle, obtenu par réunion disjointe de deux ensembles $Q[0]$ et $Q[1]$, $I \subseteq Q[0]$ est l'ensemble des états initiaux et $F \subseteq Q$ est l'ensemble des états finals
- l'ensemble des transitions est $E \subseteq Q \times (\mathcal{C}_0(X) \times \Sigma \times \mathcal{P}(X)) \times Q[0]$, avec la condition : si $(s, \ell, q) \in E$ avec $s \in Q[1]$, alors q n'est dans aucune partie $R \subseteq Q[0]$ telle que $\gamma(R) = s$.
- γ est une application de $\mathcal{P}(Q[0])$, l'ensemble des parties de $Q[0]$ privé de l'ensemble vide, dans $Q[1]$.

Dans la plupart des exemples, on choisit $Q[1] = \mathcal{P}(Q[0])$ et l'identité pour γ. Pour obtenir une sémantique comparable à celle des systèmes temporisés usuels, utilisant un système de transitions mixte $\mathcal{T}_m$, il faut d'abord associer un état limite à un ordinal limite i.
La définition est la même que pour les automates transfinis non temporisés, c'est une condition analogue à une condition de Muller : notons $Succ^{<i}$ l'ensemble de tous les ordinaux successeurs strictement inférieurs à i et $\mathbf{q} = (q_j)_{j \in Succ^{<i}}$, alors $lim(\mathbf{q})$ est l'ensemble des états r pour lesquels il existe une suite strictement croissante d'ordinaux successeurs $(j_p)_{p\in\mathbb{N}}$ de limite i, telle que $q_{j_p} = r$ pour tout $p \in \mathbb{N}$. Enfin, l'état limite est $q_i = \gamma(lim(\mathbf{q}))$.
Considérons maintenant un chemin $(q_0, X(t_0)) \xrightarrow{(a_1,t_1)} (q_1, X(t_1)) \cdots$ de $\mathcal{T}_m$, où $X(t)$ désigne le n-uplet des valuations des horloges de X à la date t et supposons que la date t_i associée à l'ordinal limite i est finie. Dans ce cas, pour chaque horloge x, il existe une sous-suite convergente $x(t_h)$ des valuations de x à des dates croissantes t_h, ce qui permet de poser $x(t_i) = lim_{h\to i} x(t_h)$.
Un mot est accepté s'il est l'étiquette d'un chemin partant d'un état de I avec des valuations nulles, et finissant dans un état de F.

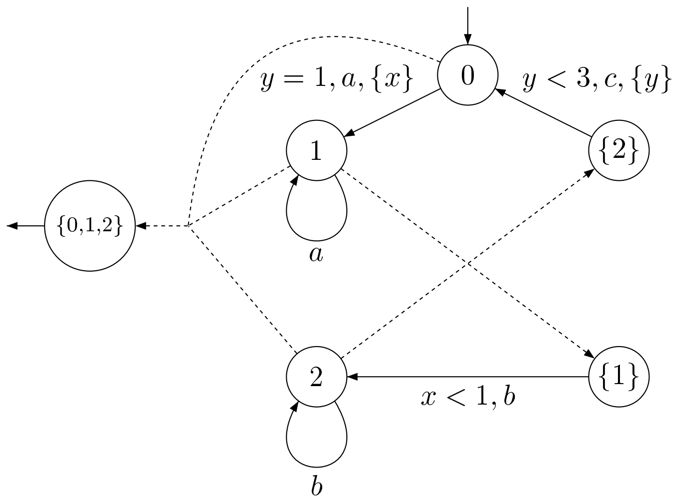


Fig. 19 – Un tt-automate

Par exemple, l'automate de la figure 19 accepte les mots temporisés transfinis $(a_i, t_i)_{i<\omega^2}$ tels que :
- $(a_i)_{i<\omega^2} = (a^\omega b^\omega c)^\omega$
- pour chaque facteur $a^\omega b^\omega c$, les dates du premier a et du premier b diffèrent de moins de 1 et le c se produit moins de 2 unités de temps après ce premier a.

Sur cette figure, les flèches en pointillés représentent le passage à l'état limite. Par exemple, en partant de l'état initial 0, l'automate exécute une transition a, suivie d'une infinité d'autres dans l'état 1, ce qui déclenche le passage dans l'état limite $\{1\}$. De même, lorsqu'un chemin d'étiquette $a^\omega b^\omega c$ passant par les trois états de type 0 (0, 1, 2) a été parcouru ω fois, on obtient l'état limite $\{0, 1, 2\}$.

Propriétés des langages.
Nous désignons par $TTL(n)$ la classe de langages acceptés par des automates transfinis pour lesquels les mots sont de longueur strictement inférieure à ω^n et $TTL = \cup_{n\geq 1} TTL(n)$. Ainsi, la classe usuelle TL des automates temporisés coïncide avec $TTL(1)$ et le langage accepté par l'automate de la figure 19 est dans $TTL(2)$. Les principales propriétés de la classe TTL sont résumées par les deux points suivants :

Théorème 4.6 ([BP00]).

1. *la classe TTL est fermée par union, intersection, concaténation, étoile et puissance ω,*
2. *le vide est décidable dans TTL.*

Le point 1 est obtenu par des adaptations des constructions pour le cas non temporisé. Nous ne les détaillons pas, afin de mettre encore une fois l'accent sur la question du quotient, qui intervient dans la démonstration du point 2. Ce travail nécessite quelques étapes préliminaires, qui permettent d'obtenir, à partir d'un tt-automate acceptant un langage L, un tt-automate $\mathcal{A} = (X, L, Q, E, I, F, \gamma_0)$ acceptant aussi L avec les propriétés suivantes :
- $Q[1] = \mathcal{P}(Q[0])$ et γ_0 est l'identité,
- toutes les transitions entrant dans un état donné remettent le même ensemble d'horloges à zéro,

- si q est un état de $Q[1]$, duquel partent des transitions, alors pour toute exécution atteignant q, la suite de dates associée est convergente.

La dernière étape repose sur la construction d'un automate transfini non temporisé $\mathcal{B}$ acceptant $Untime(L)$, où L est un langage de la classe TTL.

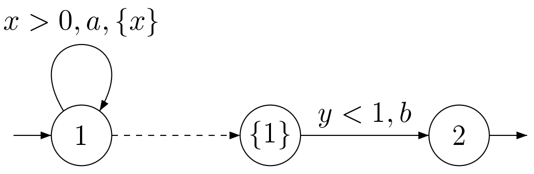


FIG. 20 – Un tt-automate pour L

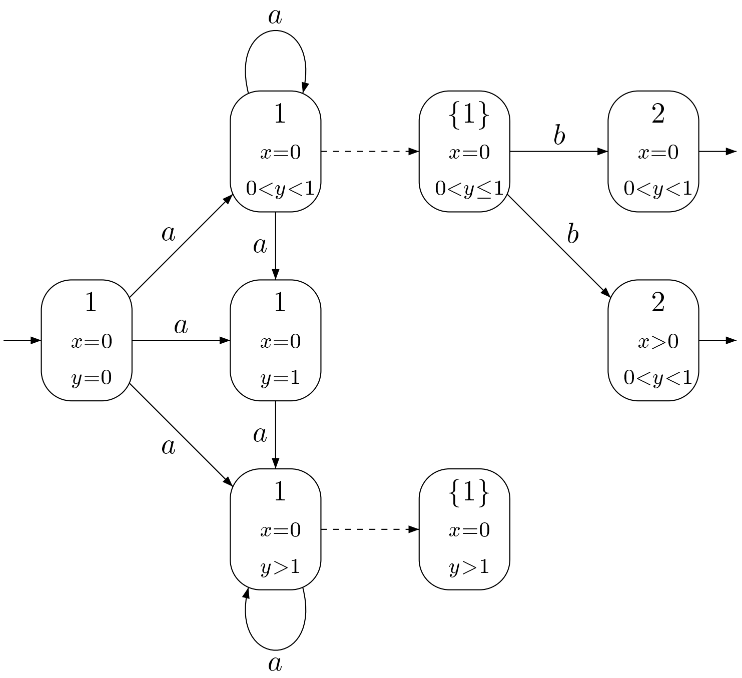


FIG. 21 – L'automate construit pour $Untime(L)$

Cette construction utilise à nouveau les régions associées aux contraintes d'horloges mais impose de définir une notion de limite pour l'ensemble des régions parcourues au cours d'un circuit. En effet, en prenant pour le nouvel automate des états de type 0 de la forme usuelle (q, ρ), où $q \in Q[0]$ et ρ est une région, on obtient comme limite un ensemble de la forme $s = \{(q_1, \rho_1), \ldots, (q_m, \rho_m)\}$. On définit donc $\gamma(s)$ comme une paire $(\mathbf{q}, Z)$, où $\mathbf{q} = \{q_1, \ldots, q_m\}$ et Z est l'ensemble des valuations, ou encore la *zone*, effectivement atteinte le long du circuit. Par exemple, la figure 20 représente un tt-automate acceptant le langage

$L = \{(a^\omega b, (t_i)_{i \leq \omega+1}) \mid \ 0 < t_0 < t_1 < \ldots < t_\omega \leq t_{\omega+1} < 1\}$

et la figure 21 représente l'automate des régions correspondant à la construction, qui accepte

$Untime(L) = \{a^\omega b\}$.

L'état $(\{1\}, x = 0 \wedge 0 < y \leq 1)$ est l'état limite obtenu lorsque l'état $(1, x = 0 \wedge 0 < y < 1)$ (de type 0) est infiniment répété.
L'état puits $(\{1\}, x = 0 \wedge y > 1)$ est obtenu par limite de $(1, x = 0 \wedge y > 1)$, avec cette fois pour la région $v = (x = 0) \wedge (y > 1)$, la zone limite $Z = lim(\{v\}) = v$.

5 Substitutions temporisées

La notion de substitution est le cadre formel correspondant à l'opération de raffinement. Par exemple, une procédure P est représentée à un certain niveau par une simple action a, qui est remplacée, dans la suite du développement, par l'ensemble des exécutions de P. Formellement, on associe un langage à toute lettre a d'un alphabet, et cette application est étendue en un morphisme. Les substitutions et leurs propriétés ont été étudiées depuis longtemps pour des modèles non temporisés, avec un intérêt particulier pour les classes de langages fermées par ces opérations. Le problème est plus délicat pour les langages temporisés, puisqu'une action se produit à une certaine date, ou avec une certaine durée, et qu'elle doit être elle-même remplacée par un langage comportant des contraintes de temps. D'autre part, le fait de considérer des mots convergents qui peuvent s'emboîter permet d'envisager de substituer à certaines actions, non seulement des langages de mots finis, mais aussi des ensembles de mots convergents. L'étude qui suit est tirée de [BGP97a, BP00]. Après avoir proposé une définition des substitutions temporisées pour des langages de la classe TTL, nous examinons les propriétés de fermeture de ces langages par substitution et nous montrons que la classe TTL peut être reconstruite à partir de la classe usuelle TL par des substitutions successives.

5.1 Sémantique des substitutions temporisées

Pour être capable de définir une notion de substitution dans le cadre temporisé, la première étape est d'interpréter un élément de base, c'est-à-dire une action temporisée (a, t). Dans [BGP97a], nous avons examiné deux sémantiques possibles. La plus simple, qui est sous-jacente à la plupart des modèles et largement utilisée en pratique depuis les travaux de [BC85, CHPP87], consiste à considérer que l'action a se produit de manière instantanée à la date t. Dans ce cas, pour deux actions successives $(a, t)(b, t')$, $t' \geq t$, la différence $t' - t$ représente un temps d'attente entre a et b. Une autre interprétation repose sur l'utilisation du délai $t' - t$ comme la durée maximale de l'action b, qui se produit, avec une durée inconnue, peut-être non nulle, dans l'intervalle $]t, t']$. Dans les deux cas, t est la date de fin de a.
Par exemple, pour $\Sigma = \{a, b\}$ et $\Sigma' = \{c, e\}$, on peut associer à a et b les langages temporisés suivants :

$$\begin{aligned} L_a &= \{(c,t)(e,t') \mid t \geq 1 \text{ et } t' - t \leq 2\} \\ L_b &= \{(e,t_1)\ldots(e,t_n)(c,t_{n+1}) \mid t_{i+1} - t_i < 1\} \end{aligned}$$

Considérons maintenant le langage $L = \{(a, t)(b, t + 4) \mid 0 \leq t \leq 3\}$.
Si les actions sont instantanées, l'image de L par cette substitution est :
$L' = \{(c, t)(e, t)(e, t + 4) \ldots (e, t + 4)(c, t + 4) \mid 1 \leq t \leq 3\}$.
Si les actions ont une durée, l'image de L est :
$L'' = \{(c, t)(e, t')(e, t_1) \ldots (e, t_n)(c, t_{n+1}) \mid 1 \leq t \leq t' \leq 3$ et $t' - t \leq 2$ et $t_1 \leq t'$ et $t_{i+1} - t_i < 1$ pour $i \geq 1$ et $t_{n+1} - t' = 4\}$.

Les projections constituent un cas particulier intéressant des substitutions, puisqu'elles permettent de considérer certaines actions comme internes, et de les rendre non observables. Elles correspondent au remplacement de toute lettre a par un des deux langages $L_a = \{\varepsilon\}$ ou $L_a = \{a\} \times \mathbb{T}$. Par exemple, effacer a dans L produit le langage $\{(b, t) \mid 4 \leq t \leq 7\}$.

Nous nous intéressons maintenant au cas où ces langages sont dans TTL.

5.2 Propriétés de fermeture

Dans le cadre usuel des automates non temporisés, une substitution est donnée par une famille d'automates finis $\mathcal{A}_a$ acceptant des langages L_a. Pour obtenir l'image par cette substitution d'un langage accepté par un automate $\mathcal{A}$, le mécanisme de base consiste à remplacer dans $\mathcal{A}$ toutes les transitions étiquetées par a, par une copie de $\mathcal{A}_a$.
Pour des langages temporisés, les automates $\mathcal{A}_a$ et $\mathcal{A}$ sont susceptibles de contenir des contraintes d'horloges. De plus, le franchissement d'une transition d'étiquette a dans $\mathcal{A}$ correspond à une transition $(q, \mathbf{x}) \xrightarrow{(a,d)} (q', \mathbf{x}')$ de $\mathcal{T}_m$. L'automate $\mathcal{A}_a$ doit alors admettre $(q, \mathbf{x})$ comme configuration initiale et produire une configuration finale acceptable. Nous considérons d'abord des substitutions non effaçantes, ce qui exclut le cas des projections et nous définissons une catégorie particulière de tt-automates, qui se comportent comme des transitions.

Transitions généralisées.
La classe GT des *transitions généralisées* contient les tt-automates $\mathcal{A} = (X, \Sigma, Q, E, I, F, \gamma)$ tels que :

- le mot vide ε n'appartient pas à $L(\mathcal{A})$ et
- pour toute exécution convergente de $\mathcal{A}$, commençant à la date t dans une configuration $(q, X(t))$, avec $q \in I$, et se terminant dans une configuration $(q_\alpha, X(t_\alpha))$, avec $q_\alpha \in F$, pour toute horloge $x \in X$, si x a été remise à zéro au moins une fois, alors sa valeur finale est $x(t_\alpha) = 0$, sinon $x(t_\alpha) = x(t) + t_\alpha - t$.

On note alors $L(\mathcal{A})_{X(t)}$ le langage obtenu en commençant les exécutions de $\mathcal{A}$ au temps t avec les valuations $X(t)$. Pour un mot convergent $w \in L(\mathcal{A})_{X(t)}$, on définit $\mathcal{X}(w, X(t))$ comme l'ensemble des valuations obtenues par les exécutions de $\mathcal{A}$ étiquetées par w, partant des valuations $X(t)$.

Une *tt-substitution* est une application $\sigma : \Sigma \longrightarrow GT$ qui associe à chaque lettre $a \in \Sigma$ une transition généralisée $\sigma(a) = A_a$ sur un alphabet Σ'. On note aussi $\sigma = (\mathcal{A}_a)_{a\in\Sigma}$ le n-uplet associé. L'image d'une action temporisée (a, t) par σ est définie pour une valuation initiale $X(t_0)$, $t \geq t_0$ par :

$$\sigma(a, t)_{X(t_0)} = \{w \in L(\mathcal{A})_{X(t_0)} \mid \tau(w) = t\}$$

On pourra considérer une valeur infinie de t lorsqu'on veut substituer à une action finale a un mot non convergent. On définit la *composition* (également appelée *concaténation absorbante* dans [ACM97]) de deux mots $w = (a_i, t_i)_{i<\alpha}$ et $w' = (a'_i, t'_i)_{i<\alpha'}$, si $\tau(w) \leq t'_0$, notée $w \circ w'$ par :

$$w \circ w' = ((a''_i, t''_i))_{i<\alpha+\alpha'} \text{ avec } \begin{cases} (a''_i, t''_i) = (a_i, t_i) & \text{pour tout } i < \alpha \\ (a''_{\alpha+i}, t''_{\alpha+i}) = (a'_i, t'_i) & \text{pour tout } i < \alpha'. \end{cases}$$

Cette notion est aussi utilisée dans [ACM97] Le franchissement de deux transitions consécutives (a, t) suivie de (b, t') accepte la composition $(a, t) \circ (b, t')$, si $t' \geq t$. Lorsque $\mathcal{A}_a$ et $\mathcal{A}_b$ sont substitués respectivement à a et b, les exécutions associées qui commencent aux valuations $X(t_0)$ acceptent la substitution de $(a, t) \circ (b, t')$, définie par :

$$\begin{aligned} \sigma((a, t) \circ (b, t'))_{X(t_0)} = \; & \{w \circ w' \mid w \in (L(\mathcal{A}_a))_{X(t_0)}, \tau(w) = t, w' \in (L(\mathcal{A}_b))_{X(t)} \\ & \text{pour une valuation } X(t) \in \mathcal{X}(w, X(t_0)) \text{ et } \tau(w') = t'\} \end{aligned}$$

Cette définition s'étend aux mots par induction sur la longueur, et pour un langage L, on pose : $\sigma(L) = \cup_{w\in L}(\sigma(w)_{X(0)})$. Remarquons que dans ce cas, on retrouve la définition usuelle

avec 0 comme date initiale et des valuations nulles.
Si L et L' sont deux langages de TTL, on dit que L' est un *raffinement* de L s'il existe une substitution σ telle que $\sigma(L) = L'$.

Nous obtenons alors le résultat suivant :

Théorème 5.1 ([BP00]). *TTL est la fermeture de $TL = TTL(1)$ par substitution.*

La démonstration de ce théorème nécessite plusieurs étapes. La première consiste à examiner le cas des substitutions définies par un n-uplet $(A_a)_{a\in\Sigma}$ de transitions généralisées telles que les langages $L_a = L(\mathcal{A}_a)$ soient dans $TTL(1)$. Appliquer une telle substitution σ au langage $a_I = \{(a,t) \mid t \in I\}$, où a est une lettre de Σ et I un intervalle de temps, produit comme image $\sigma(a_I) = L_a \cap \{w \mid \tau(w) \in I\}$, qui est reconnaissable lorsque L_a est fini. En revanche, l'ensemble $\Sigma_I^{inf} = \{w \mid w$ est infini et $\tau(w) \in I\}$ n'est en général pas reconnaissable. Plus précisément :

Proposition 5.2. *Soient c_1, c_2 des réels tels que $0 \leq c_1 < c_2$.*

1. *Pour un intervalle I de la forme $[0, c_2]$, $]c_1, +\infty[$ ou $]c_1, c_2]$, et un langage L de TTL, $L_I = \{w \in L \mid \tau(w) \in I\}$ appartient à TTL.*
2. *Pour un intervalle I de la forme $[c_1, c_2[$, $\{c_2\}$ ou $[c_2, +\infty[$, le langage Σ_I^{inf} n'est pas dans TL_ε.*

Nous obtenons alors :

Proposition 5.3. *Soit L un langage de $TTL(n)$ sur un alphabet Σ, accepté par un tt-automate $\mathcal{A}$ avec un ensemble X d'horloges. Soit σ une substitution définie par une famille $(\mathcal{A}_a)_{a\in\Sigma}$ de transitions généralisées dans $TTL(1)$. Notons λ_a l'ensemble des horloges remises à zéro dans $\mathcal{A}_a$. Supposons que pour toute lettre a de Σ,*

1. *pour toute transition $q \xrightarrow{\varphi,a,\lambda} q'$ de $\mathcal{A}$ d'étiquette contenant a, $\lambda_a \subseteq \lambda$ et*
2. *(a) ou bien L_a ne contient que des mots finis,*

   (b) ou bien pour toute transition $q \xrightarrow{\varphi,a,\lambda} q'$ de $\mathcal{A}$ d'étiquette contenant a, et pour toute horloge x, la garde φ est une conjonction de formules $x \in I$, pour un intervalle de la forme $I =]c_1, c_2]$, $I = [0, c_2]$ ou $I =]c_1, \infty[$,

alors $\sigma(L)$ appartient à TTL.

Notons que la condition 1 peut-être facilement obtenue lorsque les ensembles d'horloges Y_a des automates $(\mathcal{A}_a)_{a\in\Sigma}$ sont supposés disjoints et disjoints de l'ensemble X des horloges de $\mathcal{A}$. Il suffit alors d'ajouter à X l'ensemble $Y = \cup_{a\in\Sigma} Y_a$ et de remettre à zéro les horloges de λ_a sur les transitions d'étiquette a dans $\mathcal{A}$.
La construction dans le cas 2.(b) est elle-même réalisée en plusieurs phases successives que nous ne détaillons pas ici. Dans une deuxième étape, nous montrons une propriété de décomposition :

Proposition 5.4. *Pour tout entier $n \geq 1$, tout langage de $TTL(n+1)$ est le raffinement d'un langage de $TTL(n)$.*

Finalement, en combinant les deux résultats précédents, on obtient le résultat général de fermeture de TTL par des substitutions vérifiant les conditions de la proposition 5.3, pour des transitions généralisées quelconques dans TTL.

Les classes TL et TL_ε.

Dans ce paragraphe, nous considérons maintenant des TL_ε-substitutions, données par un n-uplet $(\mathcal{A}_a)_{a\in\Sigma}$, telles que les langages $L_a = L(\mathcal{A}_a)$ soient dans TL_ε et ne contiennent que des mots temporisés finis. Nous dirons qu'une telle substitution est une TL-substitution si de plus les langages L_a sont tous dans TL. Comme précédemment, une substitution est non effaçante si le mot vide ε n'appartient pas à L_a. Nous avons alors :

Proposition 5.5 ([BGP97a]).

1. *TL est fermée par TL-substitution non effaçante,*
2. *TL_ε est fermée par TL_ε-substitution,*
3. *TL n'est pas fermée par projection.*

Le premier point est une conséquence immédiate de la proposition 5.3 et le second résulte d'une extension naturelle de la construction précédente, où les ε-transitions sont autorisées dans tous les automates. En particulier, la classe TL_ε est fermée par projection [AD90]. Le troisième point indique que ce n'est pas le cas pour TL. En effet, le langage $L_2 = \{(a_i, 2i)_{i\geq 1} \in (\{a,b\} \times \mathbb{T})^\infty\}$ est accepté par l'automate de la figure 22.

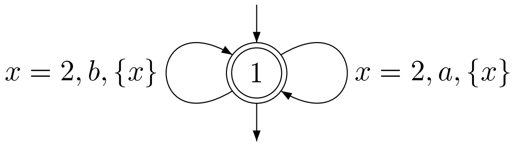


FIG. 22 – Un automate pour L_2

Mais l'image de L_2 par la projection qui efface le b est le langage *Pair* qui n'est pas dans TL (proposition 4.3).

6 Vérification d'un algorithme de contrôle du protocole ABR

La première étude [BFKM01] concerne un algorithme de contrôle des débits d'émission pour des usagers d'un réseau ATM. Cet algorithme, appelé $\mathcal{B}'$, a été proposé par Christophe Rabadan [Rab97] de France-Telecom, et fait actuellement partie de la norme ITU I-371.1 [ITU].

6.1 Le protocole ABR

Dans les réseaux ATM, le routage est réalisé par commutation de paquets, ce qui permet une plus grande flexibilité des communications, avec des débits qui peuvent varier suivant la charge du réseau. Parmi les contrats de connexion les plus récents, le mode *Available Bit Rate* (ABR) autorise un débit variable au cours d'une même session. Dans ce cas, le mécanisme d'offre et de contrôle des taux d'émission des usagers est lui-même dynamique. Comme illustré de façon schématique dans la figure 23, ce mécanisme comporte deux éléments.

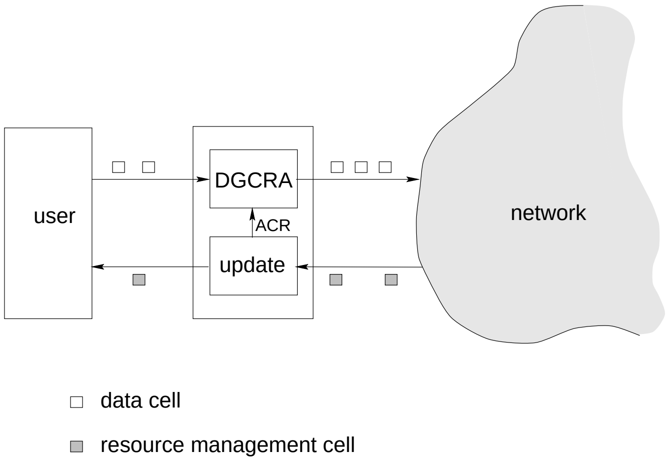


FIG. 23 – conformance control

Le premier consiste en un module de test, qui utilise un algorithme appelé DGCRA (Dynamic Generic Control of cell Rate Algorithm), simplement pour vérifier que le taux d'émission courant ne dépasse pas le taux courant autorisé, noté Acr (Allowed cell rate). Le second (update) doit réaliser à tout instant une mise à jour de ce taux Acr, en fonctions des informations provenant du réseau. Notre étude concerne précisément cette fonction de calcul, pour un instant futur donné t, du taux Acr(t), en fonction des valeurs successives du débit, qui apparaissent sur les cellules RM (Resource Management cells) en transit du réseau vers l'utilisateur.

Calcul théorique. Pour obtenir une valeur idéale de Acr(t), on considère donc la suite $(\mathrm{R}_0, \ldots, \mathrm{R}_n)$ des valeurs de taux contenues dans les cellules RM, ordonnées par leurs dates d'arrivée : $r_i \leq r_{i+1}$ pour tout i. Intuitivement, la valeur convenable de Acr(t) serait celle de la dernière cellule RM arrivée avant t, c'est-à-dire R_n, où $n = \mathrm{last}(t)$, en posant $\mathrm{last}(u) = max\{i \mid r_i \leq u\}$. Un premier problème provient des durées de transmission : l'utilisateur ne connaît cette valeur qu'après un certain délai τ, correspondant à un circuit complet entre lui-même et le contrôle. Ce délai est également variable, et la norme prévoit des bornes inférieure

(τ_3) et supérieure (τ_2) pour τ, ce qui impose de prendre en compte les valeurs R_i telles que i soit compris entre $m = \mathrm{last}(t-\tau_2)$ et $n = \mathrm{last}(t-\tau_3)$ et de prendre pour $\mathrm{Acr}(t)$ la plus grande de ces valeurs. Formellement, si $m \leq n$ et $r_m, r_{m+1}, \ldots, r_n$ sont les dates successives d'arrivée des cellules RM, avec

$$r_m \leq t - \tau_2 < r_{m+1} \leq r_{m+2} \leq \cdots \leq r_n \leq t - \tau_3,$$

et si $\mathrm{R}_m, \mathrm{R}_{m+1}, \ldots, \mathrm{R}_n$ sont les valeurs correspondantes, alors :

$$\mathrm{Acr}(t) = max\{\mathrm{R}_i, m \leq i \leq n\}.$$

Le cas où n et m sont égaux se produit lorsqu'aucune nouvelle cellule RM n'est arrivée dans l'intervalle $[t-\tau_2, t-\tau_3[$. Une telle spécification nécessiterait le calcul, à tout instant s, de la plus grande valeur des cellules RM reçues dans l'intervalle $]s-\tau_2, s]$. Avec plusieurs centaines de valeurs en général, un tel calcul n'est pas considéré actuellement comme raisonnable, ce qui conduit à une approche calculant une approximation.

Calcul d'une approximation. L'algorithme $\mathcal{B}'$ ne conserve à tout moment que deux valeurs de cellules RM, dans des variables notées Efi et Ela, avec leurs dates d'arrivée respectives tfi et tla. Il calcule une valeur approchée A de $\mathrm{Acr}(t)$ de la façon suivante : Lorsque la date courante s atteint la valeur tfi, A prend la valeur Efi et une seule valeur est conservée (Efi :=Ela, tfi :=tla). Lors de l'arrivée d'une nouvelle cellule RM à la date $s = r_k$, les variables Efi, Ela, tfi et tla sont mises à jour d'une manière qui dépend des positions de r_k par rapport tfi et tla et de R_k par rapport à Efi et Ela.

Objectif de vérification. Intégrer cet algorithme dans la norme a nécessité une étape de vérification : il faut démontrer qu'un usager n'est pas défavorisé par une comparaison avec A, c'est-à-dire que pour une donnée qui arrive à l'instant t, la valeur A est un majorant de $\mathrm{Acr}(t)$. Pour pouvoir comparer ces valeurs, $\mathrm{Acr}(t)$ est calculé par une méthode incrémentale. Dans les premiers travaux réalisés en Coq [BBC+97] par F. Klay et J. F. Monin [Mon98, MK99], ce calcul reposait sur un point de vue fonctionnel du premier ordre : on conserve à tout instant s une estimation E de la fonction Acr, qui est mise à jour lors de l'arrivée d'une cellule RM. Plus précisément, lorsqu'une cellule R_k arrive à la date r_k, la nouvelle fonction E' est obtenue à partir de E, selon les positions respectives l'argument t par rapport à r_k :

	$t < r_k + \tau_3$	$r_k + \tau_3 \leq t < r_k + \tau_2$	$r_k + \tau_2 \leq t$
$\mathrm{E}'(t)$	$\mathrm{E}(t)$	$max(\mathrm{E}(t), \mathrm{R}_k)$	R_k

On montre alors que la valeur $\mathrm{E}(t)$ obtenue au temps s est égale à $\mathrm{Acr}(t)$, pour $t \leq s + \tau_3$. Remarquons que ces fonctions E sont constantes sur des intervalles de temps minimaux $[a, b[$, où a et b sont de la forme $r_i + \tau_j$. Elles peuvent donc être représentées par des listes de paires de la forme $(t, \mathrm{E}(t))$, pour une date t, qui sont alors considérées comme des ordonnanceurs. Même en ne conservant que les paires significatives, les listes sont de très grande taille. De ce point de vue, l'algorithme $\mathcal{B}'$ constitue un ordonnanceur de taille inférieure ou égale à 2, contenant les paires $(\mathrm{tfi}, \mathrm{Efi})$ et $(\mathrm{tla}, \mathrm{Ela})$.

Une autre façon de décrire cet algorithme consiste à faire jouer à t le rôle d'un paramètre. Dans ce cadre, un algorithme $\mathcal{I}_t$ calcule une valeur E_t de la fonction ci-dessus à chaque réception d'une cellule RM par :

	$r_k \leq t-\tau_2$	$t-\tau_2 < r_k \leq t-\tau_3$	$t-\tau_3 < r_k$
E'_t	R_k	$max(\mathrm{E}_t, \mathrm{R}_k)$	E_t

FIG. 24 – Calcul de E_t

La valeur E_t est clairement égale à $\mathrm{Acr}(t)$ lorsque $s = t$. Si A est la valeur calculée par $\mathcal{B}'$, la propriété de correction de $\mathcal{B}'$, notée U_t pour une valeur t du paramètre, s'exprime alors de la façon suivante : quand $s = t$, $\mathrm{A} \geq \mathrm{E}_t$, et doit être vérifiée pour toutes les valeurs du paramètre t. Dans la suite, ce paramètre est omis et l'algorithme $\mathcal{I}$ est considéré comme une spécification.

Le cadre formel utilisé pour exprimer et vérifier cette propriété doit pouvoir manipuler explicitement une notion de temps. C'est pourquoi les systèmes hybrides linéaires ont paru bien adaptés à cette étude de cas. La sous-classe utilisée dans ce travail est constituée d'automates paramétrés ou p-automates. Ces automates sont une extension avec paramètres des modèles avec mises à jour étudiés dans [BDFP00a, BDFP00b]. Ils peuvent également être vus comme des variantes des modèles paramétrés proposés par Alur, Henzinger et Vardi dans [AHV93] (voir [Fri98]).

6.2 Les p-automates

Variables et contraintes. L'ensemble X des variables d'un p-automate est une partition $X = P \cup W \cup \{s\}$ où P est un ensemble de *paramètres*, W un ensemble de variables *discrètes* et s est une horloge *universelle*, c'est-à-dire une variable de pente 1, sans remise à zéro. Les valeurs des paramètres sont fixées par une contrainte initiale et n'évoluent plus par la suite. Les variables discrètes ont aussi une pente nulle mais peuvent être mises à jour lors des transitions.

Un *terme (paramétré)* est une expression de la forme $w + \Sigma_i \alpha_i p_i + \beta$, $s + \Sigma_i \alpha_i p_i + \beta$ ou $\Sigma_i \alpha_i p_i + \beta$, où w est une variable discrète, p_i est un paramètre et α_i, β sont des constantes dans $\mathbb{Z}$. Un terme sans paramètre est obtenu pour un ensemble d'indices vide. La sous-classe de contraintes formées à partir de ces termes est notée $\mathcal{C}_p$.

Avec les notations du premier chapitre, un saut est représenté par une conjonction $\varphi \wedge \psi$, où φ est une *garde* et ψ une mise à jour. La garde est une contrainte convexe de $\mathcal{C}_p$ à laquelle peut être ajoutée l'expression *asap*. La mise à jour généralise celle des automates rectangulaires : c'est une conjonction d'inégalités entre une variable discrète et un terme, notée $w' \bowtie term$ comme dans la présentation générale, avec w' une copie primée de w et $\bowtie \in \{<, \leq, =, \geq, >\}$. Ici encore, $w' = w$ est implicite si w' n'apparaît pas dans ψ.

Contrôle et états symboliques. Le graphe de contrôle est un automate fini dont chaque mode $q \in Q$ est de plus étiqueté par un invariant I_q. Les invariants sont des contraintes convexes de $\mathcal{C}_p$, qui permettent d'assurer la progression du temps : un automate ne peut rester dans un mode que si l'invariant associé est satisfait (la valeur par défaut d'un invariant est *true*).

Si $V = \mathbb{R}^X$ est l'ensemble des valuations, un état *symbolique* est un sous-ensemble de $Q \times V$ de la forme $\{q\} \times [\![\theta]\!]$, noté aussi par abus de langage $q \times \theta$, pour un mode $q \in Q$ et une contrainte θ. Un tel état est *admissible* si θ implique l'invariant I_q. Les deux types de transitions du système associé à un p-automate sont naturellement étendus aux états symboliques. À partir d'un état admissible $q \times \theta$, on atteint un autre état admissible $q' \times \theta'$:

- soit par un changement de modes $(q, \varphi, a, \psi, q')$, si θ implique φ et $(\mathbf{s}, \mathbf{w}')$ satisfait θ' si et seulement si $\exists \mathbf{w} \;\; (\mathbf{s}, \mathbf{w}) \models \theta \;\wedge \psi(W, W' := \mathbf{w}, \mathbf{w}')$,
- soit par un délai dans le mode $q' = q$, lorsque l'invariant correspondant reste vérifié et $(\mathbf{s}', \mathbf{w})$ satisfait θ' si et seulement si $\exists \mathbf{s} \leq \mathbf{s}' \;\; (\mathbf{s}, \mathbf{w}) \models \theta$.

Notons que les paramètres ont été omis puisqu'ils ne sont jamais modifiés.

Modélisation du système. Les algorithmes $\mathcal{I}$ and $\mathcal{B}'$ correspondent à des programmes réactifs, qui réalisent des opérations lors d'événements extérieurs, comme l'arrivée de cellules RM, ou lorsque le temps atteint certaines valeurs. Il est donc naturel d'utiliser un automate supplémentaire, qui fonctionne comme un générateur d'événements. Le système est alors représenté par un produit synchronisé classique des trois p-automates des figures 25, 26 et 27. Ces automates décrivent respectivement l'environnement, $\mathcal{I}$ et $\mathcal{B}'$, et ils partagent en particulier l'horloge universelle s.

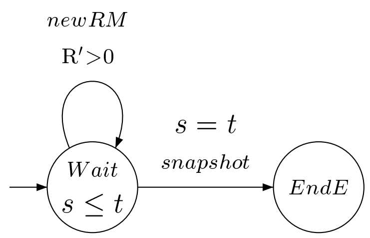


FIG. 25 – Automate pour l'environnement

L'automate de l'environnement produit deux types d'événements : les arrivées de cellules RM et une vue instantanée du système (ou *snapshot*) qui a lieu à l'instant paramétré t. La variable discrète R contient le taux de la dernière cellule reçue, dont la valeur est choisie de façon non déterministe par l'affectation $\mathrm{R}' > 0$.

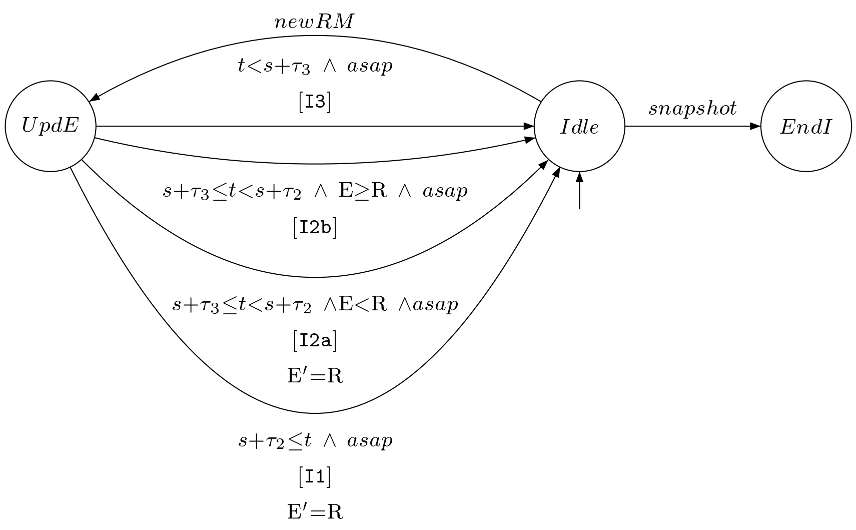


FIG. 26 – Automate pour $\mathcal{I}$

L'automate associé à l'algorithme $\mathcal{I}$ reflète le calcul de la figure 24, où le calcul de plus grand élément a été divisé en deux parties, désignées par `[I2a]` et `[I2b]`.

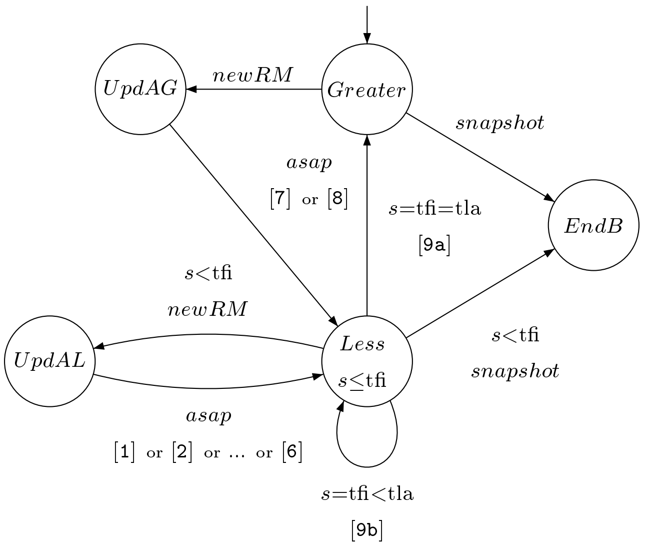


FIG. 27 – Automate pour $\mathcal{B}'$

De même, l'automate associé à l'algorithme $\mathcal{B}'$ reflète le calcul de la valeur approchée A évoqué plus haut, que nous ne détaillons pas.

6.3 Vérification

Deux méthodes ont été utilisées dans ce cadre pour vérifier des propriétés d'états symboliques (ou de réunions finies de tels états). La première est fondée sur les techniques d'assertions à la Floyd-Hoare, et consiste à montrer qu'un état Π est un invariant du modèle [Sha93]. La seconde est issue du model-checking : elle caractérise l'ensemble des états accessibles du système et montre que cet ensemble est contenu dans Π. Notons que le modèle est indécidable [AHV93] et que le succès de la méthode dépend de la terminaison du calcul des états accessibles. Dans le modèle décrit ci-dessus, la propriété U : when $s = t$, $\mathrm{A} \geq \mathrm{E}$ correspond à un état global du système après l'action *snapshot*. En notant le mode correspondant $q_\infty = (EndE, EndI, EndB)$, et en désignant par λ une variable parcourant l'ensemble des modes, la propriété devient : $U : \lambda = q_\infty \Rightarrow \mathrm{A} \geq \mathrm{E}$.

Preuve par invariants.
Cette méthode nécessite d'introduire dix propriétés auxiliaires Aux_i, $1 \leq i \leq 10$, et de démontrer que $\mathrm{Inv} \equiv U \wedge \bigwedge_{i=1}^{10} \mathrm{Aux}_i$ est un invariant du système. De plus, certaines de ces propriétés font intervenir une variable supplémentaire r, qui représente la date d'arrivée de la dernière cellule RM. Une légère modification de l'automate décrivant l'environnement, illustrée dans la figure 28, permet de résoudre facilement cette question.

Après un codage des p-automates dans Coq [Mon98], il est possible de prouver la propriété voulue.

Analyse d'accessibilité.
L'état (symbolique) initial du système est défini par $Init = q_+ \times \theta_0$,

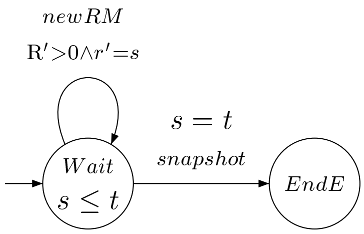


FIG. 28 – Automate de l'environnement avec variable r

où $q_+ = (Wait, Idle, Greater)$ et θ_0 est la contrainte
$(s = \mathrm{tfi} = \mathrm{tla}) \wedge (\mathrm{R} = \mathrm{E} = \mathrm{A} = \mathrm{Efi} = \mathrm{Ela}) \wedge\ (0 < \tau_3 < \tau_2)$.
On associe à la propriété U l'état $\overline{U} = q_\infty \times (\mathrm{A} < \mathrm{E})$ où elle n'est pas vérifiée et on teste si l'intersection $Post^*(Init) \cap \overline{U}$ est vide. Ce test est réalisé à l'aide de l'outil HyTech [HHW95], dans lequel les trois p-automates se codent directement. Le produit des automates et l'ensemble $Post^*(Init) \cap \overline{U}$ sont ensuite calculés automatiquement, avec comme résultat une intersection vide, ce qui montre la propriété. D'autres expériences du même type avaient déjà été réalisées [Fri98] avec l'outil Gap qui contient un programme logique de calcul de point fixe.

Conclusion.

Ce travail est actuellement poursuivi dans le cadre du projet RNRT Calife, où différentes variantes de l'algorithme $\mathcal{B}'$ ont déjà été vérifiées par une analyse d'accessibilité utilisant HyTech. Cependant, ce calcul échoue pour des algorithmes plus généraux [RSK00], utilisant des ordonnanceurs de taille strictement supérieure à deux. Ce dernier travail utilisait le démonstrateur PVS. Une preuve utilisant Isabelle sur les p-automates a également été obtenue par P. Castéran et D. Rouillard au LaBri [CR00]. L'approche par invariants est plus difficilement adaptable puisqu'elle nécessite de modifier certains invariants auxiliaires et d'en introduire de nouveaux. L'ensemble de ces travaux peut être considéré comme une étape préliminaire au développement combiné de démonstrateurs et de model-checkers, suggéré dans [Man94, RSS95, PS96].

7 Vérification de réseaux de Petri

Les variables utilisées dans les réseaux de Petri sont des *places* ou *compteurs*, à valeurs entières. Traditionnellement, l'analyse de ces modèles utilise donc la résolution de contraintes entières. Dans plusieurs sous-classes de réseaux de Petri, lorsque le calcul des états accessibles se termine, l'ensemble résultant est décrit par une formule de l'arithmétique de Presburger [FO97, BGP97b, SKR98, CJ98]. Cependant, la complexité des algorithmes étant très élevée, il peut sembler intéressant d'effectuer la résolution sur $\mathbb{R}$, qui est généralement plus facile [Sch86]. En particulier, les outils conçus pour la vérification de systèmes temporisés font typiquement appel à l'arithmétique réelle et la modélisation des réseaux de Petri en termes de systèmes linéaires, décrite au paragraphe 2.5, suggère fortement l'utilisation de tels outils. Cependant, une telle méthode appliquée à des entiers ne fournit qu'une surapproximation de l'ensemble des solutions [CS90].

Les travaux décrits dans ce chapitre, issus de [BF99], isolent certaines conditions dans lesquelles cette méthode permet d'obtenir un résultat exact. Cette méthode repose sur une observation de Shostack [Sho81], qui utilise la procédure de Fourier-Motzkin sur $\mathbb{R}$, pour éliminer des variables à valeurs entières dans des classes particulières de formules. Des extensions de ce travail ont été ensuite examinées dans [DP01].

7.1 Arithmétique réelle

Nous rappelons rapidement la procédure d´élimination de Fourier-Motzkin en suivant la présentation de [HSB97].

Pour l'ensemble X de variables, les formules considérées sont des comparaisons de termes linéaires à coefficients dans $\mathbb{Z}$. Pour une contrainte atomique de la forme $\Sigma_{i=1}^{n} a_i x_i \geq c$ ou $\Sigma_{i=1}^{n} a_i x_i = c$, on suppose qu'une même variable n'apparaît pas deux fois. On note $vars(\varphi)$ l'ensemble des variables d'une formule φ. Étant donnée une telle variable x, on note φ_x^0 la conjonction des contraintes atomiques de φ ne contenant pas x, et φ_x^+ (resp φ_x^-) l'ensemble des contraintes atomiques équivalentes à $ax \leq e$ (resp. $e \leq bx$), où a et b sont des entiers naturels et e une expression linéaire ne contenant pas x. L'algorithme de projection suivant élimine x de φ et retourne la formule $\psi \leftrightarrow \exists x \varphi$.

projection (φ, x)
`si` $\varphi \leftrightarrow \varphi' \wedge x = e$ `alors`
% remplacement
    $\psi := \varphi'$ où toutes les occurrences de x sont remplacées par e
`sinon`
% élimination de Fourier-Motzkin
    $\psi := \varphi_x^0$
    `pour tout` $A \in \varphi_x^+$ tel que $A \leftrightarrow ax \leq e^+$
        `pour tout` $B \in \varphi_x^-$ tel que $B \leftrightarrow e^- \leq bx$
            $\psi := \psi \wedge ae^- \leq be^+$
        `finpour`
    `finpour`
`finsi`
`retourner` ψ ;

Par exemple, pour la formule $\varphi : \ 2x - 3y - 1 \geq 0 \wedge \ 5y - 3x + 2 \geq 0$, on a $\varphi_x^+ : \ 3x \leq 5y + 2$ et $\varphi_x^- : \ 3y + 1 \leq 2x$. Par combinaison, on obtient $9y + 3 \leq 6x \leq 10y + 4$. Calculer $\exists x \varphi$ sur $\mathbb{Z}$

nécessite de trouver un multiple de 6 entre $9y + 3$ et $10y + 4$, alors que par élimination sur $\mathbb{R}$, on obtient $9y + 3 \leq 10y + 4$, qui fournit par simplification la formule $\psi : y + 1 \geq 0$.

La formule ψ est donc une surapproximation de l'ensemble des solutions *entières* puisque l'algorithme suppose qu'il existe toujours une solution x à $ae^- \leq abx \leq be^+$, si $ae^- \leq be^+$, ce qui est vrai sur $\mathbb{R}$ mais pas sur $\mathbb{N}$. Cependant, si $a = 1$ ou $b = 1$, il existe toujours une solution entière. Ainsi, dans le cas particulier où, pour la formule φ, tous les coefficients de x apparaissant dans les bornes inférieures de x valent 1 (ou si tous les coefficients de x apparaissant dans les bornes supérieures de x valent 1), la procédure de Fourier-Motzkin est *exacte* sur $\mathbb{N}$: la formule ψ caractérise l'ensemble des solutions [Pug92].

7.2 Accessibilité

Réseaux de Petri paramétrés. Un réseau de Petri est généralement muni d'un ensemble de marquages initiaux, qui font éventuellement intervenir des paramètres. L'exemple de la figure 29 a été proposé par M. Latteux pour modéliser le fonctionnement d'une piscine ([Bra83, DA89, ABBR92]). Dans ce réseau, tous les arcs sont de poids 1.

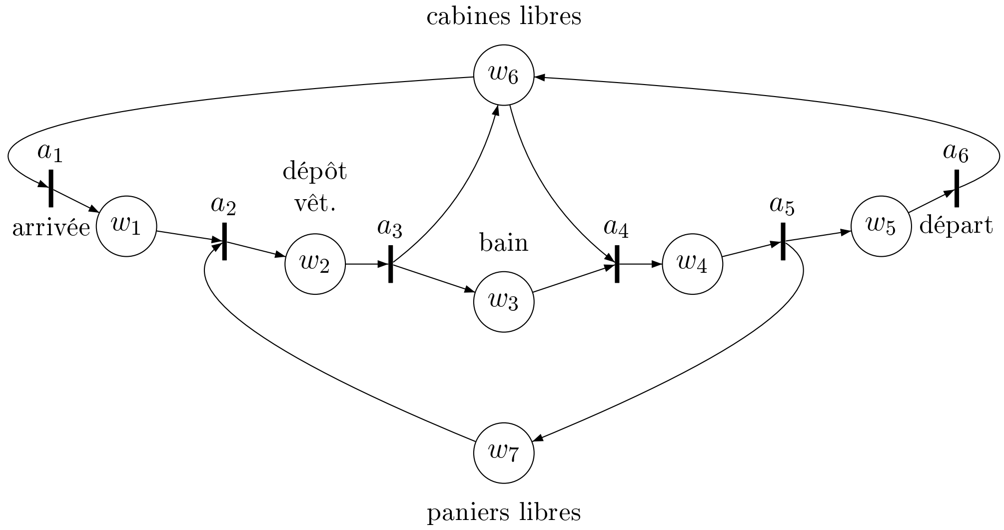


FIG. 29 – Un réseau de Petri pour l'exemple de la piscine

Dans cet exemple, deux paramètres p_1 et p_2 représentent respectivement le nombre de cabines libres et le nombre de paniers. Les conditions initiales sont exprimées par la contrainte $\theta_0 : \; w_1 = w_2 = w_3 = w_4 = w_5 = 0 \; \wedge \; w_6 = p_1 \wedge w_7 = p_2 \wedge p_1 \geq 1 \wedge p_2 \geq 1$.

Plus généralement, on considère un ensemble des variables de la forme $X = W \cup P$, où W est un ensemble de variables discrètes et P un ensemble de paramètres. Les sauts sont des contraintes de la forme $\varphi \wedge \psi$, où φ est une conjonction de contraintes atomiques $w \geq c$, pour $w \in W$ et $c \in \mathbb{N}$, et la mise à jour ψ est une conjonction de contraintes atomiques $w' = w + c$, où $c \in \mathbb{Z}$. Rappelons que le contrôle n'est pas nécessaire dans ce contexte. Sur un tel système linéaire, on peut appliquer les procédures de projection des systèmes temporisés pour résoudre les contraintes sur $\mathbb{N}$ associées naturellement aux problèmes d'accessibilité. Pour la sous-classe

$\mathcal{C}_n$ de $\mathcal{C}(X)$ construite à partir des formules atomiques

$$x + c \geq y,\ x \geq y + c, x \geq c,\ c \geq x,\quad \text{où } x, y \in X,\ c \in \mathbb{N}$$

sur l'ensemble de variables $X = W \cup P$, nous avons :

Proposition 7.1 ([BF99]).

1. *La procédure d'élimination de Fourier-Motzkin est exacte dans la classe $\mathcal{C}_n$.*
2. *La classe $\mathcal{C}_n$ est fermée par Post et lorsque le calcul (par élimination) des itérés $Post^k$ termine, la formule obtenue est encore dans $\mathcal{C}_n$.*

En effet, soit $\varphi \in \mathcal{C}_n$ et $a \in \Sigma$. La formule $Post_a(\varphi)$ est le résultat de l'élimination sur $\mathbb{N}$ de $\mathbf{x}'$ dans $\exists \mathbf{x}'\ \ \mathbf{x}' \models \varphi \wedge \mathbf{x}' \xrightarrow{a} \mathbf{x}$. On obtient par remplacement :

$$\mathbf{x} \models Post_a(\varphi) \Leftrightarrow\ \mathbf{x} - \mathbf{m}_a \models \varphi \wedge\ \mathbf{x} \geq \mathbf{g}_a + \mathbf{m}_a.$$

Le résultat provient alors du fait que $\mathcal{C}_n$ est fermée par remplacement de $x \in X$ par $x - m$.

Cette méthode permet en particulier de vérifier l'absence de blocage dans un réseau de Petri. Pour un état $\mathbf{x}$, cette propriété s'exprime par la formule $\overline{\varphi_b}$:

$$\mathbf{x} \models \overline{\varphi_b} \Leftrightarrow \bigvee_{a\in\Sigma} \mathbf{x} \geq \mathbf{g}_a.$$

L'existence d'un blocage, correspondant à la négation de $\overline{\varphi_b}$, peut être mise sous la forme d'une disjonction finie $\bigvee_{i\in I} \varphi_i$. Avec des conditions initiales décrites par une formule φ_0 de $\mathcal{C}_n$, le blocage se traduit donc par :

$$\exists i \in I\ \ \ Post^*(\varphi_0) \wedge\ \ \varphi_i \text{ a une solution.}$$

Fusions de transitions et métatransitions. Une technique classique consiste à ajouter à un réseau de Petri de nouvelles transitions obtenues par fusion. On simule ainsi en une seule étape l'exécution d'une séquence $\sigma = a_1 a_2 \dots a_k$. La relation d'accessibilité est alors étendue à σ en lui associant un couple de vecteurs $(\mathbf{g}_\sigma, \mathbf{m}_\sigma)$, définis inductivement par :

$$\begin{array}{lcl@{\qquad\qquad}lcl} \mathbf{m}_\varepsilon & = & \mathbf{0} & \mathbf{m}_{a\sigma} & = & \mathbf{m}_a + \mathbf{m}_\sigma \\ \mathbf{g}_\varepsilon & = & \mathbf{0} & \mathbf{g}_{a\sigma} & = & max(\mathbf{g}_a, \mathbf{g}_\sigma - \mathbf{m}_a) \end{array}$$

En particulier, lorsqu'une même transition a est itérée, on obtient pour la séquence $\sigma = a^n$:

$$\mathbf{m}_\sigma = n.\mathbf{m}_a \text{ et } \mathbf{g}_\sigma = \mathbf{g}_a - (n-1).\mathbf{m}'_a$$

où $\mathbf{m}'_a = min(\mathbf{0}, \mathbf{m}_a)$ est obtenu en remplaçant les composantes positives par 0.
Par exemple, l'itération de la transition a dans le réseau de la figure 8 du paragraphe 2.5 permet d'obtenir :

$$\mathbf{w} \xrightarrow{a^n} \mathbf{w}' \text{ ssi } \mathbf{w} \geq (9,1,0,0) \wedge\ \mathbf{w}' = \mathbf{w} + n.(-3,0,2,1)$$

et dans le réseau de la piscine, la fusion des transitions a_1 et a_2 est caractérisée par : $\mathbf{m}_{r_1 r_2} = (0,1,0,0,0,-1,-1)$ et $\mathbf{g}_{r_1 r_2} = (0,0,0,0,0,1,1)$, d'où

$$\mathbf{x} \xrightarrow{r_1 r_2} \mathbf{x}' \text{ ssi } w_6 \geq 1 \wedge w_7 \geq 1 \wedge w'_2 = w_2 + 1 \wedge w'_6 = w_6 - 1.$$

Remarquons qu'une telle extension ne modifie pas les ensembles $Post^*$ et Pre^*, ce qui préserve la proposition 7.1.

Une métatransition permet de simuler une boucle a^+ en une seule étape :

$$\mathbf{x} \xrightarrow{a^+} \mathbf{x}' \text{ si } \exists n \in \mathbb{N} - \{0\}\ \mathbf{x} \xrightarrow{a^n} \mathbf{x}'.$$

Pour une formule $\varphi \in \mathcal{C}_n$, on obtient alors :

$$\mathbf{x} \models Post_{a^+}(\varphi) \Leftrightarrow\ \exists n \geq 1\ \mathbf{x} - n.\mathbf{m}_a \models \varphi \wedge\ \mathbf{x} \geq \mathbf{g}_a - (n-1).\mathbf{m}'_a + n.\mathbf{m}_a$$

L'utilisation de metatransitions est une technique dite d'*accélération* [BW94, WB98]. Dans ce cas, l'élimination de n sur $\mathbb{R}$, lorsqu'elle se termine, fournit seulement une surapproximation $UPost$ de $Post^*$. Le calcul est exact seulement si tous les coefficients de n dans les bornes inférieures (ou dans les bornes supérieures) sont égaux à 1, dans les formules intermédiaires $UPost^k(\varphi)$ [BF99].

7.3 Applications

Nos résultats expérimentaux ont été obtenus avec HyTech, qui comporte une procédure d'élimination sur les réels utilisant la bibliothèque de manipulation de polyèdres de N. Halbwachs [Hal93]. Les résultats des exemples traités sont récapitulés dans le tableau ci-dessous.

	metatransitions	fusions	nombre d'itérations	temps
Système de production Détection de blocage	non	non	81	200 sec.
Lecteurs-écrivains Exclusion mutuelle	oui	non	20	12 sec.
Piscine Détection de blocage	oui	oui	22	50 sec.
PNCSA Détection de blocage	oui	oui	53	15 min.

Le (petit) système de production automatisée est modélisé par le réseau de Petri de la figure 30, extrait de [CX97] et introduit dans [ZDD93]. Il contient quatre machines, deux robots, deux tampons de capacité 3 et une cellule d'assemblage. Les variables x_i, $1 \leq i \leq 25$ sont les places et la vingt-sixième variable est un paramètre noté p. L'objectif est de rechercher les valeurs de p produisant un blocage, avec la contrainte initiale φ_0, qui correspond à $x_1 = p$, $x_2 = x_4 = x_7 = x_{12} = x_{13} = x_{16} = x_{19} = x_{24} = 1$, $x_{10} = x_{15} = 3$, et les autres places nulles.

L'absence de blocage est prouvée pour $1 \leq p \leq 4$ dans [ZDD93], pour $1 \leq p \leq 8$ dans [CX97], où une séquence menant à un blocage est également obtenue pour $p = 9$. Dans [Fri98], l'ensemble des marquages accessibles est calculé automatiquement par élimination sur $\mathbb{N}$, montrant ainsi que φ_0 produit un blocage si et seulement si $p \geq 9$. Cependant, ce calcul a pris plusieurs heures.

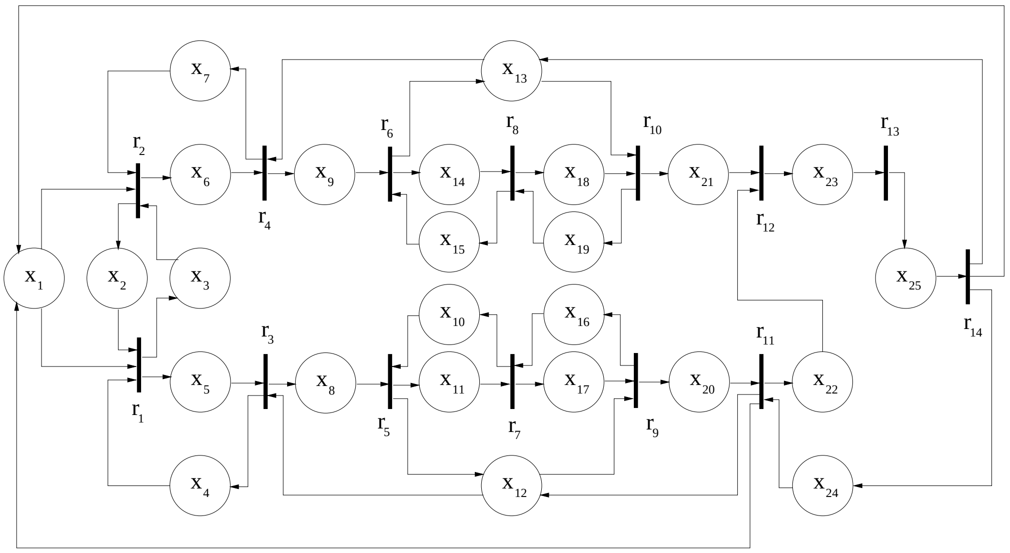


FIG. 30 – Système de production

Avec HyTech, nous obtenons l'ensemble $Post^*(\varphi_0)$ sous la forme (d'une disjonction) de 560 régions de la forme $(p = x_1 + d_0 \wedge x_1 \geq c_1 \wedge \bigwedge_{i=2}^{25} x_i = c_i)$ ou $(p = d_0 \wedge x_1 = c_1 \wedge \bigwedge_{i=2}^{25} x_i = c_i)$, avec $c_i \in \{0, 1, \cdots, 12\}$ pour $0 \leq i \leq 25$. L'intersection de $Post^*$ avec la formule caractérisant les blocages produit $p = x_1 + 9 \ \wedge \ x_1 \geq 0 \wedge x_{11} = 3 \wedge x_{15} = 2 \wedge x_3 = x_4 = x_6 = x_8 = x_{14} = x_{17} = x_{18} = x_{21} = x_{24} = 1$, les autres variables étant nulles. On retrouve donc la condition nécessaire et suffisante $(p \leq 8)$ de l'absence de blocage.

Le réseau de Petri de la figure 31, extrait de [Fin93], représente une partie du protocole de communication PNCSA [GIE]. Il comporte 31 places et 38 transitions, avec un marquage initial φ_0 pour lequel toutes les variables sont nulles sauf $E = F = 1$. Le blocage, qui s'exprime par une disjonction $\vee_{i=1}^{4} \varphi_i$, n'avait pas été mis en évidence jusque là. Il est obtenu en calculant $Upre(\varphi_1)$, de façon analogue à $UPost(\varphi_0)$, sur le réseau initial étendu par des fusions de transitions. On vérifie ensuite que $Upre(\varphi_1) = Pre^*(\varphi_1)$ et que cette formule contient φ_0 parmi les 375 contraintes qui la composent. On peut aussi exhiber un chemin de φ_0 à φ_1 de la forme $(a_1 a_{14} a_{17} a_{21} a_{34} a_{37})^{n_1} (a_1 a_2 a_{13} a_{17} a_{21} a_{22} a_{31} a_{37})^{n_2} (a_1 a_{14} a_{17} a_{21} a_{34} a_{37})^{n_3} (a_1 a_2 a_3)^{n_4} (a_{21} a_{34} a_{37})^{n_5}$, pour des entiers strictement positifs n_i, $1 \leq i \leq 5$. Avec ces entiers tous égaux à 1, on produit une instance de φ_1 dans laquelle $A = 3$, $B = 4$, $F = K = M = O = P = R = 1$ et tous les autres compteurs sont nuls.

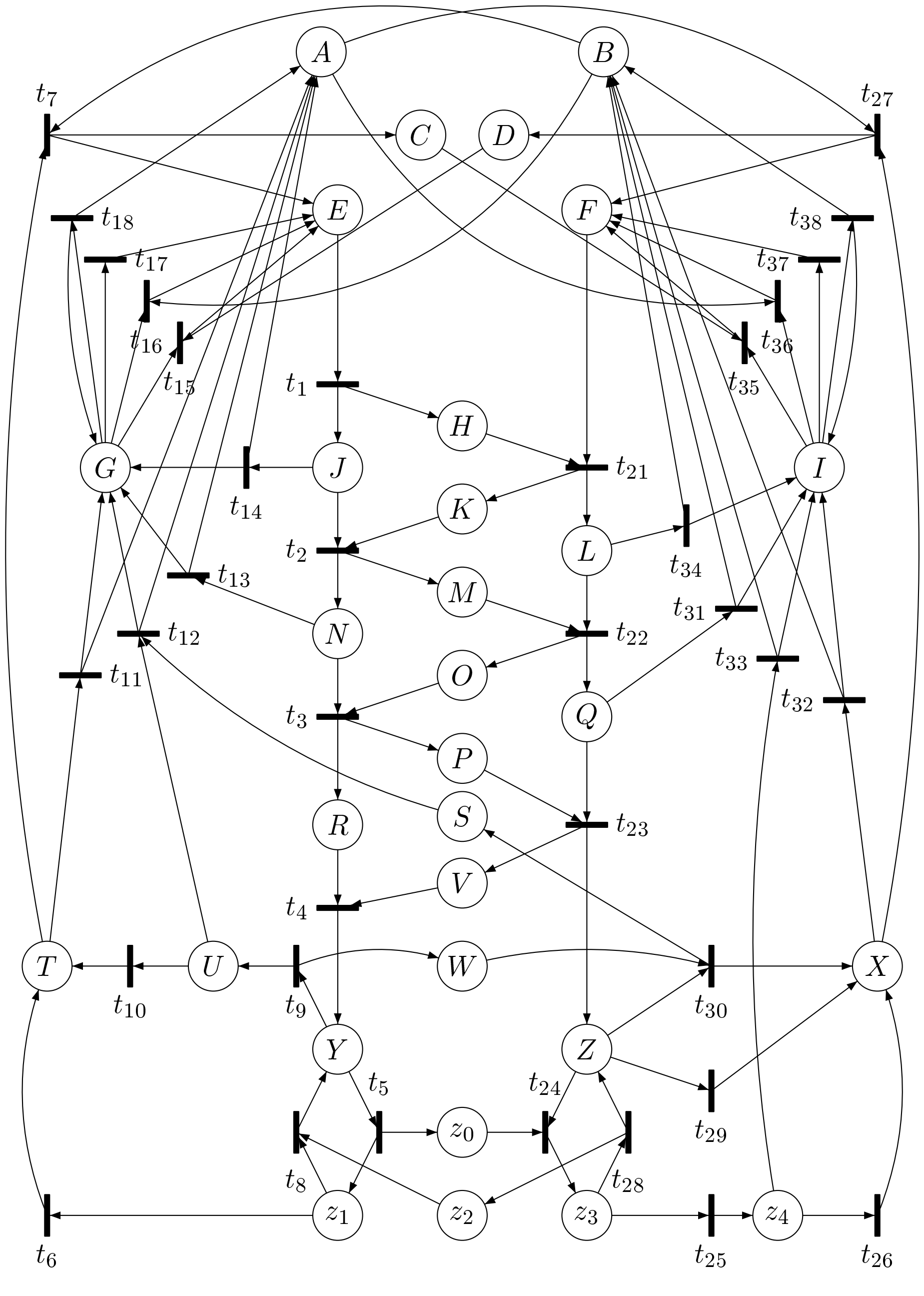


Fig. 31 – Protocole PNCSA

8 Conclusion et perspectives

La vérification de systèmes temporisés est un domaine riche d'aspects théoriques et pratiques, qui est encore largement susceptible de s'étendre dans les années qui viennent, compte tenu des développements industriels récents, par exemple sur les systèmes embarqués. Si les travaux des dix dernières années sur les modèles temporisés ont montré leur puissance, à la fois en termes d'expressivité et de pratique de la vérification, plusieurs points méritent d'être encore approfondis, pour apporter une meilleure compréhension des mécanismes sous-jacents. Les questions peuvent être regroupées selon qu'elles concernent les modèles ou les applications.

Modèles. Par exemple, la notion de reconnaissabilité est bien établie pour les langages usuels, mais elle est moins claire en ce qui concerne les langages temporisés, où les séquences de dates réelles font apparaître des problèmes particuliers. Il serait intéressant d'examiner cet aspect, à la lumière des travaux récents, pour obtenir des caractérisations intrinsèques qui permettraient de mieux maîtriser la spécification des systèmes. Dans la même direction, il me semble nécessaire de poursuivre les comparaisons de modèles temporisés, amorcées avec L. Sierra. En effet, la diversité des outils existants, du point de vue de leurs spécifications, ne facilite pas le travail des utilisateurs qui souhaitent mener de multiples opérations de vérification. Avec P. Gastin et A. Petit, nous souhaitons étendre la définition des substitutions temporisées dans un cadre un peu élargi d'automates temporisés, dans lesquels coexistent des actions instantanées et des signaux qui persistent pour une certaine durée. Ce travail permettrait une synthèse des mécanismes de raffinement, qui ont jusqu'à présent été étudiés séparément dans différents contextes sémantiques. Enfin, entre le modèle simple des automates temporisés, décidable mais souvent insuffisamment expressif, et le modèle très général des automates hybrides, où la vérification peut ne pas aboutir, il reste un espace à explorer. C'est dans cet esprit qu'est conçue l'ACI-jeunes chercheurs, coordonnée par F. Laroussinie et F. Cassez (de l'IRCCyN, École Centrale de Nantes). Je compte participer à cette action, dans sa première phase où nous souhaitons isoler des familles particulières de systèmes temporisés, pour la modélisation de problèmes d'ordonnancement. Dans ces systèmes, les variables pertinentes sont des chronomètres qui fonctionnent comme des horloges, susceptibles d'être arrêtées pendant un certain temps, pour repartir ensuite. Une étude précise de problèmes pratiques d'ordonnancement pourrait mettre en évidence un modèle intermédiaire où la vérification reste possible.

Vérification. L'expérience acquise dans les différentes études de cas, qu'elles soient académiques ou industrielles, permet d'opérer un retour sur les méthodes utilisées, à deux points de vue. Les comparaisons d'outils font apparaître différentes techniques mises en oeuvre dans l'analyse des systèmes. Une étude fine des techniques à base de matrices (DBM), utilisées actuellement pour les automates temporisés, montre qu'elles peuvent, dans certains cas, être étendues aux systèmes avec chronomètres. Ainsi, une fois défini un modèle intermédiaire, nous envisageons de développer de nouveaux algorithmes applicables aux DBM. Enfin, un sujet important dans la pratique est l'adéquation d'un outil à une application donnée. Cet aspect de la vérification est encore assez peu étudié, les essais étant souvent faits par des équipes qui choisissent précisément l'outil qu'elles développent. Nous pensons qu'il faut intensifier le traitement d'exemples identiques par plusieurs outils, de façon à identifier des particularités des protocoles ou même des contextes, susceptibles de mieux convenir à certains outils. Cette question devient plus cruciale à une époque où de nouveaux prototypes sont proposés, fondés sur des méthodes très diverses.

Références

[ABBL98] L. Aceto, P. Bouyer, A. Burgueño, and K. G. Larsen. The power of reachability testing for timed automata. In *Proc. 18th Conf. Found. of Software Technology and Theor. Comp. Sci. (FST&TCS'98), Chennai, India, Dec. 1998*, volume 1530 of *Lecture Notes in Computer Science*, pages 245–256. Springer, 1998.

[ABBR92] A. Arnold, J. Beauquier, B. Bérard, and B. Rozoy. *Programmes parallèles, modèles et validation*. Armand Colin, 1992.

[ACH+92] R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill, and H. Wong-Toi. Minimization of timed transition systems. In *Proc. 3rd Int. Conf. Concurrency Theory (CONCUR'92), Stony Brook, NY, USA, Aug. 1992*, volume 630 of *Lecture Notes in Computer Science*, pages 340–354. Springer, 1992.

[ACH94] R. Alur, C. Courcoubetis, and T. A. Henzinger. The observational power of clocks. In *Proc. 5th Int. Conf. Theory of Concurrency (CONCUR'94), Uppsala, Sweden, Aug. 1994*, volume 836 of *Lecture Notes in Computer Science*, pages 162–177. Springer, 1994.

[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.

[ACM97] E. Asarin, P. Caspi, and O. Maler. A Kleene theorem for timed automata. In *Proc. 12th IEEE Symp. Logic in Computer Science (LICS'97), Warsaw, Poland, June–July 1997*, pages 160–171. IEEE Comp. Soc. Press, 1997.

[AD90] R. Alur and D. Dill. Automata for modeling real-time systems. In *Proc. 17th Int. Coll. Automata, Languages, and Programming (ICALP'90), Warwick University, England, July 1990*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990.

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

[AFH94] R. Alur, L. Fix, and T. A. Henzinger. A determinizable class of timed automata. In *Proc. 6th Int. Conf. Computer Aided Verification (CAV'94), Stanford, CA, USA, June 1994*, volume 818 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 1994.

[AH92a] R. Alur and T. A. Henzinger. Back to the future : towards a theory of timed regular languages. In *Proc. 33rd IEEE Symp. Foundations of Computer Science (FOCS'92)*, pages 177–186. IEEE Computer Society Press, 1992.

[AH92b] R. Alur and T. A. Henzinger. Logics and models of real time : A survey. In *Real-Time : Theory in Practice, Proc. REX Workshop, Mook, NL, June 1991*, volume 600 of *Lecture Notes in Computer Science*, pages 74–106. Springer, 1992.

[AH97] R. Alur and T. A. Henzinger. Real-time system = discrete system + clock variables. *Software Tools for Technology Transfer*, 1 :86–109, 1997.

[AHLP00] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas. Discrete abstractions of hybrid systems. *Proc. of the IEEE*, 88(7) :971–984, 2000.

[AHV93] R. Alur, T.A. Henzinger, and M. Vardi. Parametric real-time reasoning. In *Proceeding of the 25th Annual ACM Symp. on Theory of Computing (STOC'93)*, pages 592–601, 1993.

[AK96] R. Alur and R. P. Kurshan. Timing analysis in COSPAN. In *Proc. Workshop Hybrid Systems III : Verification and Control, New Brunswick, NJ, USA, Oct. 1995*, volume 1066 of *Lecture Notes in Computer Science*, pages 220–231. Springer, 1996.

[BBC+97] B. Barras, S. Boutin, C. Cornes, J. Courant, J.-C. Filliâtre, E. Giménez, H. Herbelin, G. Huet, C. Muñoz, C. Murthy, C. Parent, C. Paulin-Mohring, A. Saïbi, and B. Werner. The Coq proof assistant reference manual : version 6.1. Technical report, INRIA Rocquencourt, 1997.

[BBFM01] J. Beauquier, B. Bérard, L. Fribourg, and F. Magniette. Proving convergence of self-stabilizing systems using first-order rewriting and regular languages. *Distributed Computing*, 14(2) :83–95, 2001.

[Büc73] R. Büchi. The monadic second order theory of ω_1. In *The Monadic Second Order Theory of All Countable Ordinals*, volume 328 of *Lecture Notes in Mathematics*, pages 1–126. Springer, 1973.

[BC85] G. Berry and L. Cosserat. The Esterel synchronous programming language and its mathematical semantics. In *Proccedings of the CMU seminar on concurrency*, number 197 in Lecture Notes in Computer Science. Springer, 1985.

[BD00] B. Bérard and C. Dufourd. Timed automata and additive clock constraints. *Information Processing Letters*, 75(1–2) :1–7, 2000.

[BDFP00a] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Are timed automata updatable ? In *Proc. 12th Int. Conf. Computer Aided Verification (CAV'2000), Chicago, IL, USA, July 2000*, volume 1855 of *Lecture Notes in Computer Science*, pages 464–479. Springer, 2000.

[BDFP00b] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Expressiveness of updatable timed automata. In *Proc. 25th Int. Symp. Math. Found. Comp. Sci. (MFCS'2000), Bratislava, Slovakia, Aug. 2000*, volume 1893 of *Lecture Notes in Computer Science*, pages 232–242. Springer, 2000.

[BDGP98] B. Bérard, V. Diekert, P. Gastin, and A. Petit. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36(2) :145–182, 1998.

[BDM+98] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos : a model-checking tool for real-time systems. In *Proc. 10th Int. Conf. Computer Aided Verification (CAV'98), Vancouver, BC, Canada, June-July 1998*, volume 1427 of *Lecture Notes in Computer Science*, pages 546–550. Springer, 1998.

[Bed96] N. Bedon. Finite automata and ordinals. *Theoretical Computer Science*, 156 :119–144, 1996.

[BER94] A. Bouajjani, R. Echahed, and R. Robbana. Verification of context-free timed systems using linear hybrid observers. In *Proc. 6th Int. Conf. Computer Aided Verification (CAV'94), Stanford, CA, USA, June 1994*, volume 818 of *Lecture Notes in Computer Science*, pages 118–131. Springer, 1994.

[BF99] B. Bérard and L. Fribourg. Reachability analysis of (timed) Petri nets using real arithmetic. In *Proc. 10th Int. Conf. Concurrency Theory (CONCUR'99), Eindhoven, The Netherlands, Aug. 1999*, volume 1664 of *Lecture Notes in Computer Science*, pages 178–193. Springer, 1999.

[BFKM01] 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*, 2001. To appear.

[BGP97a] B. Bérard, P. Gastin, and A. Petit. Refinement and abstraction for timed languages. Research Report LSV-97-3, Lab. Specification and Verification, ENS de Cachan, Cachan, France, April 1997.

[BGP97b] T. Bultan, R. Gerber, and W. Pugh. Symbolic model-checking of infinite state systems using Presburger arithmetic. In *Proc. 9th Int. Conf. Computer Aided Verification (CAV'97), Haifa, Israel, June 1997*, volume 1254 of *Lecture Notes in Computer Science*, pages 400–411. Springer, 1997.

[BLL+98] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, Wang Yi, and C. Weise. New generation of UPPAAL. In *Proc. Int. Workshop on Software Tools for Technology Transfer (STTT'98), Aalborg, Denmark, July 1998*, pages 43–52, 1998. Proceedings available as BRICS Notes Series NS98/4.

[BLP+99] G. Behrmann, K. G Larsen, J. Pearson, C. Weise, Wang Yi, and J. Lind-Nielsen. Efficient timed reachability analysis using clock difference diagrams. In *Proc. 11th Int. Conf. Computer Aided Verification (CAV'99), Trento, Italy, July 1999*, volume 1633 of *Lecture Notes in Computer Science*, pages 341–353. Springer, 1999.

[BLS00] B. Bérard, A. Labroue, and Ph. Schnoebelen. Verifying performance equivalence for Timed Basic Parallel Processes. In *Proc. 3rd Int. Conf. Foundations of Software Science and Computation Structures (FOSSACS'2000), Berlin, Germany, Mar.-Apr. 2000*, volume 1784 of *Lecture Notes in Computer Science*, pages 35–47. Springer, 2000.

[BM01] J. C. M. Baeten and C. A. Middelburg. Process algebra with timing : real time and discrete time. In J. A. Bergstra, A. Ponse, and S. A. Smolka, editors, *Handbook of Process Algebra*, chapter 10, pages 627–684. Elsevier Science, 2001.

[BP99] P. Bouyer and A. Petit. Decomposition and composition of timed automata. In *Proc. 26th Int. Coll. Automata, Languages, and Programming (ICALP'99), Prague, Czech Republic, July 1999*, volume 1644 of *Lecture Notes in Computer Science*, pages 210–219. Springer, 1999.

[BP00] B. Bérard and C. Picaronny. Accepting Zeno words : a way toward timed refinements. *Acta Informatica*, 37(1) :45–81, 2000.

[BP01] P. Bouyer and A. Petit. A Kleene/Büchi-like theorem for clock languages. *Journal of Automata, Languages and Combinatorics*, 2001. To appear.

[BPT01] P. Bouyer, A. Petit, and D. Thérien. An algebraic characterization of data and timed languages. In *Proc. 12th Int. Conf. Concurrency Theory (CONCUR'2001), Aalborg, Denmark, Aug. 2001*, volume 2154 of *Lecture Notes in Computer Science*, pages 248–261. Springer, 2001.

[Bér95] B. Bérard. Untiming timed languages. *Information Processing Letters*, 55(3) :129–135, 1995.

[Bra83] G.W. Brams. *Réseaux de Petri : Théorie et Pratique*. Masson, 1983.

[BS97] S. Bornot and J. Sifakis. Relating time progress and deadlines in hybrid systems. In *Proc. Int. Workshop Hybrid and Real-Time Systems (HART'97), Grenoble, France, Mar. 1997*, volume 1201 of *Lecture Notes in Computer Science*, pages 286–300. Springer, 1997.

[BS98] S. Bornot and J. Sifakis. On the composition of hybrid systems. In *Proc. of Hybrid Systems : Computation and Control*, volume 1386 of *Lecture Notes in Computer Science*, pages 49–63. Springer, 1998.

[BW94] B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In *Proc. 6th Int. Conf. Computer Aided Verification (CAV'94), Stanford, CA, USA, June 1994*, volume 818 of *Lecture Notes in Computer Science*, pages 55–67. Springer, 1994.

[Cho78] Y. Choueka. Finite automata, definable sets and regular expressions over ω^n-tapes. *Journal of Computer and System Sciences*, 17 :81–97, 1978.

[CHPP87] P. Caspi, N. Halbwachs, D. Pilaud, and J. Plaice. Lustre : a declarative language for programming synchronous systems. In *Proceedings of the 14th Symp. on Principles of Programming Languages*, 1987.

[CJ98] H. Comon and Y. Jurski. Multiple counters automata, safety analysis, and Presburger arithmetic. In *Proc. 10th Int. Conf. Computer Aided Verification (CAV'98), Vancouver, BC, Canada, June-July 1998*, volume 1427 of *Lecture Notes in Computer Science*, pages 268–279. Springer, 1998.

[CR00] P. Castéran and D. Rouillard. Reasoning about parametrized automata. In *Proceedings of the 8th Conference on Real-Time Systems, mars 2000, Paris*, pages 107–119. Teknea, 2000.

[CS90] J. M. Colom and M. Silva. Convex geometry and semiflows in P/T nets. a comparative study of algorithms for computation of minimal p-semiflows. In *Advances in Petri*

Nets 1990, LNCS 483, volume 483 of *Lecture Notes in Computer Science*, pages 79–112. Springer, 1990.

[CX97] F. Chu and X.-L. Xie. Deadlock analysis of Petri nets using siphons and mathematical programming. *IEEE Trans. on Robotics and Automation*, 13(6) :793–804, 1997.

[CY92] C. Courcoubetis and M. Yannakakis. Minimum and maximum delay problems in real-time systems. *Formal Methods in System Design*, 1(4) :385–415, 1992.

[DA89] R. David and H. Alla. *Du Grafcet aux Réseaux de Petri*. Hermès, 1989.

[DGP97] V. Diekert, P. Gastin, and A. Petit. Removing ϵ-transitions in timed automata. In *Proc. 14th Ann. Symp. Theoretical Aspects of Computer Science (STACS'97), Lübeck, Germany, Feb.-Mar. 1997*, volume 1200 of *Lecture Notes in Computer Science*, pages 583–594. Springer, 1997.

[Dil89] D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proc. Int. Workshop Automatic Verification Methods for Finite State Systems (CAV'89), Grenoble, June 1989*, number 407 in Lecture Notes in Computer Science, pages 197–212. Springer, 1989.

[Dim00] C. Dima. Real-time automata and the Kleene algebra of sets of real numbers. In *Proc. 17th Ann. Symp. Theoretical Aspects of Computer Science (STACS'2000), Lille, France, Feb. 2000*, number 1770 in Lecture Notes in Computer Science, pages 279–289. Springer, 2000.

[DP01] G. Delzanno and A. Podelski. Constraint-based deductive model-checking. *International Journal on Software Tools for Technology Transfer*, 2001. to appear.

[DT98] C. Daws and S. Tripakis. Model-checking of real-time reachability properties using abstractions. In *Proc. 4th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98), Lisbon, Portugal, March 1998*, volume 1384 of *Lecture Notes in Computer Science*, pages 313–329. Springer, 1998.

[DZ98] F. Demichelis and W. Zielonka. Controlled timed automata. In *Proc. 9th Int. Conf. Concurrency Theory (CONCUR'98), Nice, France, Sep. 1998*, volume 1466 of *Lecture Notes in Computer Science*, pages 455–469. Springer, 1998.

[Fin93] A. Finkel. The minimal coverability graph for Petri nets. In *Advances in Petri Nets 1993*, volume 674 of *Lecture Notes in Computer Science*, pages 210–243. Springer, 1993.

[FO97] L. Fribourg and H. Olsén. Proving safety properties of infinite state systems by compilation into Presburger arithmetic. In *Proc. 8th Int. Conf. Concurrency Theory (CONCUR'97), Warsaw, Poland, Jul. 1997*, volume 1243 of *Lecture Notes in Computer Science*, pages 213–227. Springer, 1997.

[Fri98] L. Fribourg. A closed-form evaluation for extended timed automata. Research Report LSV-98-2, Lab. Specification and Verification, ENS de Cachan, Cachan, France, March 1998. 17 pages.

[GHJ97] V. Gupta, T. A. Henzinger, and R. Jagadeesan. Robust timed automata. In *Proc. Int. Workshop Hybrid and Real-Time Systems (HART'97), Grenoble, France, Mar. 1997*, number 1201 in Lecture Notes in Computer Science, pages 331–345. Springer, 1997.

[GIE] Protocole normalisé de connexion au système d'autorisation : spécifications PNCSA, version 2. Document C - Communication OSI, GIE CB, 1988.

[GL95] R.L. Grossman and R.G. Larson. An algebraic approach to hybrid systems. *Theoretical Computer Science*, 138 :101–112, 1995.

[GS94] R. Gorrieri and G. Siliprandi. Real-time system verification using P/T nets. In *Proc. 6th Int. Conf. Computer Aided Verification (CAV'94), Stanford, CA, USA, June 1994*, volume 818 of *Lecture Notes in Computer Science*, pages 14–26. Springer, 1994.

[Hal93] N. Halbwachs. Delay analysis in synchronous programs. In *Proc. 5th Int. Conf. Computer Aided Verification (CAV'93), Elounda, Greece, June 1993*, volume 697 of *Lecture Notes in Computer Science*, pages 333–346. Springer, 1993.

[Hen95] T. A. Henzinger. Hybrid automata with finite bisimulations. In *Proc. 22nd Int. Coll. Automata, Languages, and Programming (ICALP'95), Szeged, Hungary, July 1995*, volume 944 of *Lecture Notes in Computer Science*, pages 324–335. Springer, 1995.

[Hen96] T. A. Henzinger. The theory of hybrid automata. In *Proc. 11th IEEE Symp. Logic in Computer Science (LICS'96), New Brunswick, NJ, USA, July 1996*, pages 278–292. IEEE Comp. Soc. Press, 1996.

[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.

[HHW97] T. A. Henzinger, Pei-Hsin Ho, and H. Wong-Toi. HyTech : A model-checker for hybrid systems. *Journal of Software Tools for Technology Transfer*, 1(1–2) :110–122, 1997.

[HK97] T. A. Henzinger and P. W. Kopke. Discrete-time control for rectangular hybrid automata. In *Proc. 24th Int. Coll. Automata, Languages, and Programming (ICALP'97), Bologna, Italy, July 1997*, volume 1256 of *Lecture Notes in Computer Science*, pages 582–593. Springer, 1997.

[HKPV98] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata ? *Journal of Computer and System Sciences*, 57(1) :94–124, 1998.

[HKWT95] T. A. Henzinger, P. W. Kopke, and H. Wong-Toi. The expressive power of clocks. In *Proc. 22nd Int. Coll. Automata, Languages, and Programming (ICALP'95), Szeged, Hungary, July 1995*, number 944 in Lecture Notes in Computer Science, pages 335–346. Springer, 1995.

[HMP92] T. A. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks ? In *Proc. 19th Int. Coll. Automata, Languages, and Programming (ICALP'92), Vienna, Austria, July 1992*, number 623 in Lecture Notes in Computer Science, pages 545–558. Springer, 1992.

[HMP94] T. A. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies for timed transition systems. *Information and Computation*, 112(2) :273–337, 1994.

[HNSY94] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2) :193–244, 1994.

[HPZ95] M. R. Hansen, P. K. Pandya, and C. Zhou. Finite divergence. *Theoretical Computer Science*, 138 :113–139, 1995.

[HRS98] T. A. Henzinger, J.-F. Raskin, and P.-Y. Schobbens. The regular real-time languages. In *Proc. 25th Int. Coll. Automata, Languages, and Programming (ICALP'98), Aalborg, Denmark, July 1998*, volume 1443 of *Lecture Notes in Computer Science*, pages 580–591. Springer, 1998.

[HSB97] W. Harvey, P. J. Stuckey, and A. Borning. Compiling constraint solving using projection. In *Proc. 3rd Int. Conf. on Principles and Practice of Constraint Programming (CP'97), Schloss Hagenberg, Austria, Oct. 1997*, volume 1330 of *Lecture Notes in Computer Science*, pages 491–505. Springer, 1997.

[HW92] J. C. Hemmer and P. Wolper. Ordinal finite automata and languages. Technical report, University of Liege, 1992.

[ITU] Traffic control and congestion control in B-ISDN. ITU-T Recommendation I.371.1, 1997.

[JLHM91] M. Jaffe, N. Leveson, M. Heimdahl, and B. Melhard. Software requirements analysis for real-time process-control systems. *IEEE Trans. Software Eng.*, 17 :241–258, 1991.

[JLL77] N.D. Jones, L.H. Landweber, and Y.E. Lien. Complexity of some problems in Petri nets. *Theoretical Computer Science*, 4 :277–299, 1977.

[KPSY99] Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine. Decidable integration graphs. *Information and Computation*, 150(2) :209–243, 1999.

[Lab98] A. Labroue. Conditions de vivacité dans les automates temporisés. Technical Report LSV-98-7, Lab. Specification and Verification, ENS de Cachan, Cachan, France, Sept. 1998. 60 pages.

[Lar00] K. G. Larsen. Verification of timed and hybrid systems. In *Proc. 21st Int. Conf. Application and Theory of Petri Nets (ICATPN'2000), Aarhus, Denmark, June 2000*, volume 1825 of *Lecture Notes in Computer Science*, pages 39–42. Springer, 2000. Invited talk.

[LL98] F. Laroussinie and K. G. Larsen. CMC : A tool for compositional model-checking of real-time systems. In *Proc. IFIP Joint Int. Conf. Formal Description Techniques & Protocol Specification, Testing, and Verification (FORTE-PSTV'98), Paris, France, Nov. 1998*, pages 439–456. Kluwer Academic, 1998.

[LPY99] G. Lafferriere, G. J. Pappas, and S. Yovine. A new class of decidable hybrid systems. In *Proc. of Hybrid Systems : Computation and Control*, volume 1569 of *Lecture Notes in Computer Science*, pages 137–149. Springer, 1999.

[LPY01] G. Lafferriere, G. J. Pappas, and S. Yovine. Symbolic reachability computations for families of linear vector fields. *Journal of Symbolic Computation*, 32(3) :231–253, 2001.

[LS00] F. Laroussinie and Ph. Schnoebelen. The state explosion problem from trace to bisimulation equivalence. In *Proc. 3rd Int. Conf. Foundations of Software Science and Computation Structures (FOSSACS'2000), Berlin, Germany, Mar.-Apr. 2000*, volume 1784 of *Lecture Notes in Computer Science*, pages 192–207. Springer, 2000.

[Man94] Z. Manna. Beyond model checking. In *Proc. 6th Int. Conf. Computer Aided Verification (CAV'94), Stanford, CA, USA, June 1994*, volume 818 of *Lecture Notes in Computer Science*, pages 220–221. Springer, 1994.

[Min67] M. Minski. *Computation : Finite and Infinite Machines.* Prentice Hall, 1967.

[MK99] J.-F. Monin and F. Klay. Correctness proof of the standardized algorithm for ABR conformance. In *Proc. of the World Congress on Formal Methods in the Development of Computing Systems (FM'99), Toulouse, France, Sept. 1999*, volume 1708 of *Lecture Notes in Computer Science*, pages 662–681. Springer, 1999.

[MMP91] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In *Real-Time : Theory in Practice, Proc. REX Workshop, Mook, NL, June 1991*, volume 600 of *Lecture Notes in Computer Science*, pages 447–484, 1991.

[Mon98] J.-F. Monin. Proving a real time algorithm for ATM in Coq. In *Proc. Int. Workshop on Types for Proofs and Programs (TYPES'96), Aussois, France, Dec. 1996. Selected papers*, volume 1512 of *Lecture Notes in Computer Science*, pages 277–293. Springer, 1998.

[NSY92] X. Nicollin, J. Sifakis, and S. Yovine. Compiling real-time spedifications into extended automata. *IEEE Transactions on Software Engineering*, 18(9) :794–804, 1992.

[NSY93] X. Nicollin, J. Sifakis, and S. Yovine. From ATP to timed graphs and hybrid systems. *Acta Informatica*, 30(2) :181–202, 1993.

[OSY94] A. Olivero, J. Sifakis, and S. Yovine. Using abstractions for the verification of linear hybrid systems. In *Proc. 6th Int. Conf. Computer Aided Verification (CAV'94), Stanford, CA, USA, June 1994*, volume 818 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 1994.

[PS96] A. Pnueli and E. Shahar. A platform for combining deductive with algorithmic verification. In *Proc. 8th Int. Conf. Computer Aided Verification (CAV'96), New Brunswick, NJ, USA, July-Aug. 1996*, volume 1102 of *Lecture Notes in Computer Science*, pages 184–195. Springer, 1996.

[Pug92] W. Pugh. A practical integer algorithm for exact array dependence analysis. *Communications of the ACM*, 35(8) :102–114, 1992.

[Rab97] C. Rabadan. L'ABR et sa conformité. Technical report NT DAC/ARP/034, CNET Lannion, 1997.

[Ros82] J. G. Rosenstein. *Linear orderings.* Academic Press, New York, 1982.

[RSK00] M. Rusinowitch, S. Stratulat, and F. Klay. Mechanical verification of an ideal incremental ABR conformance algorithm. In *Proc. 12th Int. Conf. Computer Aided Verification (CAV'2000), Chicago, IL, USA, July 2000*, volume 1855 of *Lecture Notes in Computer Science*, pages 344–357. Springer, 2000.

[RSS95] S. Rajan, N. Shankar, and M.K. Srivas. An integration of model checking with automated proof checking. In *Proc. 7th Int. Conf. Computer Aided Verification (CAV'95), Liège, Belgium, July 1995*, volume 939 of *Lecture Notes in Computer Science*, pages 84–97. Springer, 1995.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming.* Wiley, 1986.

[Sha93] A.U. Shankar. An introduction to assertional reasoning for concurrent systems. *ACM Computing Surveys*, 25(3) :225–262, 1993.

[Sho81] R. Shostack. Deciding linear inequalities by computing loop residue. *Journal of the ACM*, 28(4) :769–779, 1981.

[SKR98] T.R. Shiple, J.H. Kukula, and R.K. Ranjan. A comparison of Presburger engines for EFSM reachability. In *Proc. 10th Int. Conf. Computer Aided Verification (CAV'98), Vancouver, BC, Canada, June-July 1998*, volume 1427 of *Lecture Notes in Computer Science*, pages 280–292. Springer, 1998.

[SY96] J. Sifakis and S. Yovine. Compositional specification of timed systems. In *Proc. 13th Ann. Symp. Theoretical Aspects of Computer Science (STACS'96), Grenoble, France, Feb. 1996*, volume 1046 of *Lecture Notes in Computer Science*, pages 347–359. Springer, 1996.

[TY96] S. Tripakis and S. Yovine. Analysis of timed systems based on time-abstracting bisimulations. In *Proc. 8th Int. Conf. Computer Aided Verification (CAV'96), New Brunswick, NJ, USA, July-Aug. 1996*, volume 1102 of *Lecture Notes in Computer Science*, pages 232–243. Springer, 1996. Long version in Formal Methods in System Design 18(1) :25-68, 2001.

[WB98] P. Wolper and B. Boigelot. Verifying systems with infinite but regular state spaces. In *Proc. 10th Int. Conf. Computer Aided Verification (CAV'98), Vancouver, BC, Canada, June-July 1998*, volume 1427 of *Lecture Notes in Computer Science*, pages 88–97. Springer, 1998.

[Woj85] J. Wojciechowski. Finite automata on transfinite sequences and regular expressions. *Fundamenta Informaticae*, 8.3-4 :379–396, 1985.

[YL93] M. Yannakakis and D. Lee. An efficient algorithm for minimizing real-time transition systems. In *Proc. 5th Int. Conf. Computer Aided Verification (CAV'93), Elounda, Greece, June 1993*, number 697 in Lecture Notes in Computer Science, pages 210–224. Springer, 1993.

[ZDD93] M.C. Zhou, F. Dicesare, and A.A. Desrochers. A hybrid methodology for synthesis of Petri net models for manufacturing systems. *IEEE Trans. on Robotics and Automation*, 8(3) :350–361, 1993.