Chapitre 4

Vérification par automates temporisés

4.1. Introduction

La vérification des systèmes réactifs, systèmes critiques ou des systèmes embarqués est un problème très important aujourd'hui. Les approches par *model checking* — la vérification automatique de modèles — se sont largement développées ces dernières années [SCH 01, CLA 99]. Cela suppose de modéliser le comportement du système à vérifier et d'énoncer la propriété de correction attendue sous la forme d'une propriété d'accessibilité, d'un automate ou d'une formule de logique temporelle. Ensuite on peut utiliser un *model checker* afin de vérifier si le modèle satisfait ou non la propriété.

Certains systèmes manipulent des aspects temps réel de manière explicite, ces contraintes sont de nature quantitative et concernent des durées ou des délais, comme par exemple des temps de réponse ou des *time-outs*. Les modèles classiques ainsi que les techniques de *model checking* ont été étendus à ce type de systèmes. En 1990, Alur et Dill ont notamment proposé le modèle des automates temporisés [ALU 94a, ALU 93a] pour décrire le comportement des systèmes intégrant des contraintes quantitatives sur le temps. Ces automates contiennent des horloges qui évoluent de manière continue avec le temps et qui mesurent ainsi les délais séparant les différentes actions du système modélisé. Des algorithmes de vérification ont été proposés pour ces modèles [ALU 93a, HEN 94, LAR 95]. Des formalismes de spécification, par exemple les logiques temporelles, ont aussi été étendus pour pouvoir énoncer des propriétés

Chapitre rédigé par Patricia BOUYER et François LAROUSSINIE.

temps réel sur les systèmes [ALU 94c, ALU 93a, ACE 02]. Enfin des outils de *model checking* ont été développés [YOV 97, LAR 97b] et appliqués avec succès sur des exemples issus du milieu industriel [BEN 02, TRI 98].

Dans ce chapitre, nous présentons le modèle des automates temporisés. Nous expliquons comment les procédures de décision sont obtenues sur ces systèmes ayant un nombre infini d'états. Nous mentionnons aussi plusieurs variantes de ces modèles (forme des contraintes, des mises à jour, etc.) et quelques extensions comme les systèmes hybrides linéaires [ALU 95, HEN 96]. Quelques sous-classes des automates temporisés présentant des propriétés particulières sont aussi décrites. Enfin nous terminons ce chapitre par une présentation des aspects algorithmiques de la vérification d'automates temporisés et par une présentation de l'outil Uppaal [LAR 97b].

4.2. Automates temporisés, exemples et sémantique

Les automates temporisés [ALU 94a] ont été introduits par R. Alur et D. Dill dans les années 1990. Il s'agit d'automates classiques munis d'horloges qui évoluent de manière continue et synchrone avec le temps. Chaque transition contient une garde (sur la valeur des horloges) décrivant quand la transition peut être exécutée et un ensemble d'horloges qui doivent être remises à zéro lors du franchissement de la transition. Chaque état de contrôle contient un invariant (une contrainte sur les horloges) qui peut restreindre le temps d'attente dans l'état et donc forcer l'exécution d'une transition d'action. Le domaine de temps peut être $\mathbb{N}$, l'ensemble des entiers naturels, ou bien $\mathbb{Q}_{\geq 0}$, l'ensemble des rationnels positifs, ou bien même $\mathbb{R}_{\geq 0}$, l'ensemble des réels positifs. Ici, nous supposerons que c'est $\mathbb{R}_{\geq 0}$, mais il faut remarquer que la plupart des résultats ne sont pas modifiés lorsque l'on considère $\mathbb{Q}_{\geq 0}$ ou $\mathbb{N}$.

4.2.1. *Quelques notations*

Soit X un ensemble d'horloges à valeur dans $\mathbb{R}_{\geq 0}$. Une valuation v pour X est une fonction ($v : X \to \mathbb{R}_{\geq 0}$) qui associe à chaque horloge x sa valeur $v(x)$. On note $\mathbb{R}_{\geq 0}^X$ l'ensemble des valuations pour X. Étant donné un réel $d \in \mathbb{R}_{\geq 0}$, on note $v + d$ la valuation qui associe à l'horloge x la valeur $v(x) + d$. Si r est un sous-ensemble de X, $[r \leftarrow 0]v$ représente la valuation v' définie par : $v'(x) = 0$ pour tout $x \in r$ et $v'(x) = v(x)$ pour $x \in X \backslash r$.

On note $\mathcal{C}(X)$ l'ensemble des contraintes d'horloges sur X, c'est-à-dire l'ensemble des combinaisons booléennes de contraintes atomiques de la forme $x \bowtie c$ avec $x \in X$, $\bowtie \in \{=, <, \leq, >, \geq\}$ et $c \in \mathbb{N}$. On désigne par $\mathcal{C}_{\lessdot}(X)$ la restriction de $\mathcal{C}(X)$ aux combinaisons positives ne contenant que des contraintes $x \leq c$ ou $x < c$. Les contraintes d'horloges s'interprètent de manière naturelle sur les valuations d'horloges : une valuation v satisfait une contrainte atomique $x \bowtie c$ lorsque $v(x) \bowtie c$, l'extension aux contraintes quelconques est alors immédiate. Lorsqu'une valuation v satisfait une contrainte g, on écrit $v \models g$.

4.2.2. *Automates temporisés, syntaxe et sémantique*

Formellement, un automate temporisé est défini comme suit :

DÉFINITION.– *Un* automate temporisé $\mathcal{A}$ *est un* 6*-uplet* $(L, \ell_0, X, \mathsf{Inv}, T, \Sigma)$ *où :*

– L *est un ensemble fini* d'états de contrôle *ou* localités*,*

– $\ell_0 \in L$ *est la* localité initiale*,*

– X *est un ensemble fini d'*horloges*,*

– $T \subseteq L \times \mathcal{C}(X) \times \Sigma \times 2^X \times L$ *est un ensemble fini de* transitions *;* $e = \langle \ell, g, a, r, \ell' \rangle \in T$ *représente une transition de* ℓ *vers* ℓ'*,* g *est la garde associée à* e*,* r *est l'ensemble d'horloges devant être remises à zéro et* a *est l'étiquette de* e*. On note aussi* $\ell \xrightarrow{g,a,r} \ell'$*,*

– $\mathsf{Inv} : L \to \mathcal{C}_{<}(X)$ *associe un* invariant *à chaque état de contrôle,*

– Σ *est un alphabet d'action.*

Un exemple d'automate temporisé est donné sur la figure 4.2.

Un état ou une configuration d'un automate temporisé est une paire $(\ell, v) \in L \times \mathbb{R}_{\geq 0}^X$ où ℓ représente l'état de contrôle courant et v une valuation pour les horloges. La sémantique d'un automate temporisé est définie sous la forme d'un système de transitions temporisé qui comporte des transitions d'action (étiquetées par un élément de Σ) et des transitions de temps (étiquetées par des durées réelles) :

DÉFINITION.– *Un* système de transition temporisé *(STT) est un quadruplet* $\mathcal{S} = (S, s_0, \to, \Sigma)$ *où* S *est un ensemble (éventuellement infini) d'états,* $s_0 \in S$ *est l'état initial et* $\to \subseteq S \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times S$ *est la relation de transition. La relation* $\to$ *vérifie trois propriétés suivantes : (1) si* $s \xrightarrow{0} s'$*, alors* $s = s'$*, (2) si* $s \xrightarrow{d} s'$ *et* $s' \xrightarrow{d'} s''$ *avec* $d, d' \in \mathbb{R}_{\geq 0}$*, alors* $s \xrightarrow{d+d'} s''$*, et (3) si* $s \xrightarrow{d} s'$ *avec* $d \in \mathbb{R}_{\geq 0}$*, alors pour tout* $0 \leq d' \leq d$*, il existe* $s'' \in S$ *tel que* $s \xrightarrow{d'} s''$*.*

Les trois conditions mentionnées ci-dessus sont classiques dans les systèmes temporisés (voir par exemple [YI 90]), elles expriment juste que le temps est continu et déterministe.

De manière standard, une exécution dans un STT est une suite de transitions consécutives. Un état $s \in S$ est dit atteignable dans $\mathcal{S}$ s'il existe une exécution menant de s_0 à s.

DÉFINITION.– *Soit* $\mathcal{A} = (L, \ell_0, X, \mathsf{Inv}, T, \Sigma)$ *un automate temporisé. La sémantique de* $\mathcal{A}$ *est définie par le STT* $\mathcal{S}_{\mathcal{A}} = (S, s_0, \rightarrow, \Sigma)$ *où :*

– $S = L \times \mathbb{R}_{\geq 0}^X$,

– $s_0 = (\ell_0, v_0)$ *avec* $v_0(x) = 0$ *pour tout* $x \in X$,

– *la relation de transition* $\rightarrow$ *correspond à deux types de transitions :*

- les transitions d'actions : $(\ell, v) \xrightarrow{a} (\ell', v')$ *si et seulement s'il existe* $\ell \xrightarrow{g,a,r} \ell' \in T$ *tel que* $v \models g$, $v' = [r \leftarrow 0]v$ *et* $v' \models \mathsf{Inv}(\ell')$.

- les transitions de temps : si $d \in \mathbb{R}_{\geq 0}$, $(\ell, v) \xrightarrow{d} (\ell, v + d)$ *si et seulement si* $v + d \models \mathsf{Inv}(\ell)$ [1].

Informellement, le système part de la configuration initiale (état de contrôle ℓ_0 et toutes les horloges à zéro), puis effectue alternativement deux types de transitions : les transitions d'actions si la valeur courante des horloges le permet (ce type de transition s'effectue de manière instantanée et certaines horloges peuvent alors être remises à zéro) et les transitions de temps qui augmentent toutes les horloges d'une même durée (les horloges sont synchrones) en respectant l'invariant associé à la localité courante.

Une exécution possible de l'automate temporisé de la figure 4.2 est : $(\ell_0, (0, 0)) \xrightarrow{2.67} (\ell_0, (2.67, 2.67)) \xrightarrow{a} (\ell_1, (2.67, 0)) \xrightarrow{1} (\ell_1, (3.67, 1)) \xrightarrow{b} (\ell_2, (3.67, 1)) \dots$ où le couple $(3.67, 1)$ représente la valuation v telle que $v(x) = 3.67$ et $v(y) = 1$.

Une exécution dans un automate temporisé peut aussi être vue comme un mot temporisé, c'est-à-dire une suite de paires (action, date). On peut la représenter par : $(\ell_0, v_0, t_0) \xrightarrow{a_1} (\ell_1, v_1, t_1) \xrightarrow{a_2} \dots \xrightarrow{a_n} (\ell_n, v_n, t_n)$ avec $t_i \in \mathbb{R}_{\geq 0}$, $t_0 = 0$ et $t_{i+1} \geq t_i$ pour tout i. La date t_i correspond à la date à laquelle l'action a_i a été effectuée. L'étape $(\ell_i, v_i, t_i) \xrightarrow{a_{i+1}} (\ell_{i+1}, v_{i+1}, t_{i+1})$ correspond à une attente de durée $t_{i+1} - t_i$ suivi du franchissement d'une transition étiquetée par l'action a_{i+1}, la valuation v_{i+1} est donc obtenue à partir de $v_i + (t_{i+1} - t_i)$ en mettant à zéro certaines horloges (selon la transition choisie). Le mot temporisé associé est alors $(a_1, t_1)(a_2, t_2) \dots$ Par exemple, le mot temporisé associé à l'exécution décrite ci-dessus est $(a, 2.67)(b, 3.67) \dots$

4.2.3. *Composition parallèle*

Il est possible de définir de manière classique une composition parallèle d'automates temporisés (ou de STT). Par exemple, on peut définir une synchronisation n-aire avec renommage. Si ce type d'opération est indispensable pour la modélisation de systèmes, elle n'ajoute pas d'expressivité sur le plan théorique : on peut toujours

1. Ce qui implique — étant donnée la forme des invariants — que $v + d' \models \mathsf{Inv}(q)$ pour tout $0 \leq d' \leq d$.

construire un automate produit ayant le même comportement (au sens de la bisimulation forte, voir la section 4.4) que la composition parallèle considérée.

4.3. Décidabilité de l'accessibilité

Dans cette partie, nous allons décrire une construction proposée dans [ALU 94a] pour décider l'accessibilité d'un état de contrôle dans un automate temporisé. Le principe de cette construction est d'abstraire les comportements des automates temporisés de telle sorte que tester l'accessibilité d'un état de contrôle dans un automate temporisé se réduira à tester l'accessibilité d'un état (ou d'un ensemble d'états) dans un automate fini.

La construction repose sur une relation d'équivalence d'index fini définie sur l'ensemble des configurations et telle qu'à partir de deux configurations équivalentes, les mêmes comportements soient possibles. Par mêmes comportements, on sous-entend que si, à partir d'une configuration, il est possible d'attendre (respectivement de franchir une transition), il est aussi possible de le faire à partir de l'autre configuration et d'arriver dans des configurations qui sont encore équivalentes. Remarquons que les durées précises d'attente ne sont pas nécessairement les mêmes : on cherche donc une relation de bisimulation « temps-abstrait » qui aurait un nombre fini de classes d'équivalence. Pour les automates temporisés, une telle relation existe, et est définie de la façon suivante. Deux configurations (ℓ, v) et (ℓ', v') sont équivalentes si $\ell = \ell'$ et si $v \equiv_M v'$ où M est la constante maximale apparaissant dans l'automate et $v \equiv_M v'$ lorsque pour toute horloge x,

1) $v(x) > M \Leftrightarrow v'(x) > M$,

2) si $v(x) \leq M$, alors $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$ et $\Big(\{v(x)\} = 0 \Leftrightarrow \{v'(x)\} = 0\Big)$[2],

et pour toutes les paires d'horloges (x, y),

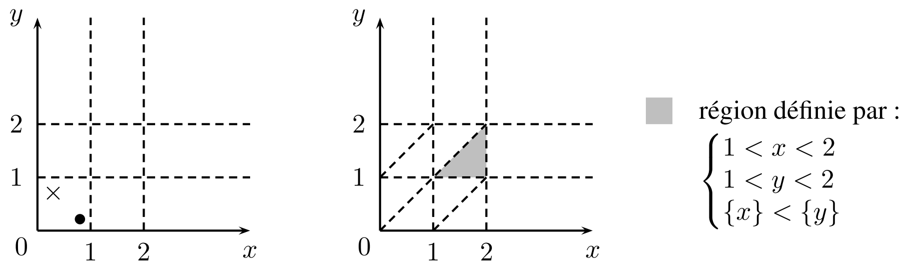
3) si $v(x) \leq M$ et $v(y) \leq M$, alors $\{v(x)\} \leq \{v(y)\} \Leftrightarrow \{v'(x)\} \leq \{v'(y)\}$.

Intuitivement, les deux premières conditions expriment que deux valuations équivalentes satisfont exactement les mêmes contraintes de l'automate. La dernière condition exprime qu'à partir de deux configurations équivalentes, l'écoulement du temps permettra aux horloges d'atteindre de nouvelles valeurs entières dans le même ordre. L'équivalence $\equiv_M$ que nous venons de définir est l'équivalence des régions, une classe d'équivalence étant alors appelée une région.

Nous illustrons cette construction sur la figure 4.1 dans le cas de deux horloges x et y, la constante maximale valant 2. La partition représentée sur la figure 4.1(a) respecte toutes les contraintes définies avec des constantes plus petites que 2, mais les

2. $\lfloor \alpha \rfloor$ représente la partie entière de α alors que $\{\alpha\}$ représente sa partie fractionnaire.

deux valuations • et $\times$ ne sont pas équivalentes vis-à-vis de l'écoulement du temps (point 3) ci-dessus) car si on laisse s'écouler du temps à partir de •, on va d'abord vérifier la contrainte $x = 1$ puis la contrainte $y = 1$, alors qu'à partir de $\times$, on va d'abord vérifier la contrainte $y = 1$ puis la contrainte $x = 1$. Les comportements possibles à partir de • et $\times$ ne sont donc pas les mêmes. La condition 3) raffine la partition donnée sur la figure 4.1(a) en rajoutant des « lignes diagonales » (représentant l'écoulement du temps) et donne la partition de la figure 4.1(b), qui va s'avérer être une relation de bisimulation « temps-abstrait ».



(a) Partition respectant 1) et 2)

(b) Partition respectant 1), 2) et 3)

Figure 4.1. *Ensemble de régions pour deux horloges avec la constante maximale qui vaut* 2

À partir de l'automate temporisé initial et de cette relation d'équivalence, nous construisons un automate fini de la façon suivante : les états de l'automate sont les paires (ℓ, R) où ℓ est un état de l'automate temporisé et R une région comme construite ci-dessus ; les transitions sont $(\ell, R) \xrightarrow{a} (\ell', R')$ s'il existe une transition $\ell \xrightarrow{g,a,r} \ell'$ dans $\mathcal{A}$, une valuation $v \in R$ et $t \geq 0$ tels que $v + t \models \mathsf{Inv}(\ell)$, $v + t \models g$, $[r \leftarrow 0](v + t) \models \mathsf{Inv}(\ell')$ et $[r \leftarrow 0](v + t) \in R'$.

L'automate fini résultant $\mathcal{R}_\mathcal{A}$ est appelé l'automate des régions associé à l'automate temporisé initial. La propriété fondamentale de cet automate fini est qu'il reconnaît exactement l'ensemble des mots $a_1 a_2 \ldots$ tels qu'il existe un mot temporisé $(a_1, t_1)(a_2, t_2) \ldots$ reconnu par l'automate temporisé initial. Ainsi, étant donné un automate temporisé $\mathcal{A}$ et son automate des régions $\mathcal{R}_\mathcal{A}$, on peut réduire le test du vide du langage reconnu par $\mathcal{A}$ ou les problèmes d'accessibilité d'un état de contrôle dans $\mathcal{A}$ à un problème d'accessibilité dans $\mathcal{R}_\mathcal{A}$. Cela fournit un algorithme pour ces deux problèmes :

THÉORÈME [ALU 90, ALU 94A].– *L'accessibilité d'un état de contrôle dans les automates temporisés est un problème PSPACE-complet.*

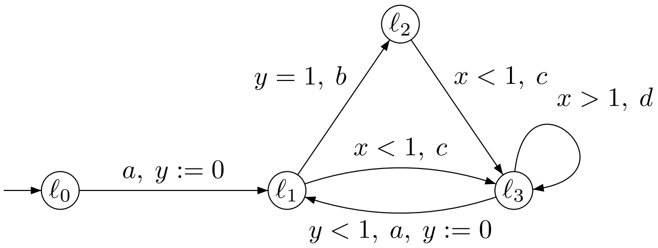


Figure 4.2. *Automate temporisé* $\mathcal{A}$

Nous illustrons la construction de l'automate des régions sur l'automate de la figure 4.2 [ALU 94a]. L'automate des régions qui lui est associé est dessiné sur la figure 4.3. Sur cet exemple, l'état ℓ_3 de $\mathcal{A}$ est accessible si et seulement si l'un des états (ℓ_3, R) où R est une région est accessible dans l'automate (fini) de la figure 4.3. Dans ce dernier automate, le chemin $(\ell_0, x = y = 0) \xrightarrow{a} (\ell_1, 0 = y < x < 1) \xrightarrow{c} (\ell_3, 0 < y < x < 1)$ mène à l'état ℓ_3, ce qui nous apprend que dans l'automate temporisé $\mathcal{A}$, il y a une exécution $(\ell_0, v_0) \xrightarrow{t_1} (\ell_0, v_0 + t_1) \xrightarrow{a} (\ell_1, v_1) \xrightarrow{t_2} (\ell_1, v_1 + t_2) \xrightarrow{c} (\ell_3, v_2)$ qui mène à l'état ℓ_3 (pour des réels t_1 et t_2).

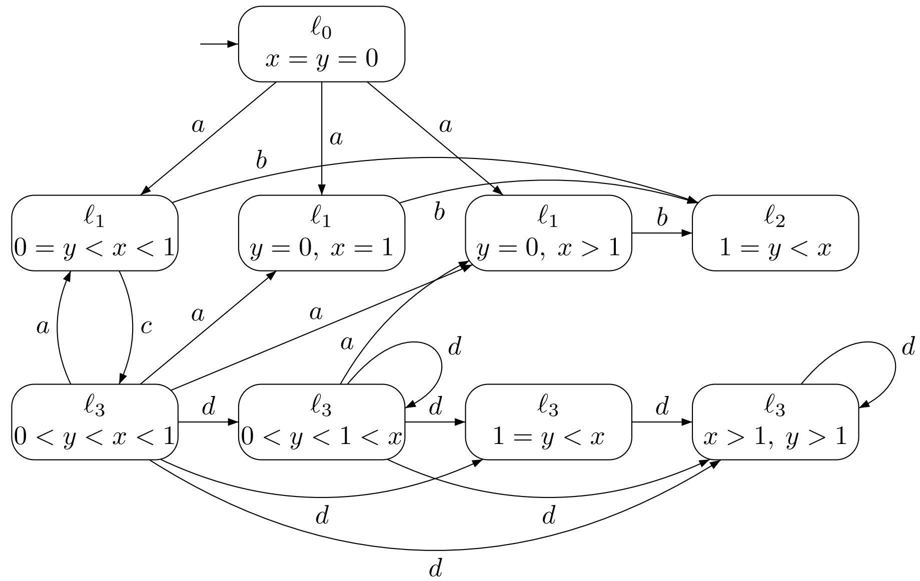


Figure 4.3. *Automate des régions associé à l'automate* $\mathcal{A}$

La complexité du problème de l'accessibilité, mentionnée dans le théorème précédent a été calculée dans le papier original [ALU 94a] :

– le côté PSPACE-dur vient de la possibilité de coder le comportement d'une machine de Turing $\mathcal{M}$ bornée linéairement en espace, sur un mot w. On peut en effet construire un automate temporisé dont la valeur des horloges représente l'état du ruban au cours de l'exécution. Notons que ce codage peut se faire avec trois horloges seulement [COU 92] ;

– le côté PSPACE-facile s'obtient en considérant un algorithme non déterministe qui stocke la configuration symbolique — une localité et une région — courante et une configuration successeur générée de manière non déterministe.

4.4. Autres problèmes de vérification

Souvent les problèmes de vérification se présentent sous la forme de problèmes d'accessibilité, ou plutôt de non-accessibilité d'un état d'erreur (dans le cas notamment des propriétés de sûreté). Néanmoins il est parfois utile de vérifier d'autres types de propriétés portant sur le comportement des systèmes. Par exemple, on peut vouloir énoncer la propriété temporisée suivante :

« L'alarme se déclenche *au plus* $3s$ *après* l'apparition d'un problème » [4.1]

Nous allons présenter plusieurs formalismes de spécification pour énoncer ce type de propriétés.

4.4.1. *Vérification et langages temporisés*

On a vu dans la partie précédente qu'on pouvait associer à une exécution dans un automate temporisé un mot temporisé, c'est-à-dire une séquence de paires (a, t) où a est une étiquette de transition et t l'instant où cette action a été effectuée. En ajoutant au modèle des automates temporisés une condition d'acceptation (états finals, conditions *à la* Büchi, Müller, etc.), il est possible de voir un automate temporisé comme une machine qui accepte un langage de mots temporisés. Si les comportements corrects du système correspondent au langage accepté par un automate temporisé, le problème de vérification peut alors se ramener à une question sur les langages temporisés : est-ce que le langage du système étudié est inclus dans celui de sa spécification ? Classiquement (dans un le cadre non temporisé), ce problème se résout en complémentant le langage de la spécification et en vérifiant que l'intersection du langage du système que l'on étudie et le langage de la spécification est vide. Malheureusement, en toute généralité, le complémentaire d'un langage accepté par un automate temporisé n'est pas reconnu par un automate temporisé, ce qui ne permet pas d'appliquer cette méthode. En outre, il a été montré dans [ALU 94a] que le problème d'inclusion de langages mentionné ci-dessus est en fait indécidable, et la vérification de telles propriétés temporisées ne pourra être faite que pour des classes restreintes de langages temporisés, comme les langages reconnus par des automates temporisés déterministes.

4.4.2. *Logique de temps arborescent*

Pour intégrer des contraintes quantitatives dans les logiques temporelles, on peut soit ajouter des contraintes aux modalités classiques, soit ajouter des horloges — on parle d'horloges de formule — et des opérateurs pour les manipuler.

La première possibilité consiste à compléter l'opérateur temporel Until (U) avec une contrainte de la forme « $\bowtie c$ » avec $\bowtie \in \{=,<,\leq,\geq,>\}$ et $c \in \mathbb{N}$. La formule $\varphi \mathsf{U}_{<c} \psi$ est vraie pour une exécution ρ si et seulement s'il existe un état, situé à moins de c unités de temps de l'état initial, vérifiant ψ et tel que tous les états précédents le long de ρ vérifient φ. De manière plus générale, il est possible d'associer un intervalle $[a;b]$ à un opérateur Until. Cette approche pour étendre les logiques temporelles est assez classique [KOY 90]. Pour CTL, elle a été proposée pour le temps discret [EME 92] et pour le temps dense [ALU 93a]. Formellement, on définit la logique TCTL (pour *Timed* CTL) à partir des propositions atomiques (étiquetant les états du système à vérifier), des opérateurs booléens ($\wedge$, $\vee$, $\neg$) et des modalités $\mathsf{E\_U}_{\bowtie c}\mathsf{\_}$ et $\mathsf{A\_U}_{\bowtie c}\mathsf{\_}$. La propriété [4.1] s'écrit alors :

$$\mathsf{AG}\Big(\,\texttt{problème} \;\Rightarrow\; \mathsf{AF}_{\leq 3}\,\texttt{alarme}\,\Big)$$

Une autre méthode pour intégrer des aspects quantitatifs dans les logiques temporelles consiste à ajouter des horloges de formules [ALU 94c] (on note H l'ensemble de ces horloges) qui augmentent de manière synchrone avec le temps, un opérateur de remise à zéro ($\underline{\texttt{in}}$) et des contraintes simples $x \bowtie c$ avec $x \in H$. La remise à zéro suivie, plus tard, d'une contrainte $x \bowtie c$ permet ainsi de mesurer le délai séparant deux états du système. Formellement, on définit TCTL_h à partir de CTL en ajoutant les contraintes $x \bowtie c$ avec $x \in H$ et l'opérateur $x\ \underline{\texttt{in}}\ \_$. La propriété [4.1] s'écrit comme suit :

$$\mathsf{AG}\Big(\texttt{problème} \Rightarrow \Big(x\ \underline{\texttt{in}}\ \Big(\mathsf{AF}(x \leq 3 \wedge \texttt{alarme})\Big)\Big)\Big)$$

L'opérateur $\underline{\texttt{in}}$ remet l'horloge x à zéro lorsque l'on rencontre un état « $\texttt{problème}$ », et il suffit donc de vérifier que $x \leq 3$ lorsque l'on rencontre un état « $\texttt{alarme}$ » pour s'assurer que le délai séparant ces deux positions est inférieur à 3.

Clairement, l'utilisation des horloges de formules permet d'exprimer tous les opérateurs de TCTL. On a l'équivalence suivante lorsque φ et ψ sont des formules de TCTL :

$$\mathsf{E}\left(\varphi \mathsf{U}_{\bowtie c}\,\psi\right) \;\equiv\; x\ \underline{\texttt{in}}\ \mathsf{E}\left(\varphi \mathsf{U}(\psi \wedge x \bowtie c)\right)$$

La logique TCTL_h permet d'exprimer des propriétés très fines, il a été montré récemment qu'elle était effectivement plus expressive que TCTL dans le temps dense [BOU 05] : l'argument repose sur le fait que la formule suivante n'a pas d'équivalent en TCTL :

$$x \ \underline{\mathtt{in}}\ \mathsf{EF}\Big(P_1 \wedge x < 1 \wedge \mathsf{EG}(x < 1 \Rightarrow \neg P_2)\Big)$$

Cette formule énonce qu'il est possible d'atteindre un état vérifiant P_1 en moins de 1 unité de temps, à partir duquel il y a une exécution où il n'y a pas d'état vérifiant P_2 avant que x ne soit égal à 1.

Nous avons le résultat suivant :

THÉORÈME [ALU 93a].– *Le model checking des logiques* TCTL *ou* TCTL_h *pour les automates temporisés est un problème PSPACE-complet.*

Les algorithmes procèdent par étiquetage d'un automate des régions étendu avec des horloges spéciales utilisées pour vérifier les contraintes temps réel dans les formules. L'outil Kronos permet de vérifier les formules de TCTL_h sur des compositions parallèles d'automates temporisés [YOV 97].

4.4.3. *Logique de temps linéaire*

Il est aussi possible d'étendre les logiques de temps linéaire pour énoncer des propriétés sur les exécutions des automates temporisés. La propriété [4.1] s'écrit alors « $\mathsf{G}(\mathtt{problème} \Rightarrow \mathsf{F}_{\leq 3}\ \mathtt{alarme})$ ». Parmi les logiques temporelles temporisées de temps linéaire, on peut notamment citer MTL [KOY 90, ALU 93b] qui associe des intervalles à la modalité U, ainsi que MITL [ALU 96] une restriction de MTL où les intervalles ne sont pas réduits à une unique valeur (c'est-à-dire la modalité $\mathsf{U}_{=c}$ n'est pas permise).

Le *model checking* de MTL est un problème indécidable lorsque l'on considère une sémantique où l'on observe continûment le système [ALU 96] : la valeur de vérité d'une proposition atomique est définie sur des intervalles de temps le long des exécutions. Le *model checking* est par contre décidable lorsque l'on considère une sémantique ponctuelle où ces propositions atomiques sont vraies à des instants [OUA 05]. Pour MITL, le *model checking* est un problème EXPSPACE-complet et devient même PSPACE-complet lorsque l'on se limite à des constraintes $< c$ ou $> c$ dans les modalités « Until » [ALU 96] .

Nous renvoyons à [ALU 92, ALU 93a, ALU 93b, HEN 94, RAS 99] pour une présentation détaillée de plusieurs logiques temporelles quantitatives.

4.4.4. *Logiques modales avec points fixes*

Les logiques modales peuvent aussi être étendues pour exprimer des propriétés temps réel. Comme dans la logique d'Hennessy et Milner [HEN 85], on peut définir des modalités portant sur les étiquettes des transitions du STT décrivant le comportement du système. On distingue la quantification existentielle, de la forme $\langle a \rangle \varphi$ (« il est possible de faire une transition a puis de vérifier φ »), et la quantification universelle, $[a]\ \varphi$ (« après toute transition étiquetée par a, φ est vraie »). On utilise le symbole δ pour les transitions de temps : $\langle \delta \rangle\ \varphi$ exprime ainsi qu'il est possible d'attendre un certain temps – sans effectuer de transition d'action – de manière à ce que φ soit vérifiée. Nous utilisons aussi des opérateurs de point fixe pour énoncer des propriétés portant sur des comportements non bornés [LAR 90, STI 01]. Les aspects quantitatifs, pour mesurer les délais séparant les actions du système étudié, sont traités à l'aide d'horloges de formule comme dans le cas de TCTL_h.

Le *model checking* pour les logique modales temporisées avec points fixes est un problème EXPTIME-complet [ACE 02].

4.4.5. *Automates de test*

On parle d'automate de test [ACE 98], lorsque l'on décrit un scénario d'erreur sous la forme d'un automate temporisé T_φ et que sa vérification se ramène à un problème d'accessibilité dans la composition parallèle de l'automate que l'on vérifie avec T_φ. Il est aussi possible de décrire la propriété de correction sous la forme d'une logique modale temporisée et de construire l'automate de test associé automatiquement. Notons que cette approche n'est possible que pour une classe restreinte de propriétés [ACE 03].

4.4.6. *Équivalences comportementales*

Les équivalences comportementales permettent aussi de comparer des systèmes temporisés. La bisimulation forte temporisée est une équivalence comportementale très souvent utilisée. Deux états (ℓ_1, v_1) et (ℓ_2, v_2) sont fortement bisimilaires (on écrit alors $(\ell_1, v_1) \approx (\ell_2, v_2)$) si et seulement si :

– pour toute transition $(\ell_1, v_1) \xrightarrow{a} (\ell_1', v_1')$ avec $a \in \Sigma$, il existe une transition $(\ell_2, v_2) \xrightarrow{a} (\ell_2', v_2')$ avec $(\ell_1', v_1') \approx (\ell_2', v_2')$, et *vice-versa* ;

– pour toute transition $(\ell_1, v_1) \xrightarrow{t} (\ell_1', v_1')$ avec $t \in \mathbb{R}_{\geq 0}$, il existe une transition $(\ell_2, v_2) \xrightarrow{t} (\ell_2', v_2')$ avec $(\ell_1', v_1') \approx (\ell_2', v_2')$, et *vice-versa*.

Cette équivalence comportementale est très forte, et en particulier, deux automates temporisés fortement bisimilaires où la relation de bisimulation respecte les états initiaux et finals acceptent les mêmes mots temporisés.

Notons que la notion de bisimulation « temps-abstrait » qui exprime la correction de l'automate des régions est beaucoup moins forte que la notion de bisimulation forte temporisée, car ici, les durées d'attente doivent être les mêmes à partir de deux états équivalents, ce qui n'est pas le cas pour la bisimulation « temps-abstrait ».

Décider si deux automates temporisés sont fortement bisimilaires est un problème EXPTIME-complet [LAR 00].

4.5. Quelques extensions des automates temporisés

Pour représenter plus facilement des systèmes, il est pratique de disposer de langages de description de haut niveau. Pour cette raison, des extensions des automates temporisés ont été considérées. Pour chacune de ces extensions, nous nous intéressons à plusieurs questions : 1) L'extension reste-t-elle décidable ? 2) L'extension ajoute-t-elle de l'expressivité ? 3) Ajoute-t-elle de la concision aux modèles que l'on peut construire ? Préserver la décidabilité du modèle est fondamental si l'on a pour but de vérifier les systèmes que l'on modélise (point 1). Il est aussi important de disposer de modèles permettant de représenter facilement de nombreux systèmes (le point 2 assure que l'on peut modéliser plus de systèmes et le point 3 porte sur la facilité de la modélisation : un modèle plus petit est souvent plus lisible et plus facilement modifiable).

4.5.1. *Les contraintes « diagonales »*

Dans le modèle des automates temporisés que nous avons présenté, les contraintes sur les horloges qui sont autorisées sont très simples et permettent uniquement de comparer la valeur d'une horloge à une constante. Dans le papier original [ALU 94a], un autre type de contraintes était évoqué, les contraintes dites diagonales, qui permettent de faire des tests du type $x - y \bowtie c$ où x et y sont des horloges, $\bowtie$ est un signe de comparaison et c est une constante entière. Le modèle des automates temporisés utilisant ce type de contraintes a les propriétés suivantes :

– l'accessibilité dans les automates temporisés avec des contraintes diagonales est un problème PSPACE-complet [ALU 94a] ;

– l'ajout des contraintes diagonales n'augmente pas l'expressivité des automates temporisés [BÉR 98] ;

– l'ajout des contraintes diagonales apporte de la concision [BOU 06].

La décidabilité (de l'accessibilité) de cette extension était déjà prouvée dans le papier original [ALU 94a], elle est aussi basée sur la construction d'un ensemble de régions, qui raffine celui que nous avons vu dans la partie 4.3 et la complexité n'augmente pas. La deuxième propriété (celle concernant l'expressivité) est bien connue, elle a été démontrée dans [BÉR 98]. Le principe est de retirer une à une les contraintes diagonales

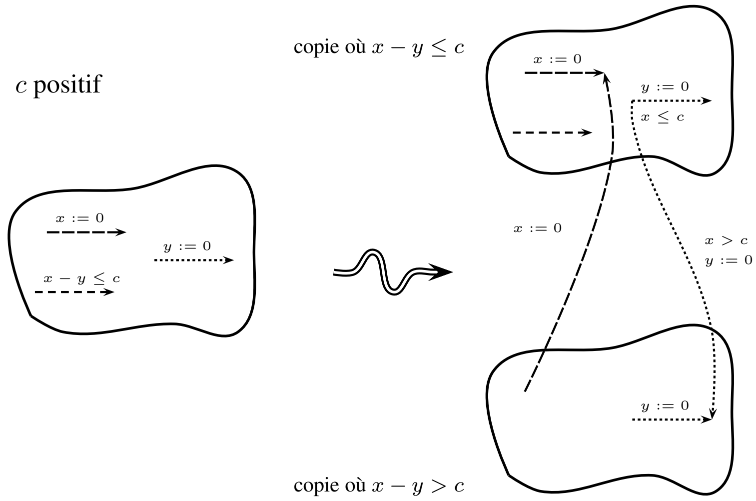


Figure 4.4. *Les contraintes diagonales sont retirées une à une*

et de construire un automate temporisé sans contraintes diagonales équivalent (l'équivalence est même la bisimulation forte temporisée). La construction qui consiste à retirer une contrainte diagonale est illustrée sur la figure 4.4 (dans ce dessin, on cherche à enlever un contrainte $x - y \leq c$ où c est un entier positif). Le principe de base est le suivant : la valeur de vérité d'une contrainte diagonale $x-y \leq c$ ne change pas lorsque le temps s'écoule, mais uniquement lorsque l'une des deux horloges impliquées dans la contrainte est remise à zéro. Ainsi, on construit deux copies de l'automate initial, dans l'une, la contrainte $x-y \leq c$ est vérifiée, alors que dans l'autre, c'est la contrainte $x-y > c$ qui est vérifiée. Lorsque x ou y est remise à zéro, on va dans l'une ou l'autre des copies selon la valeur de l'autre horloge. Par exemple, si l'on remet à zéro l'horloge y, on ira dans la copie où $x - y \leq c$ si $x \leq c$, alors qu'on ira dans la copie où $x-y > c$ si $x > c$. La construction que nous venons de décrire double la taille de l'automate, ce qui induit une construction exponentielle pour retirer toutes les contraintes diagonales. Ce coût exponentiel est inévitable, car les automates temporisés utilisant des contraintes diagonales sont exponentiellement plus concis que les automates temporisés classiques [BOU 06], ce qui signifie que des systèmes peuvent être modélisés avec des automates temporisés utilisant des contraintes diagonales exponentiellement plus petits que le seraient des automates temporisés classiques équivalents.

4.5.2. *Les contraintes additives*

D'autres types de contraintes peuvent être ajoutées aux automates temporisés, nous allons considérer les contraintes dites additives, à savoir celles de la forme $x + y \bowtie c$

où x et y sont des horloges, $\bowtie$ est un signe de comparaison et c est un entier positif. Cette extension des automates temporisés a été étudiée dans [BÉR 00], elle permet d'exprimer des langages qui ne sont pas reconnus par des automates temporisés classiques. L'automate de la figure 4.5 reconnaît un langage qu'aucun automate temporisé classique ne peut reconnaître : les actions a sont effectuées aux dates $\frac{1}{2}$, $\frac{3}{4}$, $\frac{7}{8}$, $\frac{15}{16}$, etc.

$$L^+ = \{(a^n, t_1 \dots t_n) \mid n \geq 1 \text{ and } t_i = 1 - \tfrac{1}{2^i}\}$$

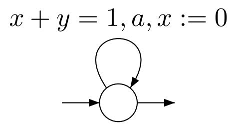


Figure 4.5. *Un langage non reconnu par un automate temporisé classique*

Ce modèle d'automates temporisés avec contraintes additives satisfait les propriétés suivantes [BÉR 00] :

– l'accessibilité dans les automates temporisés avec contraintes additives utilisant deux horloges est un problème décidable ;

– l'accessibilité dans les automates temporisés avec contraintes additives utilisant au moins quatre horloges est un problème indécidable.

La décidabilité du modèle utilisant deux horloges provient d'une extension de la construction de l'automate des régions, l'ensemble des régions utilisé étant un raffinement de l'ensemble des régions que nous avons vu jusqu'à présent. Cette construction est illustrée sur la figure 4.6. À partir de quatre horloges, le modèle devient indécidable. La preuve est assez compliquée mais intéressante, et utilise à plusieurs reprises le petit automate de la figure 4.5.

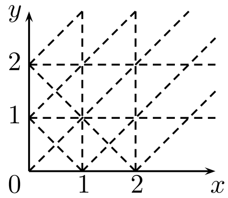


Figure 4.6. *Ensemble des régions pour les automates avec contraintes additives utilisant deux horloges*

Notons que, en ce qui concerne les automates temporisés avec contraintes additives et munis de trois horloges, la décidabilité de l'accessibilité est un problème ouvert, mais il a été montré qu'il n'est pas possible de leur associer un automate des régions fini comme cela est le cas pour les automates temporisés classiques [ROB 04].

4.5.3. *Les actions internes*

Dans les automates finis, il est bien connu que les actions internes (appelées dans ce contexte ε-transitions) peuvent être éliminées et n'apportent pas de pouvoir d'expression au modèle des automates finis (voir par exemple [HOP 79]). Le cadre temporisé est, étonnamment, bien différent : bien que les actions silencieuses n'influent pas sur la décidabilité de l'accessibilité (la construction de l'automate des régions se fait de la même façon), elles ajoutent de l'expressivité au modèle [BÉR 98]. L'automate de la figure 4.7 reconnaît un langage qu'aucun automate temporisé classique ne reconnaît. Ce langage est l'ensemble des mots temporisés sur une lettre a où toutes les dates sont entières et paires : à chaque unité de temps, soit la transition étiquetée par a est franchie, soit c'est celle qui est étiquetée par ε qui est franchie.

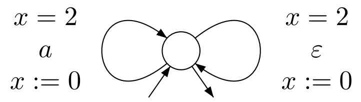


Figure 4.7. *Un langage non reconnu par un automate temporisé classique*

4.5.4. *Les mises à jour*

Dans le modèle original, les seules opérations autorisées sur les horloges sont les remises à zéro. Il est naturel de considérer des opérations un peu plus compliquées sur les horloges. Une mise à jour est une opération de la forme $x :\bowtie c$ ou $x :\bowtie y + c$ où x et y sont des horloges, $\bowtie$ est un signe de comparaison et c est un entier. Par exemple, la mise à jour $x :\leq c$ signifie que l'on affecte à x, de manière non déterministe, une valeur inférieure ou égale à la constante c ; la mise à jour $x := y - 1$ indique que l'on affecte à x la valeur de y décrémentée de 1. Les remises à zéro classiques correspondent aux mises à jour $x := 0$. Les automates temporisés utilisant ces mises à jour ont été étudiés dans [BOU 04b]. Il est assez aisé de voir que le modèle général est indécidable car les mises à jour autorisées sont très puissantes (on peut incrémenter, décrémenter et tester à zéro). Cependant, plusieurs sous-classes ont été montrées décidables. Voici quelques résutats marquants pour ces modèles [BOU 04b]. L'accessibilité dans les automates temporisés :

– avec mises à jour de la forme $x := c$ est décidable :

– avec autoincrémentation[3] et sans contrainte diagonale est décidable :

– avec autoincrémentation et contraintes diagonales est indécidable :

– avec autodécrémentation[4] est indécidable.

3. C'est-à-dire utilisant des mises à jour de la forme $x := x + 1$.

4. C'est-à-dire utilisant des mises à jour de la forme $x := x - 1$.

Là encore, les résultats de décidabilité ont été prouvés en utilisant une construction d'automates des régions avec des ensembles de régions assez fins et qui raffinent ceux que l'on a vus jusqu'à présent. Sur la figure 4.8 est dessiné un ensemble de régions pour un automate temporisé utilisant les contraintes d'horloges $\{x - y < 1, y > 1\}$ et les mises à jour $\{x := 0, y := 0, y := 1\}$. L'ensemble classique de régions serait la partition décrite avec des lignes en pointillés gras. Mais celui-ci n'est pas correct lorsque l'on utilise la mise à jour $y := 1$ à cause (entre autres) de l'image de la région grisée par la mise à jour $y := 1$ qui chevaucherait plusieurs régions (sans vérifier une propriété de bisimulation « temps-abstrait »). Il faut alors découper l'espace des valuations selon la droite $x = 2$ (ligne en pointillés fins sur la figure).

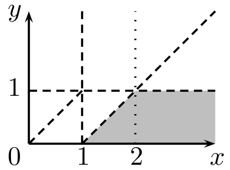


Figure 4.8. *Un ensemble de régions pour les automates temporisés utilisant les contraintes $\{x - y < 1, y > 1\}$ et les mises à jour $\{x := 0, y := 0, y := 1\}$*

Pour les automates temporisés avec autodécrémentation « $x := x - 1$ », l'accessibilité est indécidable. En effet, on peut facilement coder le comportement d'une machine à deux compteurs avec les automates temporisés utilisant ces mises à jour : la valeur du compteur C est stockée dans une horloge x_c, un incrément $C{+}{+}$ est modélisé en laissant écouler exactement une unité de temps puis en décrémentant de 1 l'horloge associée au second compteur, et l'opération $C{-}{-}$ est modélisée directement avec la mise à jour $x_c := x_c - 1$.

Notons que les classes d'automates temporisés avec mises à jour qui ont été prouvés décidables peuvent être transformés en automates temporisés classiques avec actions internes équivalents [BOU 04b][5]. Et d'autre part, ces automates sont aussi exponentiellement plus concis que les automates temporisés classiques [BOU 06].

Pour finir, on peut souligner que les mises à jour constituent des « macros » très pratiques pour la modélisation de systèmes réels. Nous pouvons par exemple citer des problèmes d'ordonnancement qui se modélisent de manière naturelle avec ce type d'opérations [FER 02].

5. La relation d'équivalence considérée est l'équivalence de langages.

4.5.5. *Les automates hybrides linéaires*

Les automates hybrides linéaires étendent les automates temporisés par :

– des contraintes linéaires générales (par exemple $3x_1 + 4x_2 - 2x_3 < 78$) ;

– des mises à jour plus riches que les simples remises à zéro : on peut associer des fonctions affines sur les variables X à chaque transition ;

– des pentes différentes pour les variables selon les états : au lieu d'avoir des horloges, on dispose de variables dynamiques.

En fait, chacune de ces extensions prise séparément rend déjà l'ensemble des problèmes de vérification indécidables ! Nous l'avons vu pour les contraintes additives et les mises à jour étendues. C'est aussi le cas pour les variables avec des pentes différentes : l'accessibilité est indécidable dans les automates où une variable peut avoir deux pentes différentes dans le système. Voir [HEN 98] ou plus récemment [RAS 05] pour un excellent *survey* sur ces questions, où sont aussi décrites des sous-classes décidables des automates hybrides linéaires.

Notons que la recherche de sous-classes des automates hybrides linéaires est une tâche importante. En effet, cela permet d'isoler des familles décidables (par exemple les automates rectangulaires initialisés [HEN 98]) ou des familles pour lesquelles des optimisations sont possibles. De plus, en pratique, certaines sous-classes sont particulièrement intéressantes, c'est le cas de la famille des p-automates [BÉR 99] pour la description de protocoles de télécommunication, ou des automates à chronomètres (c'est-à-dire munis de variables à pentes dans $\{0, 1\}$) pour les problèmes d'ordonnancement avec préemption.

Il n'existe donc pas, pour les automates hybrides linéaires, d'algorithmes pour vérifier les propriétés d'accessibilité ou plus généralement pour le *model checking*. Il est néanmoins possible de définir des semi-algorithmes (c'est-à-dire des procédures de calcul qui peuvent ne pas terminer) ou des approximations qui garantissent la terminaison mais qui ne fournissent pas toujours un résultat[6].

Le point-clé de ces approches repose sur l'usage des contraintes linéaires pour représenter les ensembles d'états du système à vérifier [ALU 95]. Cela permet d'utiliser les bibliothèques de calcul sur polyèdres pour implémenter des semi-algorithmes d'accessibilité. HyTech est un outil remarquable qui permet le calcul pas-à-pas des espaces d'états de systèmes hybrides linéaires. Il est aussi possible de lancer des calcul de points fixes (sans garantie de terminaison). Voir [HEN 97] pour une description complète et des exemples. L'outil HyTech ainsi qu'une documentation peuvent être téléchargés à l'adresse `http://www-cad.eecs.berkeley.edu/~tah/HyTech/`.

6. Dans ce cas, le programme peut répondre $S \models \phi$, $S \not\models \phi$ ou « ne sait pas ».

4.6. Quelques sous-classes des automates temporisés

Nous présentons ici quelques restrictions des automates temporisés. Souvent l'idée qui les motive est de considérer des modèles moins expressifs afin de « gagner » certaines propriétés de décidabilité ou d'améliorer la complexité des algorithmes de vérification. Ce type de démarche correspond à la recherche du meilleur compromis entre une bonne expressivité et une algorithmique efficace.

Notons que cette approche est rendue nécessaire par la complexité provenant à la fois de la mise en parallèle de plusieurs composants (comme dans le cadre classique non-temporisé) et de la temporisation (les horloges), même si d'un point de vue théorique, vérifier une mise en parallèle d'automates temporisés n'est pas plus compliqué que vérifier un unique automate temporisé (le problème est toujours PSPACE). Si les BDD sont de bonnes structures de données pour appréhender l'explosion combinatoire due à la mise en parallèle et si les DBM (que nous allons rapidement décrire dans la section 4.7) permettent de représenter facilement les contraintes temps réel, il n'existe pas de structure de données pour combiner efficacement ces deux types d'explosion.

4.6.1. *Automates event-recording*

Ici, l'idée est d'associer à chaque lettre a de l'alphabet d'action (étiquetant les transitions de l'automate temporisé) une horloge x_a et d'imposer que x_a soit remise à zéro à chaque franchissement d'une transition étiquetée par a. On impose aussi que l'automate ne contienne que des horloges de cette forme. Ainsi, pour chaque transition, il y a une unique horloge remise à zéro. La valeur de l'horloge x_a correspond au délai écoulé depuis le franchissement du dernier a. Ces modèles, appelés *event-recording*, ont été étudiés dans [ALU 94b][7].

Une telle restriction n'améliore pas la complexité : le test du vide (et donc le *model checking* de TCTL par exemple) reste PSPACE-complet. Par contre, cette classe d'automates temporisés est déterminisable et les langages associés sont fermés par complémentaire. Tester l'inclusion de deux langages devient ainsi décidable.

4.6.2. *Automates à une ou deux horloges*

Dans [COU 92], il est montré que l'accessibilité dans les automates temporisés est un problème PSPACE-complet dès qu'il y a plus de trois horloges. La recherche

7. Une autre version – les *event predicting* – est aussi présentée dans ce papier : dans ce cas les horloges indiquent non pas le temps écoulé depuis le dernier a mais le temps restant avant le prochain a.

d'algorithmes plus efficaces a motivé l'étude du cas des automates à une ou deux horloges.

Les automates à une horloge (AT1H) constituent une classe particulièrement intéressante et vérifient les propriétés suivantes [LAR 04] :

– l'accessibilité d'un état de contrôle est un problème NLOGSPACE-complet (c'est-à-dire de la même classe de complexité que l'accessibilité dans un graphe non temporisé) ;

– il existe un algorithme en temps polynomial pour le *model checking* des AT1H avec des formules de $\mathsf{TCTL}_{\leq,\geq}$ (la restriction de TCTL aux formules où les contraintes « $= c$ » ne sont pas autorisées dans les modalités « Until ») ;

– le *model checking* de TCTL est un problème PSPACE-complet.

L'idée de l'algorithme en temps polynomial pour le *model checking* de $\mathsf{TCTL}_{\leq,\geq}$ sur un AT1H $\mathcal{A}$ consiste à calculer, pour chaque sous-formule φ de la formule à vérifier, un ensemble d'intervalles $\mathsf{Sat}[\ell, \varphi] = \bigcup_i \langle \alpha_i, \beta_i \rangle$ — avec $\langle \in \{[, (\}, \rangle \in \{],)\}$ et $\alpha_i, \beta_i \in \mathbb{N} \cup \{\infty\}$ — contenant les positions (ℓ, x) vérifiant φ. On peut montrer que le nombre d'intervalles de $\mathsf{Sat}[\ell, \varphi]$ est borné par $2 \cdot |\varphi| \cdot |T_{\mathcal{A}}|$.

Nous décrivons brièvement la procédure d'étiquetage pour $\mathsf{E}\,\varphi \mathsf{U}_{\leq c} \psi$. A partir de $\mathsf{Sat}[\ell, \varphi]$ et $\mathsf{Sat}[\ell, \psi]$, on construit un automate des régions simplifié (de taille polynomiale) : les états sont des paires (ℓ, γ) où γ est un intervalle homogène pour les valeurs de vérité de φ et ψ et pour la satisfaction des gardes des transitions de $\mathcal{A}$. À chaque état (ℓ, γ) on peut associer une fonction $f_{\ell,\gamma}$ qui définit la durée minimale pour atteindre ψ (le long d'un chemin vérifiant φ) pour chaque (ℓ, x) avec $x \in \gamma$. Ces fonctions $f_{\ell,\gamma}$ ont des formes particulières : (1) la fonction est constante sur γ lorsque le plus court chemin menant à ψ comporte une transition avec remise à zéro avant toute transition de temps), (2) elle est décroissante (pente -1) lorsque le plus court chemin comporte une transition de temps avant toute remise à zéro, ou (3) elle combine les deux formes précédentes, d'abord constante pour la première partie de γ puis décroissante. Ces fonctions $f_{\ell,\gamma}$ se définissent par leurs valeurs aux bornes de γ et se calculent à l'aide d'un algorithme de plus court chemin (par exemple l'algorithme de Bellman-Ford [COR 90]). Il reste ensuite à construire une liste d'intervalles contenant les positions dont la distance à ψ est inférieure à c.

Les automates temporisés à une horloge bénéficient aussi de propriétés particulières du point de vue des langages temporisés : l'inclusion de langages temporisés de mots finis est décidable pour les AT1H [OUA 04] (mais reste indécidable pour les mots infinis [ABD 05]). De plus, si l'on considère des automates temporisés alternants, le test du vide est décidable si il n'y a qu'une seule horloge [LAS 05, OUA 05].

Concernant les automates temporisés à deux horloges, nous observons un saut de complexité important : l'accessibilité d'un état de contrôle est un problème NP-dur et le *model checking* de CTL (c'est-à-dire sans contrainte temps réel dans les formules) est déjà PSPACE-complet [LAR 04].

4.6.3. *Modèles à temps discret*

Plutôt que de considérer un domaine de temps dense, il est possible de considérer un domaine de temps discret comme les entiers naturels. Comme nous l'avons déjà dit, pour la plupart des problèmes vus jusqu'à présent, ce changement ne modifie pas substantiellement la complexité des algorithmes de vérification pour les automates temporisés, l'accessibilité et le *model checking* de TCTL restant PSPACE-complet. Afin d'obtenir des algorithmes plus efficaces avec le temps discret, il faut considérer des modèles plus simples. Il existe plusieurs solutions dans la littérature.

Tout d'abord, on peut considérer des modèles où l'on suppose que chaque transition a une durée de une unité de temps. Cette notion très simple de temporisation a été étudiée dans de nombreux travaux. Dans ce cadre, le *model checking* de TCTL est polynomial [EME 92], c'est aussi le cas dans le modèle où chaque transition fait 0 ou 1 unité de temps [LAR 03]. Ce résultat d'efficacité pour TCTL est unique. Dès que l'on considère la logique TCTL_h avec horloges de formules, on obtient des problèmes PSPACE-complet.

Une extension naturelle du modèle précédent consiste à étiqueter les transitions par des durées entières et voir les automates ainsi définis comme des automates à coûts ou des graphes valués. Dans ce cas, Il est possible d'obtenir des algorithmes de *model checking* polynomiaux pour $\mathsf{TCTL}_{\leq,\geq}$ tandis que le *model checking* de TCTL est Δ_2^p-complet [LAR 06]. Cet algorithme est implémenté dans l'outil TSMV [MAR 04], extension de l'outil NuSMV [CIM 00].

Enfin on peut considérer des automates temporisés à une horloge à valeur entière et obtenir des algorithmes polynomiaux pour $\mathsf{TCTL}_{\leq,\geq}$, mais le problème devient PSPACE-complet pour TCTL [LAR 06].

4.7. Algorithmique de la vérification

Dans cette partie, nous allons décrire les algorithmes implémentés dans des outils tels qu'Uppaal ou Kronos pour vérifier des automates temporisés. En effet, la construction de l'automate des régions que nous avons vue dans la section 4.3 de ce chapitre n'est pas utilisée en pratique car la notion de région est trop fine et rend les algorithmes inefficaces. Les outils utilisent une autre représentation symbolique (les zones) et s'appuient sur des algorithmes à la volée.

Il y a principalement deux familles de semi-algorithmes qui permettent d'analyser des systèmes à la volée. La première, appelée l'analyse en avant, consiste à calculer itérativement les successeurs des états initiaux et à tester si l'état que l'on cherche à atteindre est calculé ou pas. La seconde, en quelque sorte duale de la première, appelée l'analyse en arrière, consiste à calculer itérativement les prédécesseurs des états que l'on cherche à atteindre et à tester si un état initial se trouve dans l'ensemble calculé. Ces méthodes sont génériques et sont utilisées pour analyser de multiples systèmes, par exemple les automates hybrides [ALU 95].

Avant de décrire ces méthodes d'analyse, nous allons d'abord présenter la représentation symbolique la plus courante et la plus utilisée pour la vérification des systèmes temporisés.

4.7.1. *Les zones, une représentation symbolique pour les automates temporisés*

L'ensemble des configurations d'un automate temporisé est infini. Pour vérifier ce modèle, il est donc nécessaire de pouvoir manipuler des ensembles de configurations, et donc de disposer d'une représentation symbolique. La plus communément utilisée est la représentation symbolique appelée zone : une zone est un ensemble de valuations défini par une conjonction de contraintes simples $x \bowtie c$ ou $x-y \bowtie c$ où x et y sont des horloges, $\bowtie$ est un signe de comparaison, et c une constante entière. Dans les analyses en avant et en arrière, les objets qui seront manipulés seront des paires (ℓ, Z) où ℓ est un état de contrôle de l'automate et Z une zone.

Sur les zones, de multiples opérations peuvent être effectuées :

– le futur d'une zone Z, défini par $\overrightarrow{Z} = \{v + t \mid v \in Z \text{ et } t \in \mathbb{T}\}$;

– le passé d'une zone Z, défini par $\overleftarrow{Z} = \{v - t \mid v \in Z \text{ et } t \in \mathbb{T}\}$;

– l'intersection de Z et Z', définie par $Z \cap Z' = \{v \mid v \in Z \text{ et } v \in Z'\}$;

– la remise à zéro de r de Z, définie par $[r \leftarrow 0]Z = \{[r \leftarrow 0]v \mid v \in Z\}$;

– la relâche de r de Z, définie par $[r \leftarrow 0]^{-1}Z = \{v \mid [r \leftarrow 0]v \in Z\}$.

Ces opérations, définies grâce à des formules du premier ordre, préservent les zones (voir le principe d'élimination de variables de Fourier-Motzkin [SCH 98]).

Nous allons maintenant présenter l'analyse en arrière car c'est la méthode la plus simple. Nous verrons ensuite l'analyse en avant, qui est en fait la méthode la plus utilisée mais aussi la plus compliquée à mettre en œuvre.

4.7.2. *Analyse en arrière des automates temporisés*

L'analyse en arrière consiste à calculer les prédécesseurs des états que l'on cherche à atteindre, en commençant par les prédécesseurs en un pas, puis ceux en deux pas, etc., et à tester si l'état initial se trouve dans l'ensemble calculé. Si c'est le cas, cela signifie que les états recherchés sont accessibles à partir de l'état initial. Si le calcul a terminé mais que ce n'est pas le cas, cela signifie que ces états ne sont pas accessibles. Le principe de l'analyse en arrière est illustré sur la figure 4.9.

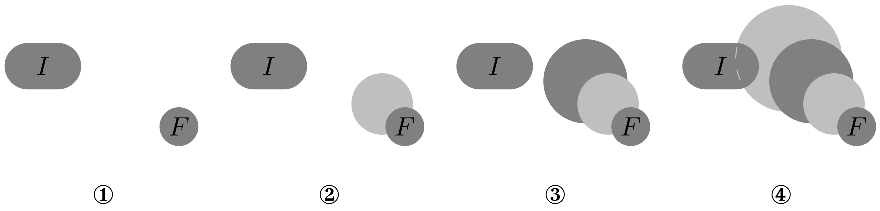


Figure 4.9. *Analyse en arrière : pas à pas, les prédécesseurs de l'état final sont calculés*

Un pas de l'analyse en arrière se calcule aisément à l'aide des zones. En effet, si $t = \ell \xrightarrow{g,a,r} \ell'$ est une transition de l'automate temporisé et si Z' est une zone, l'ensemble des prédécesseurs de (ℓ', Z') en un pas en prenant la transition t est l'ensemble des configurations (ℓ, v) où v est dans la zone $Z = \overleftarrow{g \cap [r \leftarrow 0]^{-1}(Z' \cap (r = 0))}$.

La particularité de l'analyse en arrière est que le calcul itératif décrit précédemment termine. La raison en est relativement simple : il est assez aisé de montrer que si Z' est une zone et que cette zone est une union de régions (voir la section 4.3), alors la zone Z que nous venons de décrire est non seulement une zone, mais aussi une union de régions. Comme il y a un nombre fini de régions, le nombre de paires (ℓ, Z) qui peuvent être calculées dans un calcul en arrière est fini.

Malgré les indéniables qualités de l'analyse en arrière, en pratique, elle est peu implémentée dans les outils, et l'analyse en avant lui est préférée. Les raisons en sont multiples : l'analyse en avant permet de visiter uniquement des états qui ont un sens dans le système car ils sont atteignables par des comportements réels ; d'autre part, l'analyse en arrière n'est pas propice à analyser des systèmes définis avec des structures de données de haut-niveau comme des variables entières bornées ou des morceaux de programmes C, etc. C'est en particulier le cas de l'outil Uppaal qui permet de manipuler ce type de structures de données, et qui implémente donc un algorithme d'analyse en avant. Nous reviendrons sur cet outil dans la partie 4.8.

4.7.3. *Analyse en avant des automates temporisés*

L'analyse en avant consiste à calculer les successeurs de l'état initial, en commençant par les successeurs en un pas, puis ceux en deux pas, etc., et à tester si les états recherchés se trouvent dans l'ensemble calculé. Si c'est le cas, cela signifie que certains de ces états sont accessibles à partir de l'état initial. Si le calcul a terminé mais que ce n'est pas le cas, cela signifie que ces états ne sont pas accessibles. Le principe de l'analyse en avant est illustré sur la figure 4.10.

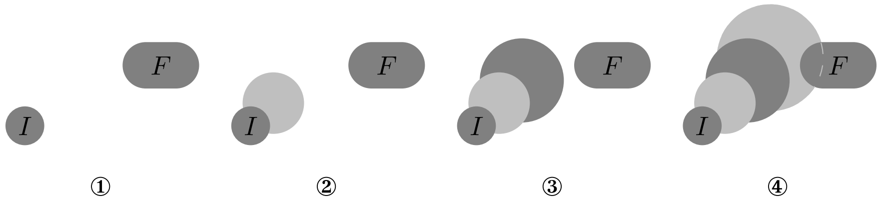


Figure 4.10. *Analyse en avant : pas à pas, les successeurs de l'état initial sont calculés*

Un pas de l'analyse en avant se calcule facilement à l'aide des zones. En effet, si $t = \ell \xrightarrow{g,a,r} \ell'$ est une transition de l'automate temporisé et si Z est une zone, l'ensemble des successeurs de (ℓ, Z) en un pas en prenant la transition t est l'ensemble des configurations (ℓ', v') où v' est dans la zone $Z' = [r \leftarrow 0](g \cap \overrightarrow{Z})$.

Contrairement au calcul en arrière, le calcul itératif en avant ne termine en général pas. Un exemple d'automate pour illustrer ce défaut de l'analyse en avant est donné sur la figure 4.11. Dans cet exemple, à chaque itération du calcul en avant, la valeur de x croît d'une unité, il ne va donc jamais terminer.

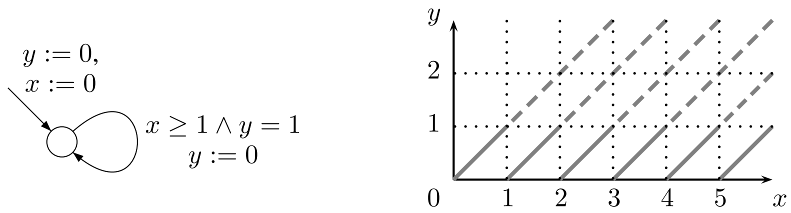


Figure 4.11. *Le calcul itératif en avant ne termine pas toujours*

Pour pallier ce problème de terminaison, un opérateur d'abstraction est en général appliqué à chaque étape du calcul. Dans la littérature, cet opérateur d'abstraction est appelé normalisation, ou extrapolation, ici nous l'appellerons extrapolation. On suppose que k est la plus grande constante qui apparaît dans les contraintes de l'automate temporisé. Si Z est une zone, l'extrapolation de Z par rapport à k est la plus petite zone contenant Z définie par des contraintes utilisant des constantes entre $-k$ et $+k$. L'intuition derrière cet opérateur d'extrapolation est la suivante : l'automate ne permettant de tester que des contraintes d'horloges bornées par k, si la valeur d'une horloge dépasse k, sa valeur précise n'a que peu d'importance, il est uniquement important de savoir qu'elle est plus grande que k. Il faut tout d'abord noter qu'appliquer cet opérateur d'extrapolation à chaque étape du calcul itératif en avant assure sa terminaison car il y a un nombre fini de zones définies par des contraintes n'utilisant que des constantes entières comprises entre $-k$ et $+k$. Cependant, un autre problème se pose : à chaque étape du calcul, c'est une surapproximation des états accessibles qui est calculée. Il se peut donc que des états soient déclarés comme accessibles par ce calcul alors qu'ils ne le sont pas en réalité.

Dans [BOU 04a], il est montré que ce calcul en avant est correct pour la classe des automates temporisés sans contrainte diagonale mais incorrect pour les automates avec contraintes diagonales.

4.7.4. *Les DBM, une structure de donnée pour les systèmes temporisés*

L'acronyme DBM signifie *Difference Bound Matrice*. C'est une structure de donnée assez classique qui permet de représenter des systèmes de contraintes de différences [COR 90], mais elle a un intérêt particulier pour la vérification des systèmes temporisés car elle permet de représenter les zones. Elle a d'abord été introduite pour analyser les réseaux de Petri temporels [BER 83], et elle est aujourd'hui intensivement utilisée pour l'analyse des automates temporisés [DIL 90].

Si n est le nombre d'horloges que l'on se fixe, une DBM M est une matrice carrée de taille $(n+1) \times (n+1)$ à coefficients entiers[8]. Si $M = (m_{i,j})_{0 \leq i,j \leq n}$, le coefficient $m_{i,j}$ représente la contrainte $x_i - x_j \leq m_{i,j}$ où $\{x_i \mid 1 \leq i \leq n\}$ est l'ensemble d'horloges et x_0 est une horloge fictive qui vaut toujours 0. Ainsi, pour représenter une contrainte $x_i \leq 6$, on écrira que $m_{i,0} = 6$ car cette contrainte est équivalente à $x_i - x_0 \leq 6$.

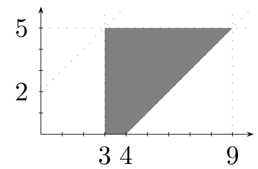


Figure 4.12. *La zone définie par les contraintes* $(x_1 \geq 3) \wedge (x_2 \leq 5) \wedge (x_1 - x_2 \leq 4)$

8. De manière générale, les coefficients d'une DBM sont des couples $(m, \prec)$ où m est un entier et $\prec$ est soit $<$, soit $\leq$, mais ici, pour simplifier, on va « oublier » les opérateurs de comparaison.

La DBM suivante représente l'ensemble de valuations défini par la contrainte $(x_1 \geq 3) \wedge (x_2 \leq 5) \wedge (x_1 - x_2 \leq 4)$ et représenté sur la figure 4.12 :

$$\begin{array}{cc} & \begin{array}{ccc} x_0 & x_1 & x_2 \end{array} \\ \begin{array}{c} x_0 \\ x_1 \\ x_2 \end{array} & \begin{pmatrix} +\infty & -3 & +\infty \\ +\infty & +\infty & 4 \\ 5 & +\infty & +\infty \end{pmatrix} \end{array}$$

Un coefficient $+\infty$ signifie qu'il n'y a pas de contrainte sur la différence d'horloges correspondante. Le coefficient $m_{0,1} = -3$ représente la contrainte $x_1 \geq 3$, car cette contrainte est équivalente à $x_0 - x_i \leq -3$.

Toute DBM représente une zone, et toute zone peut être représentée par une DBM. Cependant, une zone peut être représentée par plusieurs DBM (par exemple, dans la DBM précédente, si l'on remplace le coefficient $m_{1,0} = +\infty$ par $m_{1,0} = 9$, cela ne change pas la zone représentée). Mais il existe une forme normale pour les DBM, elle peut être calculée en appliquant un algorithme de plus courts chemins *à la* Floyd-Warshall [COR 90] : la DBM obtenue contient les plus fortes contraintes qui définissent cette zone. Pour l'exemple précédent, la DBM sous forme normale équivalente est :

$$\begin{pmatrix} 0 & -3 & 0 \\ 9 & 0 & 4 \\ 5 & 2 & 0 \end{pmatrix}$$

Toutes les opérations nécessaires à l'analyse en avant et l'analyse en arrière peuvent se faire avec les DBM (voir [BOU 04a] pour une description détaillée de l'utilisation des DBM).

Les DBM sont les structures de base pour manipuler les ensembles de configurations des automates temporisés, plusieurs améliorations sont possibles, on citera par exemple la minimisation des DBM [LAR 97a] ou l'utilisation des CDD (*« Clock Difference Diagrams »*) qui permettent de représenter de manière compacte des unions de DBM [LAR 99].

4.8. L'outil Uppaal

Uppaal est un outil de vérification pour les systèmes temporisés. Il a été développé conjointement par l'université d'Uppsala en Suède et l'université d'Aalborg au Danemark [LAR 97b]. Cet outil est téléchargeable à l'adresse `http://www.uppaal.com/`. Le modèle qui peut être vérifié par Uppaal est une variante des automates temporisés. Ce modèle est syntaxiquement très riche car il permet de représenter directement des contraintes d'urgence (une transition doit être prise immédiatement, sans attendre), des contraintes d'atomicité (une séquence de transitions doit être franchie instantanément), etc. Toutes ces facilités de modélisation n'ajoutent pas de réelle expressivité au modèle original mais permet d'écrire des modèles assez concis et lisibles.

Les propriétés que l'on peut vérifier avec l'outil Uppaal sont un sous-ensemble de TCTL$_h$ qui comprend notamment les propriétés d'accessibilité ou de sûreté, ainsi que des propriétés de réponse. Un tutoriel sur cet outil, mis à jour régulièrement, est disponible sur le site *web* mentionné ci-dessus [BEH 04].

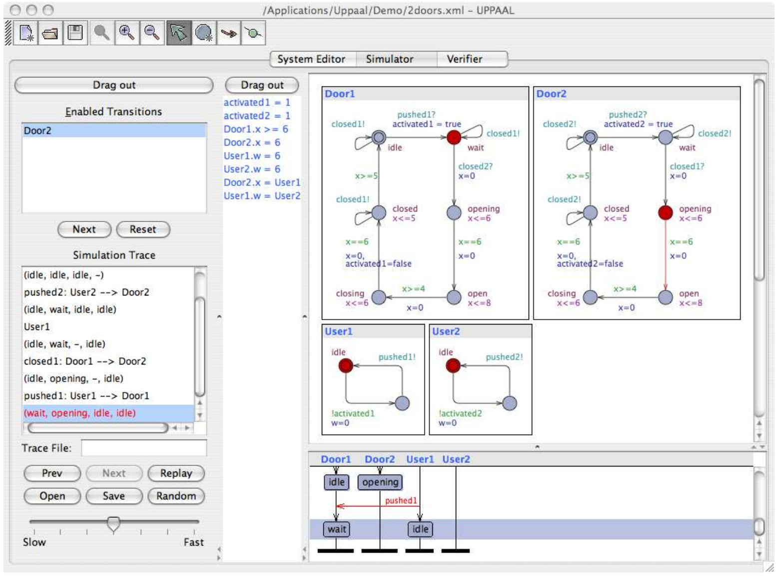


Figure 4.13. *L'outil* Uppaal

Outre l'interface de modélisation et le module de vérification, Uppaal comprend un module de simulation qui permet de voir comment le modèle que l'on a construit peut évoluer. Une capture d'écran de ce module est donnée sur la figure 4.13.

L'outil Uppaal est développé depuis maintenant un peu plus de dix ans et il a été utilisé avec succès pour la vérification de systèmes issus du milieu industriel. On peut, par exemple, évoquer des protocoles audio comme [BEN 02] ou celui utilisé par Bang & Olufsen dont l'analyse a permis la détection d'une erreur, une correction a alors pu être proposée et finalement le protocole corrigé a été validé [HAV 97].

4.9. Bibliographie

[ABD 05] ABDULLA P. A., DENEUX J., OUAKNINE J., WORRELL J., « Decidability and Complexity Results for Timed Automata via Channel Machines », *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, vol. 3580 de *Lecture Notes in Computer Science*, Springer, p. 1089–1101, 2005.

[ACE 98] ACETO L., BURGUEÑO A., LARSEN K. G., « Model-Checking via Reachability Testing for Timed Automata », *Proc. 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, vol. 1384 de *Lecture Notes in Computer Science*, Springer, p. 263–280, 1998.

[ACE 02] ACETO L., LAROUSSINIE F., « Is your Model-Checker on Time ? On the Complexity of Model-Checking for Timed Modal Logics », *Journal of Logic and Algebraic Programming*, vol. 52–53, p. 7–51, 2002.

[ACE 03] ACETO L., BOUYER P., BURGUEÑO A., LARSEN K. G., « The Power of Reachability Testing for Timed Automata », *Theoretical Computer Science*, vol. 300, n° 1–3, p. 411–475, 2003.

[ALU 90] ALUR R., DILL D., « Automata for Modeling Real-Time Systems », *Proc. 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, vol. 443 de *Lecture Notes in Computer Science*, Springer, p. 322–335, 1990.

[ALU 92] ALUR R., HENZINGER TH. A., « Logics and Models of Real-Time : A Survey », *Real-Time : Theory in Practice, Proc. REX Workshop 1991*, vol. 600 de *Lecture Notes in Computer Science*, Springer, p. 74–106, 1992.

[ALU 93a] ALUR R., COURCOUBETIS C., DILL D., « Model-Checking in Dense Real-Time », *Information and Computation*, vol. 104, n° 1, p. 2–34, 1993.

[ALU 93b] ALUR R., HENZINGER TH. A., « Real-Time Logics : Complexity and Expressiveness », *Information and Computation*, vol. 104, n° 1, p. 35–77, 1993.

[ALU 94a] ALUR R., DILL D., « A Theory of Timed Automata », *Theoretical Computer Science*, vol. 126, n° 2, p. 183–235, 1994.

[ALU 94b] ALUR R., FIX L., HENZINGER TH. A., « A Determinizable Class of Timed Automata », *Proc. 6th International Conference on Computer Aided Verification (CAV'94)*, vol. 818 de *Lecture Notes in Computer Science*, Springer, p. 1–13, 1994.

[ALU 94c] ALUR R., HENZINGER TH. A., « A Really Temporal Logic », *Journal of the ACM*, vol. 41, n° 1, p. 181–204, 1994.

[ALU 95] ALUR R., COURCOUBETIS C., HALBWACHS N., HENZINGER TH. A., HO P.-H., NICOLLIN X., OLIVERO A., SIFAKIS J., YOVINE S., « The Algorithmic Analysis of Hybrid Systems », *Theoretical Computer Science*, vol. 138, n° 1, p. 3–34, 1995.

[ALU 96] ALUR R., FEDER T., HENZINGER TH. A., « The Benefits of Relaxing Punctuality », *Journal of the ACM*, vol. 43, n° 1, p. 116–146, 1996.

[BEH 04] BEHRMANN G., DAVID A., LARSEN K. G., « A Tutorial on Uppaal », *Proc. 4th International School on Formal Methods for the Design of Computer, Communication and Software Systems : Real Time (SFM-04 :RT)*, vol. 3185 de *Lecture Notes in Computer Science*, Springer, p. 200–236, 2004.

[BEN 02] BENGTSSON J., GRIFFIOEN W. O. D., KRISTOFFERSEN K. J., LARSEN K. G., LARSSON F., PETTERSSON P., YI W., « Automated Verification of an Audio-Control Protocol using UPPAAL. », *Journal of Logic and Algebraic Programming*, vol. 52-53, p. 163-181, 2002.

[BER 83] BERTHOMIEU B., MENASCHE M., « An Enumerative Approach for Analyzing Time Petri Nets », *Proc. IFIP 9th World Computer Congress*, vol. 83 de *Information Processing*, North-Holland/ IFIP, p. 41–46, 1983.

[BÉR 98] BÉRARD B., DIEKERT V., GASTIN P., PETIT A., « Characterization of the Expressive Power of Silent Transitions in Timed Automata », *Fundamenta Informaticae*, vol. 36, n° 2–3, p. 145–182, 1998.

[BÉR 99] BÉRARD B., FRIBOURG L., « Automated Verification of a Parametric Real-Time Program : the ABR Conformance Protocol », *Proc. 11th International Conference on Computer Aided Verification (CAV'99)*, vol. 1633 de *Lecture Notes in Computer Science*, Springer, p. 96–107, 1999.

[BÉR 00] BÉRARD B., DUFOURD C., « Timed Automata and Additive Clock Constraints », *Information Processing Letters*, vol. 75, n° 1–2, p. 1–7, 2000.

[BOU 04a] BOUYER P., « Forward Analysis of Updatable Timed Automata », *Formal Methods in System Design*, vol. 24, n° 3, p. 281–320, 2004.

[BOU 04b] BOUYER P., DUFOURD C., FLEURY E., PETIT A., « Updatable Timed Automata », *Theoretical Computer Science*, vol. 321, n° 2–3, p. 291–345, 2004.

[BOU 05] BOUYER P., CHEVALIER F., MARKEY N., « On the Expressiveness of TPTL and MTL », *Proc. 25th Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS'05)*, vol. 3821 de *Lecture Notes in Computer Science*, Springer, p. 432–443, 2005.

[BOU 06] BOUYER P., CHEVALIER F., « On Conciseness of Extensions of Timed Automata », *Journal of Automata, Languages and Combinatorics*, 2006, À paraître.

[CIM 00] CIMATTI A., CLARKE E. E., GIUNCHIGLIA F., ROVERI M., « NuSMV : A New Symbolic Model Checker », *Software Tools for Technology Transfer*, vol. 2, n° 4, p. 410–425, 2000.

[CLA 99] CLARKE E., GRUMBERG O., PELED D., *Model-Checking*, The MIT Press, Cambridge, Massachusetts, 1999.

[COR 90] CORMEN TH. H., LEISERSON C. E., RIVEST R. L., *Introduction to Algorithms*, The MIT Press, Cambridge, Massachusetts, 1990.

[COU 92] COURCOUBETIS C., YANNAKAKIS M., « Minimum and Maximum Delay Problems in Real-Time Systems », *Formal Methods in System Design*, vol. 1, n° 4, p. 385–415, 1992.

[DIL 90] DILL D., « Timing Assumptions and Verification of Finite-State Concurrent Systems », *Proc. of the Workshop on Automatic Verification Methods for Finite State Systems (1989)*, vol. 407 de *Lecture Notes in Computer Science*, Springer, p. 197–212, 1990.

[EME 92] EMERSON E. A., MOK A. K., SISTLA A. P., SRINIVASAN J., « Quantitative Temporal Reasoning », *Real-Time Systems*, vol. 4, n° 4, p. 331–352, 1992.

[FER 02] FERSMAN E., PETTERSON P., YI W., « Timed Automata with Asynchrounous Processes : Schedulability and Decidability », *Proc. 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, vol. 2280 de *Lecture Notes in Computer Science*, Springer, p. 67–82, 2002.

[HAV 97] HAVELUND K., SKOU A., LARSEN K. G., LUND K., « Formal Modeling and Analysis of an Audio/Video Protocol : An Industrial Case Study Using Uppaal », *Proc. 18th IEEE Real-Time Systems Symposium (RTSS'97)*, IEEE Computer Society Press, p. 2–13, 1997.

[HEN 85] HENNESSY M., MILNER R., « Algebraic Laws for Nondeterminism and Concurrency », *Journal of the ACM*, vol. 32, n° 1, p. 137–161, 1985.

[HEN 94] HENZINGER TH. A., NICOLLIN X., SIFAKIS J., YOVINE S., « Symbolic Model-Checking for Real-Time Systems », *Information and Computation*, vol. 111, n° 2, p. 193–244, 1994.

[HEN 96] HENZINGER TH. A., « The Theory of Hybrid Automata », *Proc. 11th Annual Symposim on Logic in Computer Science (LICS'96)*, IEEE Computer Society Press, p. 278–292, 1996.

[HEN 97] HENZINGER TH. A., HO P.-H., WONG-TOI H., « HyTech : A Model-Checker for Hybrid Systems », *Journal on Software Tools for Technology Transfer*, vol. 1, n° 1–2, p. 110–122, 1997.

[HEN 98] HENZINGER TH. A., KOPKE P. W., PURI A., VARAIYA P., « What's Decidable about Hybrid Automata ? », *Journal of Computer and System Sciences*, vol. 57, n° 1, p. 94–124, 1998.

[HOP 79] HOPCROFT J. E., ULLMAN J. D., *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, 1979.

[KOY 90] KOYMANS R., « Specifying Real-Time Properties with Metric Temporal Logic », *Real-Time Systems*, vol. 2, n° 4, p. 255–299, 1990.

[LAR 90] LARSEN K. G., « Proof Systems for Satisfiability in Hennessy-Milner Logic with Recursion », *Theoretical Computer Science*, vol. 72, n° 2–3, p. 265–288, 1990.

[LAR 95] LARSEN K. G., PETTERSSON P., YI W., « Model-Checking for Real-Time Systems », *Proc. 10th International Conference on Fundamentals of Computation Theory (FCT'95)*, vol. 965 de *Lecture Notes in Computer Science*, Springer, p. 62–88, 1995.

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

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

[LAR 99] LARSEN K. G., PEARSON J., WEISE C., YI W., « Clock Difference Diagrams », *Nordic Journal of Computing*, vol. 6, n° 3, p. 271–298, 1999.

[LAR 00] LAROUSSINIE F., SCHNOEBELEN PH., « The State-Explosion Problem from Trace to Bisimulation Equivalence », *Proc. 3rd International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'00)*, vol. 1784 de *Lecture Notes in Computer Science*, Springer, p. 192–207, 2000.

[LAR 03] LAROUSSINIE F., SCHNOEBELEN PH., TURUANI M., « On the Expressivity and Complexity of Quantitative Branching-Time Temporal Logics », *Theoretical Computer Science*, vol. 297, n° 1, p. 297–315, 2003.

[LAR 04] LAROUSSINIE F., MARKEY N., SCHNOEBELEN PH., « Model Checking Timed Automata with One or Two Clocks », *Proc. 15th International Conference on Concurrency Theory (CONCUR'04)*, vol. 3170 de *Lecture Notes in Computer Science*, Springer, p. 387–401, 2004.

[LAR 06] LAROUSSINIE F., MARKEY N., SCHNOEBELEN PH., « Efficient Timed Model Checking for Discrete-Time Systems », *Theoretical Computer Science*, vol. 353, n° 1–3, p. 249–271, 2006.

[LAS 05] LASOTA S., WALUKIEWICZ I., « Alternating Timed Automata », *Proc. 8th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'05)*, vol. 3441 de *Lecture Notes in Computer Science*, Springer, p. 250–265, 2005.

[MAR 04] MARKEY N., SCHNOEBELEN PH., « Symbolic Model Checking of Simply-Timed Systems », *Proc. Joint Conference on Formal Modelling and Analysis of Timed Systems and Formal Techniques in Real-Time and Fault Tolerant System (FORMATS+FTRTFT'04)*, vol. 3253 de *Lecture Notes in Computer Science*, Springer, p. 102–117, 2004.

[OUA 04] OUAKNINE J., WORRELL J. B., « On the Language Inclusion Problem for Timed Automata : Closing a Decidability Gap », *Proc. 19th Annual Symposium on Logic in Computer Science (LICS'04)*, IEEE Computer Society Press, p. 54–63, 2004.

[OUA 05] OUAKNINE J., WORRELL J. B., « On the decidability of Metric Temporal Logic », *Proc. 19th Annual Symposium on Logic in Computer Science (LICS'05)*, IEEE Computer Society Press, p. 188–197, 2005.

[RAS 99] RASKIN J.-F., Logics, Automata and Classical Theories for Deciding Real-Time, Thèse de doctorat, Université de Namur, Belgique, 1999.

[RAS 05] RASKIN J.-F., « *An Introduction to Hybrid Automata* », Chapitre Handbook of Networked and Embedded Control Systems, p. 491–518, Springer, 2005.

[ROB 04] ROBIN A., Aux frontières de la décidabilité..., Mémoire de DEA, DEA Algorithmique, Paris, 2004.

[SCH 98] SCHRIJVER A., *Theory of Linear and Integer Programming*, Interscience Series in Discrete Mathematics and Optimization, Wiley, Chichester, UK, 1998.

[SCH 01] SCHNOEBELEN P., BÉRARD B., BIDOIT M., LAROUSSINIE F., PETIT A., *Systems and Software Verification - Model-Checking Techniques and Tools*, Springer, 2001.

[STI 01] STIRLING C., *Modal and Temporal Properties of Processes*, Texts in Computer Science, Springer, 2001.

[TRI 98] TRIPAKIS S., YOVINE S., « Verification of the Fast Reservation Protocol with Delayed Transmission using the Tool Kronos », *Proc. 4th IEEE Real-Time Technology and Applications Symposium (RTAS'98)*, IEEE Computer Society Press, p. 165–170, 1998.

[YI 90] YI W., « Real-Time Behaviour of Asynchronous Agents », *Proc. 1st International Conference on Theory of Concurrency (CONCUR'90)*, vol. 458 de *Lecture Notes in Computer Science*, Springer, p. 502–520, 1990.

[YOV 97] YOVINE S., « Kronos : A Verification Tool for Real-Time Systems », *Journal of Software Tools for Technology Transfer*, vol. 1, n° 1–2, p. 123–133, 1997.