Automates temporisés et hybrides

Modélisation et vérification

François Laroussinie

Laboratoire Spécification et Vérification,
ENS de Cachan & CNRS UMR 8643,
61, av. du Prés. Wilson, 94235 Cachan (F)
Email : fl@lsv.ens-cachan.fr

RÉSUMÉ. *L'objectif de ce document est de présenter le modèle des automates temporisés et des automates hybrides. Ces modèles défi nis dans les années 90 sont utilisés avec succès pour la vérifi cation formelle de système temps-réel. Nous présentons leur sémantique, rappelons les principaux résultats les concernant et donnons les références bibliographiques classiques de ce domaine de recherche.*

MOTS-CLÉS : *automate temporisé, système hybride, logique temporelle, model checking temporisé*

1. Introduction

Les *systèmes réactifs* et les *systèmes embarqués* occupent une place de plus en plus importante dans les applications industrielles. Ils interviennent de façon cruciale dans des domaines comme celui des transports (avionique, automobile, ferroviaire) ou du nucléaire. L'objectif de ces systèmes est de maintenir une interaction avec leur environnement. Ces systèmes peuvent présenter un aspect critique, au sens où certaines erreurs ont des conséquences graves : ils doivent alors être soumis à une phase de validation. Pour cela il est possible d'utiliser l'approche du *model checking* [35, 19], c'est à dire de construire un modèle M représentant le comportement du système étudié, de formaliser la propriété ϕ recherchée puis de calculer *automatiquement* si M satisfait ϕ ou non . Une des difficultés de la plupart de ces opérations réside dans le caractère dynamique de ces systèmes, qui ne produisent pas directement un résultat mais obéissent à des contraintes sur le déroulement de toutes leurs exécutions possibles. Certains manipulent des *aspects temporisés* de manière explicite, les contraintes sont souvent de nature quantitative et concernent des durées ou des délais, comme par exemple des temps de réponse.

En 1990, Alur et Dill ont proposé le modèle des *automates temporisés* [7, 3] 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. Les algorithmes de vérification ont été étendus à ces modèles [3, 27, 33, 28] et des outils de model checking ont été développés [37, 34, 29] et appliqués avec succès sur des exemples issus du milieu industriel [12, 36].

Les *automates hybrides* [5, 23] étendent cette démarche : il s'agit d'ajouter aux automates des variables dynamiques qui évoluent avec le temps. Le modèle ainsi obtenu est particulièrement riche (la plupart des problèmes de vérification sont indécidables !). Des méthodes et des outils d'analyse ont aussi été développées pour ces systèmes (semi-algorithmes, approximations etc.) [5, 25].

Des langages de spécification ont aussi été définis afin de permettre l'expression de propriétés intégrant des contraintes quantitatives sur les délais [8, 4, 3, 2].

Dans ce document, nous rappelons les principaux résultats sur ces modèles et nous donnons des références bibliographiques où des lecteurs intéressés pourront trouver plus d'informations.

2. Automates temporisés

2.1. *Définitions*

Les automates temporisés [7] ont été introduits par R. Alur et D. Dill dans les années 90. Il s'agit d'automates classiques munis d'horloges qui évoluent de manière continue 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. En général, le domaine de temps peut être $\mathbb{N}$ ou $\mathbb{R}_+$, ici nous prendrons $\mathbb{R}_+$ (N.B. la plupart des résultats ne sont pas modifiés lorsqu'on considère $\mathbb{N}$).

2.1.0.1. Quelques notations.

Soit X un ensemble d'horloges à valeur dans $\mathbb{R}_+$. Une valuation v pour X est une fonction ($v : X \to \mathbb{R}_+$) qui associe à chaque horloge x sa valeur $v(x)$. On note $\mathbb{R}_+^X$ l'ensemble des valuations pour X. Etant donné un réel $d \in \mathbb{R}_+$, 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, $v[r \leftarrow 0]$ 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 appelle *contrainte atomique rectangulaire* une formule de la forme $x \bowtie k$ avec $x \in X$, $k \in \mathbb{N}$ et $\bowtie \in \{=, <, \leq, >, \geq\}$. On appelle *contrainte atomique triangulaire* (ou *diagonale*) une formule de la forme $x - x' \bowtie k$. On note $\mathcal{C}(X)$ l'ensemble des combinaisons booléennes de contraintes atomiques (rectangulaires ou triangulaires) sur X.

Formellement, un automate temporisé est défini comme suit :

Définition 1 (Automate temporisé) *Un* automate temporisé A *est un 6-uplet* $(Q, q_0, X, \mathit{Inv}, \mathcal{T}, \Sigma)$ *avec :*

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

– $q_0 \in Q$ *est l'*état initial,

– X *est un ensemble fini d'*horloges *(à valeur réelle positive),*

– $\mathcal{T} \subseteq Q \times \mathcal{C}(X) \times \Sigma \times 2^X \times Q$ *est un ensemble fini de* transitions *;* $e = \langle q, g, a, r, q' \rangle \in \mathcal{T}$ *représente une transition de* q *vers* q'*,* 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* $q \xrightarrow{g,a,r} q'$*,*

– $\mathit{Inv} : Q \to \mathcal{C}(V)$ *associe un* invariant *à chaque état de contrôle,*

– Σ *est un alphabet d'action.*

Un exemple d'automate temporisé est donné Fig. 2.1.0.1. Cet automate a pour localité initiale q_0. L'invariant $x \leq 13$ de q_0 indique qu'on peut rester dans q_0 tant que x est inférieur à 13 mais pas au-delà. La transition a est exécutable lorsque $x < 4$ et son franchissement remet x et y à zéro.

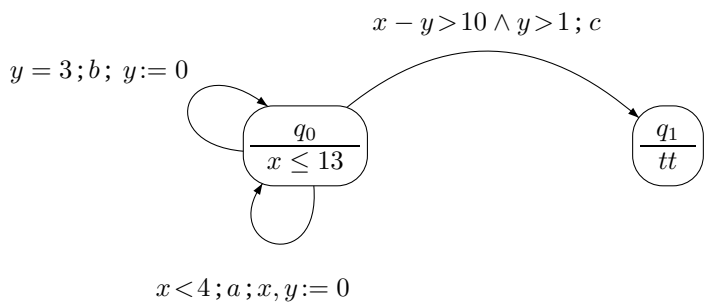


Figure 1. *Exemple d'automate temporisé*

Un *état* ou une *configuration* d'un automate temporisé est une paire $(q, v) \in Q \times \mathbb{R}^X$ où q 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 transition temporisé* qui comporte des transitions d'action (étiquetées par un élément de Σ) et des transitions de temps (étiquetées par des durées réelles) :

Définition 2 *Un système de transition temporisé (STT) $\mathcal{S}$ est un quadruplet $(S, s_0, \rightarrow, \Sigma)$ où S est un ensemble (infini) d'états, $s_0 \in S$ est l'état initial et $\rightarrow \subseteq S \times (\Sigma \cup \mathbb{R}_+) \times S$ est la relation de transition. La relation $\rightarrow$ vérifie deux propriétés de fermeture : (1) si $s \xrightarrow{d} s'$ et $s' \xrightarrow{d'} s''$ avec $d, d' \in \mathbb{R}_+$, alors $s \xrightarrow{d+d'} s''$; et (2) si $s \xrightarrow{d} s'$ avec $d \in \mathbb{R}_+$, alors $\forall 0 \leq d' \leq d$, $\exists s''$ t.q. $s \xrightarrow{d'} s''$.*

Une *exécution* ρ d'un STT $\mathcal{S}$ est une séquence de transitions $\rho = s_0 \xrightarrow{l_0} s_1 \cdots s_{n-1} \xrightarrow{l_{n-1}} s_n$ telle que $\forall 0 \leq i \leq n-1, s_i \xrightarrow{l_i} s_{i+1}$. Un état $q \in Q$ est *atteignable* dans S si il existe une exécution de q_0 à q.

Définition 3 *(Sémantique des automates temporisés) Soit A un automate temporisé $(Q, q_0, X, \mathit{Inv}, \mathcal{T}, \Sigma)$. La sémantique de A est définie par le STT $\mathcal{S}_A = (S, s_0, \rightarrow, \Sigma)$ avec :*

- *$S = Q \times \mathbb{R}^X$,*
- *$s_0 = (q_0, v_0)$ avec $v_0(x) = 0,\ \forall x \in X$,*
- *$\rightarrow$ correspond à deux types de transitions :*
  - *Les transitions d'actions : $(q, v) \xrightarrow{a} (q', v')$ ssi $\exists q \xrightarrow{g,a,r} q' \in \mathcal{T}$ et $v \models g$, $v' = v[r \leftarrow 0]$ et $v' \models \mathit{Inv}(q')$.*
  - *Les transitions de temps : Soit $d \in \mathbb{R}_+$. $(q, v) \xrightarrow{d} (q, v + d)$ ssi $v + d' \models \mathit{Inv}(q)$ pour tout $0 \leq d' \leq d$.*

Informellement : le système part de la configuration initiale (état de contrôle q_0 et toutes les horloges à zéro), puis effectue deux types de transitions : Les transitions d'action si la valeur courante des horloges le permet (ce type de transition s'effectue de manière *instantanée* et certaines horloges peuvent 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 l'état courant.

Une exécution possible de l'automate temporisé de la Figure 2.1.0.1 est : $(q_0, (0,0)) \xrightarrow{2.67} (q_0, (2.67, 2.67)) \xrightarrow{0.33} (q_0, (3,3)) \xrightarrow{b} (q_0, (3,0)) \xrightarrow{3} (q_0, (6,3)) \ldots$

Une exécution d'automate temporisé peut aussi être vue comme un mot temporisé, i.e. une séquence de paires (action, date). On peut la représenter comme une suit : $(q_0, v_0, t_0) \xrightarrow{a_1} (q_1, v_1, t_1) \xrightarrow{a_2} \ldots \xrightarrow{a_n} (q_n, v_n, t_n)$ avec $t_i \in \mathbb{R}_+$, $t_0 = 0$ et $t_{i+1} \geq t_i$ pour tout i. La date t_i correspond à la date où l'action a_i a été effectuée. L'étape $(q_i, v_i, t_i) \xrightarrow{a_{i+1}} (q_{i+1}, v_{i+1}, t_{i+1})$ correspond à une attente de durée $t_{i+1} - t_i$ puis le franchissement de 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 $(a_1, t_1)(a_2, t_2) \ldots$ Voir [7] pour une théorie des langages temporisés.

2.1.0.2. 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 nécessaire pour la modélisation de systèmes, elle n'ajoute pas d'expressivité sur le plan théorique : on peut toujours construire un automate produit ayant le même comportement que la composition parallèle considérée.

2.2. *Enoncer des propriétés temps-réel*

Les automates temporisés permettent de modéliser des systèmes temps-réel. Il est aussi nécessaire de définir des langages de spécification qui permettent d'énoncer des propriétés temps-réel, par exemple, on peut chercher à énoncer que "l'alarme se déclenche au plus *3 secondes* après la détection d'un problème". Pour cela on peut utiliser des extensions des logiques temporelles ou modales ou des automates de test.

2.2.0.3. Logiques temporelles "temporisés".

Il existe de nombreuses extensions de logiques temporelles qui permettent l'expression de contraintes quantitatives sur les délais séparant les actions du système à vérifier. La première consiste à compléter l'opérateur temporel Until (U) avec une contrainte de la forme $\bowtie k$ avec $\bowtie \in \{=, <, \leq, \geq, >\}$ et $k \in \mathbb{N}$. La formule $\phi \mathsf{U}_{\bowtie k} \psi$ est vraie pour un chemin ρ ssi il existe un état q, situé à moins de k unités de temps de l'état initial, et vérifiant ψ et tel que tous les états précédents vérifient ϕ. On étend de la même manière les opérateurs F ("un jour") ou G ("toujours"). Dans ce cadre la propriété décrite ci-dessus pourra s'écrire en $TCTL$ (*Timed* CTL, une logique de temps arborescent) par $\mathsf{AG}(\texttt{pb} \Rightarrow \mathsf{AF}_{\leq 3}\texttt{alarme})$ ou en $TLTL$ (une logique de temps linéaire) par $\mathsf{G}(\texttt{pb} \Rightarrow \mathsf{F}_{\leq 3}\texttt{alarme})$.
Une autre méthode consiste à ajouter des horloges aux logiques temporelles (on parle d'*horloges de formules*), des opérateurs de remises à zéro et des contraintes du même types que celles utilisées dans les automates. La formule précédente s'écrira alors de la manière suivante : $\mathsf{AG}(\texttt{pb} \Rightarrow x\mathbf{in}(\mathsf{AF}(x < 3 \wedge \texttt{alarme})))$. L'opérateur $\mathbf{in}$ remet l'horloge x à zéro lorsque l'on rencontre un état vérifiant pb, et il suffit donc de vérifier que $x < 3$ lorsqu'on rencontre un état vérifiant alarme pour s'assurer que le délai séparant les deux positions est bien inférieur à 3. Cette méthode permet d'exprimer des propriétés plus fines mais est moins intuitive que la précédente.

Il est aussi possible d'étendre les logiques modales avec des horloges de formules et d'utiliser des point fixes pour exprimer des propriétés sur le comportement des systèmes [3, 2].

Nous renvoyons à [8, 9, 3, 10, 27, 30, 2] pour une présentation détaillée de ces diverses solutions.

2.2.0.4. Automates de test.

Pour exprimer des propriétés, il est parfois plus simple de les décrire sous la forme d'un automate temporisé T. On place alors cet automate en parallèle avec le système à vérifier et on définit une synchronisation adéquate. Il s'agit ensuite de vérifier que certains états de T sont ou ne sont pas accessibles. Voir [1] pour une présentation de cette approche.

2.3. *Problèmes de vérification*

On distingue plusieurs problèmes de vérification selon les propriétés à vérifier.

– L'accessibilité d'un état de contrôle : il s'agit, étant donné un (ou une composition parallèle d') automate(s) temporisé(s) S et un état de contrôle q, de décider si il existe une exécution de S menant à une configuration (q, v). Ce type de problème est le problème le plus connu, tous les outils de model-checking traitent ces questions (par ex. UppAal).
Le problème dual de l'accessibilité est l'invariance : on cherche à alors à vérifier que tout état atteignable vérifie une certaine propriété (atomique).

– Le model-checking : il s'agit, étant donné une formule de logique temporelle (ou modale) ϕ et un système temporisé S, de vérifier si S satisfait ϕ (noté $S \models \phi$). Les outils Kronos ou HCMC permettent de vérifier ce genre de propriétés.

– Propriétés comportementales ou sur les langages : on peut aussi s'intéresser à des propriétés du type bisimulation ou simulation entre automates. Une autre approche consiste à voir les exécutions des automates comme des mots temporisés et les problèmes de vérification s'expriment alors sous la forme d'inclusion de langages temporisés etc. Voir [6].

2.4. *Analyse des automates temporisés*

Une fois que l'on a défini un automate temporisé A et spécifié une propriété ϕ, il reste à trouver un algorithme pour vérifier $A \models \phi$. Pour cela, il n'est pas possible d'utiliser directement les algorithmes standards de model-checking car le STT S_A correspondant à l'automate contient un nombre infini d'états. Une solution possible serait d'abstraire le STT S_A en un système de transition fini S'_A t.q. S_A et S'_A vérifient les mêmes propriétés.

Dans le reste de cette section, nous considérons le problème de l'accessibilité d'un état de contrôle (les autres problèmes de vérification peuvent être traités de manière similaire). Etant donnés A et un état de contrôle q_F, l'objectif est donc de décider si q_F est atteignable depuis (q_0, v_0). Nous allons passer de l'ensemble d'états $Q \times \mathbb{R}_+^X$

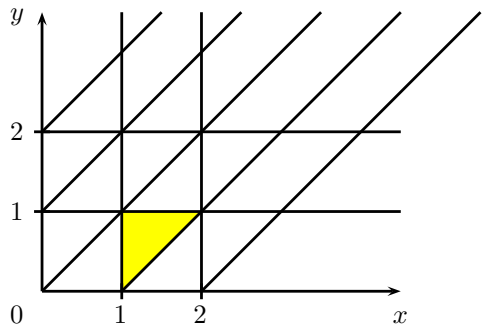


Figure 2. *Exemple de* $(\mathbb{R}_+^X)_{/\equiv_M}$ *avec* $M = 2$ *et* $X = \{x, y\}$

de S_A à $Q \times (\mathbb{R}_+^X)_{/\sim}$ pour S'_A tel que (1) $(\mathbb{R}_+^X)_{/\sim}$ soit fini et (2) pour tout u et v avec $u \sim v$, on ait : (q, u) et (q, v) ont les mêmes états accessibles.

Pour avoir les mêmes états accessibles, il suffit de pouvoir exécuter les mêmes transitions d'action (donc de satisfaire les mêmes gardes), d'arriver dans des configurations équivalentes (après les resets) et de pouvoir simuler les mêmes transitions de temps. Pour satisfaire les mêmes gardes de A, il est suffisant de considérer les gardes de $\mathcal{C}(X)$ ayant une constante entière inférieure à la constante maximale M utilisée dans les contraintes de A (gardes et invariants), on note $\mathcal{C}_M(X)$ cet ensemble (fini modulo équivalences booléennes). Formellement, nous cherchons donc une équivalence $\sim$ telle que $u \sim v$ entraîne les deux propriétés suivantes :

1) $\forall g \in \mathcal{C}_M(X),\ u \models g$ ssi $v \models g$ et $\forall r \subseteq X,\ u[r \leftarrow 0] \sim v[r \leftarrow 0]$.

2) $\forall t \geq 0,\ \exists t' \geq 0$, t.q. $u + t \sim v + t'$ (et symétriquement).

On dit alors que $\sim$ est M-stable. Une équivalence M-stable et d'index fini serait donc un bon candidat pour abstraire S_A car le problème d'accessibilité initial dans une structure infinie se ramenerait à un problème d'accessibilité dans un graphe fini.

En fait, il suffit de considérer l'équivalence $\equiv_{\mathcal{C}_M(X)}$ induite par l'ensemble de contraintes de $\mathcal{C}_M(X)$. On peut en effet montrer que $\equiv_{\mathcal{C}_M(X)}$ est M-stable (la preuve est assez longue et laissée en exercice :-) et elle est clairement d'index fini.

Si on considère le cas de $X = \{x, y\}$ avec $M = 2$, l'équivalence $\equiv_{\mathcal{C}_M(X)}$ engendre 70 classes d'équivalences représentées sur la Figure 2 sous la forme de points, segments ou surfaces. A l'avenir nous noterons $\equiv_M$ l'équivalence $\equiv_{\mathcal{C}_M(X)}$.

La zone grisée correspond aux valuations vérifiant $1 < x < 2 \wedge 0 < y < 1 \wedge y - 1 < x < y$.

Remarque 1 *Si l'automate A ne contient pas de garde diagonale, on utilise classiquement une autre équivalence définie comme suit : $u \sim v$ ssi :*

– $\forall x \in X, u(x) > M \ \equiv \ v(x) > M$
$u(x) \leq M \ \Rightarrow \ \lfloor u(x) \rfloor = \lfloor v(x) \rfloor \ \wedge \ \mathit{frac}(u(x)) = 0 \equiv \mathit{frac}(v(x)) = 0$,
– $\forall x, x' \in X, (u(x) \leq M \wedge u(x') \leq M) \Rightarrow (\mathit{frac}(u(x)) \leq \mathit{frac}(u(x')) \equiv \mathit{frac}(v(x)) \leq \mathit{frac}(v(x'))$

□

Etant donnée une valuation u, on note $[u]$ la classe d'équivalence de $\equiv_M$ contenant u. L'équivalence $\equiv_M$ est M-stable. Cela entraîne que toutes les valuations d'une même classe γ de $(\mathbb{R}_+^X)_{/\equiv_M}$ vérifient les même gardes de $\mathcal{C}_M(X)$. On peut ainsi noter $\gamma \models \phi$ lorsque les valuations de γ vérifient ϕ. De plus, étant données deux valuations u et v de γ, les valuations $u[r \leftarrow 0]$ et $v[r \leftarrow 0]$ appartiennent à la même classe pour tout $r \subseteq X$ (i.e. $[u[r \leftarrow 0]] = [v[r \leftarrow 0]]$) ; on peut ainsi définir la classe $\gamma[r \leftarrow 0]$. Enfin on peut aussi définir une opération `Succ` sur les classes : $\texttt{Succ}(\gamma) = \{[u + t] \mid u \in \gamma \text{ et } t \in \mathbb{R}_+\}$. $\texttt{Succ}(\gamma)$ contient toutes les classes γ' contenant des valuations qu'il est possible d'atteindre par une transition de temps à partir d'une valuation de γ. Toutes ces opérations sont aisément calculables à partir des définitions de $\equiv_M$.

Dans la plupart des articles de la littérature, une *région* de A désigne une classe d'équivalence γ ou une paire (q, γ). Nous pouvons maintenant définir le *Graphe des régions* [3] de l'automate A :

Définition 4 *Etant donné un automate temporisé $A = (Q, q_0, X, \mathit{Inv}, \mathcal{T}, \Sigma)$ et M la constante maximale apparaissant dans les gardes et invariants de A, le graphe des régions de A pour l'accessibilité est le système de transition fini $\mathcal{R}_A = (S', s'_0, \rightarrow, \Sigma)$ avec :*

– $S' = Q \times (\mathbb{R}_+^X)_{/\equiv_M}$,

– $s'_0 = (q_0, \gamma_0)$ *avec* $\gamma_0 = [v_0]$,

– $\rightarrow$ *se décompose en deux types de transitions :*

- Les actions : $(q, \gamma) \xrightarrow{a} (q', \gamma')$ *ssi* $\exists q \xrightarrow{g,a,r} q' \in \mathcal{T}$, $\gamma \models g$, $\gamma' = \gamma[r \leftarrow 0]$ *et* $\gamma' \models \mathit{Inv}(q')$.

- Les délais : $(q, \gamma) \xrightarrow{\delta} (q, \gamma')$ *ssi* $\gamma' \in \mathit{Succ}(\gamma)$ *et* $\gamma'' \models \mathit{Inv}(q)$ *pour tout* γ'' *t.q.* $\gamma'' \in \mathit{Succ}(\gamma)$ *et* $\gamma' \in \mathit{Succ}(\gamma'')$.

Il y a correspondance entre les exécutions de A et celles de $\mathcal{R}_A$:

Proposition 1 *Pour toute exécution* $(q_0, v_0, t_0) \xrightarrow{a_1} (q_1, v_1, t_1) \xrightarrow{a_2} \ldots \xrightarrow{a_n} (q_n, v_n, t_n)$ *de A, il existe une exécution* $(q_0, \gamma_0) \xrightarrow{\delta^*} \xrightarrow{a_1} (q_1, \gamma_1) \xrightarrow{\delta^*} \xrightarrow{a_2} \ldots \xrightarrow{\delta^*} \xrightarrow{a_n} (q_n, \gamma_n)$ *dans $\mathcal{R}_A$ (et réciproquement).*

En conclusion, on a le corollaire suivant :

Corollaire 1 *Un état de contrôle q_F est atteignable dans un automate temporisé A ssi un état de la forme (q_F, γ) est accessible dans $\mathcal{R}_A$.*

Cela donne un algorithme pour décider de l'accessibilité d'un état de contrôle.

2.4.0.5. Les autres problèmes de vérification.

La technique que nous venons de présenter est adaptable aux autres problèmes de vérification mentionnés précédemment. Par exemple, pour le model checking de TCTL avec les opérateurs $\mathsf{E\_U}_{\bowtie k}\_$ et $\mathsf{A\_U}_{\bowtie k}\_$, il suffit de construire un graphe des régions avec une horloge supplémentaire afin de mesurer les délais séparant deux configurations le long d'une exécution. Si on considère les versions de TCTL avec des horloges de formule, il suffit d'ajouter ces horloges au graphe de régions. Ensuite il reste à adapter l'algorithme d'étiquetage classique de CTL.

2.5. *Complexité*

L'algorithme d'accessibilité basé sur le graphe des régions n'est pas utilisé en pratique dans les outils de model checking en raison de la taille du graphe des régions. En effet, la taille de $(\mathbb{R}_+^X)_{/\equiv_M}$ est exponentielle dans le nombre d'horloges ($|X|$) et dans le codage de M ($|(\mathbb{R}_+^X)_{/\equiv_M}|$ est en $O(|X|! \cdot M^{|X|})$). Cette explosion combinatoire empêche la construction explicite de $\mathcal{R}_A$. En fait, l'accessibilité dans un automate temporisé est un problème difficile :

Théorème 1 *([3, 20])*

– L'accessibilité dans les automates temporisés est PSPACE-complet. [3]

– L'accessibilité dans les automates temporisés où les contraintes n'utilisent que les constantes 1 ou 2 est PSPACE-complet. [20]

– L'accessibilité dans les automates temporisés avec 3 horloges est PSPACE-complet. [20]

Notons qu'il est facile de montrer que l'accessibilité se résout en temps polynomial lorsque l'automate temporisé ne contient qu'une seule horloge. Le cas des automates avec deux horloges est ouvert.

Ces résultats de complexité sont à comparer à ceux existants pour les structures de Kripke classiques, le tableau 1 rappelle les principaux résultats [1].

1. AF-μ-cal est le μ-calcul sans alternation.

	Structures de Kripke	Automates temporisés
Accessibilité	NLOGSPACE-Complet	PSPACE-Complet
$TCTL$:	$O(\|S\| \cdot \|\phi\|)$	PSPACE-Complet
$TLTL$:	PSPACE-Complet	indécidable si $\mathcal{T} = \mathbb{R}$ EXPSPACE-Complet si $\mathcal{T} = \mathbb{N}$
AF-μ-cal.	$O(\|S\| \cdot \|\phi\|)$	EXPTIME-Complet

Tableau 1. *Complexité comparée : structure de Kripke et AT*

On peut remarquer que les complexités obtenues pour la vérification des automates temporisés (hormis celle pour $TLTL$ avec $\mathbb{R}$ comme domaine de temps) correspondent exactement à celles obtenues pour les compositions parallèles de structures de Kripke. En fait, du point de vue de la complexité, ajouter une horloge revient à ajouter un processus. De plus, il n'y a pas de saut de complexité lorsqu'on considère des compositions parallèles d'automates temporisés au lieu d'un unique AT. Cf [2].

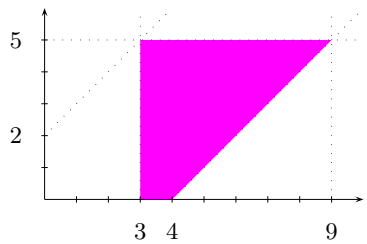
2.6. *Algorithmes et outils*

Le saut de complexité induit par les contraintes d'horloges a nécessité le développement de structures de données spécifiques pour manipuler les ensembles de valuations. L'idée générale est d'éviter de construire le graphe des régions et plutôt d'utiliser des structures de données symboliques, permettant de représenter des ensembles plus grands de valuations. Il existe plusieurs solutions, certaines s'inspirent des BDD : CDD (Clock Difference Diagram) [32], NDD (Numerical Decision Diagram) [11], RED ...

Ici nous nous limiterons à la description des DBM (Difference Bound Matrix), la structure de données la plus utilisée dans le domaine. Voir [22, 15] pour des présentations complètes. On va représenter les ensembles convexes de valuations obtenus par conjonctions de contraintes atomiques de la forme $x - y \bowtie k$ et $x \bowtie k$, on les appelle des *zones*. Tout d'abord, on suppose l'existence d'une horloge x_0 toujours nulle : Cela permet de représenter toutes les contraintes sous la forme d'une conjonction de contraintes sur des différences d'horloges ($x \bowtie l$ devient $x - x_0 \bowtie k$). On peut alors voir ce genre de contraintes comme un graphe valué où une arête de poids k entre les horloges x_i et x_j représente la contrainte $x_i - x_j \leq k$. Ce graphe peut être représenté sous la forme d'une matrice. L'exemple suivant montre une contrainte et sa matrice associée :

$$(x_1 \geq 3) \ \wedge \ (x_2 \leq 5) \ \wedge \ (x_1 - x_2 \leq 4) \qquad \begin{array}{c} \\ x_0 \\ x_1 \\ x_2 \end{array} \begin{array}{c} \begin{array}{ccc} x_0 & x_1 & x_2 \end{array} \\ \begin{pmatrix} +\infty & \mathbf{-3} & +\infty \\ +\infty & +\infty & \mathbf{4} \\ \mathbf{5} & +\infty & +\infty \end{pmatrix} \end{array}$$

En fait, à partir des contraintes $x_2 - x_0 \leq 5$ et $x_0 - x_1 \leq -3$, on peut déduire $x_2 - x_1 \leq 2$. On peut donc ajouter cette contrainte dans la matrice sans changer l'ensemble de valuations correspondant. Un point très important des DBM, c'est la possibilité de les *normaliser*. En effet, en appliquant l'algorithme de Floyd pour obtenir tous les plus courts chemins entre les sommets d'un graphe, on obtient l'unique matrice représentant les contraintes les plus fortes entre toutes les horloges. Dans l'exemple précédent, on obtient finalement :



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

En plus de la normalisation, des algorithmes pour décider si une zone est inclue dans une autre ou si deux zones représentent le même ensemble de valuations, ou pour calculer l'intersection de deux zones sont facilement implémentables (et efficaces).

Des problèmes subtiles existent : pendant plusieurs années l'algorithme classique d'accessibilité à la volée a été considéré correct, alors qu'il faisait une surapproximation de l'ensemble des configurations atteignables. Ainsi l'algorithme pouvait répondre que tel état était accessible alors qu'aucune exécution réelle n'existait ! Ce bug venait d'une opération d'extrapolation, il a été découvert récemment par Patricia Bouyer, voir [15] pour une présentation détaillée des DBM et de ces problèmes.

Les DBM sont des structures de données efficaces mais elles ne permettent pas la manipulation d'unions de zones, ce qui est nécessaire dans tous les algorithmes de model checking. C'est ce qui a motivé le développement des CDD [32] mais sans obtenir les mêmes avancées que les BDD avaient permises dans le domaine de la vérification non temporisée. Des optimisations dans la manipulation des DBM ont aussi été proposées (par exemple [31]).

On retrouve les algorithmes standards du model checking. Il y a d'une part les algorithmes à la volée, où l'on construit le graphe d'états (ici de zones) à la demande. L'outil UppAal [34], le plus connu des model checkers temporisés, utilise cet algorithme. UppAal permet aussi l'usage de variables entières et de diverses structures. Voir `http://www.uppaal.com/`.

L'outil Kronos [37] (`http://www-verimag.imag.fr/TEMPORISE/kronos/`) permet de vérifier des propriétés TCTL. Il utilise un algorithme d'étiquetage classique et permet aussi la vérification de propriétés d'accessibilité à la volée.

L'outil HCMC [29] (`http://www.lsv.ens-cachan.fr/~fl/`) permet de vérifier des propriétés écrites en logique modale temporisée. Il utilise un algorithme compositionnel : Etant donné un problème $(A_1|\dots|A_n) \models \phi$, on construit une formule $\phi_{/A_n}$ telle que $(A_1|\dots|A_n) \models \phi$ ssi $(A_1|\dots|A_{n-1}) \models \phi_{/A_n}$. Des règles de simplification permettent d'éviter (pas toujours !) que la taille de formule n'explose en cours de processus.

3. Automates hybrides linéaires

On peut prolonger la démarche des automates temporisés (i.e. automates + horloges) en ajoutant des variables dynamiques qui évoluent dans le temps selon certaines lois ; on obtient alors les *automates hybrides*. Lorsque l'on impose que les variables évoluent selon des pentes à valeur dans $\mathbb{Z}$, on obtient la classe des *automates hybrides linéaires*. Cette classe est très riche, elle contient évidemment les automates temporisés, puisque dans ce cas, toutes les variables ont une pente égale à 1.

3.1. *Définition et sémantique*

3.1.0.6. Notations.

Soit $X = \{x_1, \dots, x_n\}$ un ensemble de variables à valeur dans $\mathbb{R}$. On note $\mathcal{E}(X)$ les expressions linéaires sur X, c'est-à-dire les expressions de la forme $a_1 \cdot x_1 + \dots a_n \cdot x_n + a_0$ avec $a_i \in \mathbb{Z}$. Une *contrainte linéaire* ou *polyhédrale* sur X est de la forme $e \bowtie k$ avec $e \in \mathcal{E}(X)$, $\bowtie \in \{=, <, \leq, >, \geq\}$ et $k \in \mathbb{Z}$; l'ensemble des combinaisons booléennes de contraintes linéaires sur X est noté $\mathcal{C}_l(X)$.
On note $\mathcal{L}(X)$ l'ensemble des applications affines à coefficients dans $\mathbb{Z}$ sur $\mathbb{R}^n$; un élément α de $\mathcal{L}(X)$ est défini par une paire (A, B) où A est une matrice $n \times n$ à coefficients dans $\mathbb{Z}$ et $B \in \mathbb{Z}^n$. Etant donnée une valuation $v \in \mathbb{R}^n$ pour X, $\alpha(v)$ désigne la valuation $A \cdot v + B$. Ces applications sont utilisées pour les mises à jour des variables lors du franchissement des transitions ; en général on se contente de les décrire sous une forme simplifiée (par ex. "$x := 3y + 2$").

Formellement on a :

Définition 5 *Un* automate hybride linéaire *(AHL) est un 7-uplet* $(Q, q_0, X, \mathtt{Act}, \mathtt{Inv}, \mathcal{T}, \Sigma)$ *avec :*

– Q est un ensemble (fini) d'états de contrôle ou de localités.

– $q_o \in Q$ est l'état initial.

– $\mathtt{Act} : Q \to \mathbb{Z}^X$ est une fonction qui associe un vecteur d'activité à chaque état. Un tel vecteur associe à chaque variable une pente à valeur dans $\mathbb{Z}$.

– $\mathtt{Inv} : Q \to \mathcal{C}_l(X)$ associe à chaque état un invariant (i.e. une contrainte sur les variables de X).

– $\mathcal{T} \subseteq Q \times \mathcal{C}_l(X) \times \Sigma \times \mathcal{L}(X) \times Q$ est un ensemble de transition comprenant une garde, une étiquette et une mise à jour à effectuer lors de son franchissement.

– Σ *est un alphabet d'action.*

Etant donnés une valuation v pour X, un vecteur d'activité $\bar{d} \in \mathbb{Z}^n$ et $t \in \mathbb{R}_+$, on définit la valuation $v + \bar{d} \cdot t$ comme suit : $\forall x_i \in X$, on a $(v + \bar{d} \cdot t)(x_i) = v(x_i) + \bar{d}_i \cdot t$.

Comme les automates temporisés, une configuration d'un AHL est une paire (q, v) où q est un état de contrôle et v une valuation pour X. Là encore, on définit sa sémantique sous la forme d'un STT :

Définition 6 *(Sémantique des AHL) Soit A un AHL $(Q, q_0, X, \mathtt{Act}, \mathtt{Inv}, \mathcal{T}, \Sigma)$. La sémantique de A est définie par le STT $\mathcal{S}_A = (S, s_0, \rightarrow, \Sigma)$ avec :*

– $S = Q \times \mathbb{R}^X$,

– $s_0 = (q_0, v_0)$ *avec* $v_0(x) = 0,\ \forall x \in X$,

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

- Les transitions d'actions : $(q, v) \xrightarrow{a} (q'v')$ *ssi* $\exists q \xrightarrow{g,a,\alpha} q' \in \mathcal{T}$ *et* $v \models g$, $v' = \alpha(v)$ *et* $v' \models \mathtt{Inv}(q')$.

- Les transitions de temps : Soit $t \in \mathbb{R}_+$. $(q, v) \xrightarrow{t} (q, v + \mathtt{Act}(q) \cdot t)$ *ssi* $v + \mathtt{Act}(q) \cdot t' \models \mathtt{Inv}(q)$ *pour tout* $0 \leq t' \leq t$.

Considérons un brûleur à gaz [24] qui fuit de temps à autre. Supposons que toute fuite est détectée et résorbée en moins de 1s et qu'un délai minimum de 30s sépare deux fuites. On souhaite maintenant vérifier que le temps total de fuite est au plus $1/20$ du temps total durant tout interval supérieur à 60s. On peut modéliser un tel système par l'automate hybride linéaire de la Figure 3. L'état q_0 correspond à l'état où le système fuit. L'horloge y mesure le temps total de fonctionnement, l'horloge x mesure le temps de présence dans chacun des états (elle est remise à zéro à chaque changement d'état) et z mesure le temps total de fuite (elle n'est active que dans l'état q_0). La propriété à vérifier est donc $(y \geq 60) \Rightarrow (20z \leq y)$.

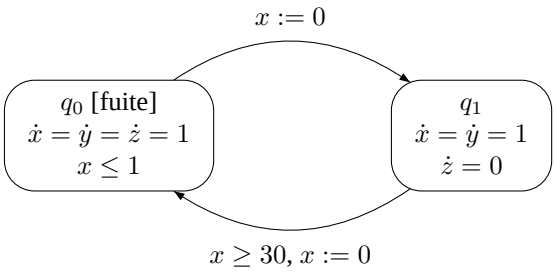


Figure 3. *Exemple de AHL : un brûleur à gaz*

De la même manière que pour les AT, on peut définir une composition parallèle qui permet de décrire facilement des systèmes complexes. Et là encore, on peut utiliser des logiques temporisés pour énoncer des propriétés quantitatives, on peut même ajouter des variables dynamiques dans les logiques [18].

3.2. *Indécidabilité*

Clairement les automates hybrides linéaires étendent les automates temporisés par :

– des gardes linéaires générales au lieu de contraintes de type $x \bowtie k$ ou $x - y \bowtie k$,

– des mises à jour plus riches au lieu de simples remises à zéro,

– des pentes différentes pour les variables selon les états et non de simples horloges.

En fait, chacune de ces extensions prise séparément rend déjà l'ensemble des problèmes de vérification indécidable ! En effet, l'ajout de gardes de la forme $x + y \bowtie k$ dans les automates temporisés suffit à rendre l'accessibilité indécidable [13]. L'ajout de mises à jour étendue comme $x := x + 1$ rend aussi l'accessibilité indécidable

(voir [17, 16] pour un tableau précis de ces résultats de décidabilité et d'indécidabilité). Enfin, considérer des variables avec des pentes différentes rend très vite les problèmes indécidables, nous avons notamment le résultat suivant [26] : l'accessibilité est indécidable dans les automates où une variable a deux pentes différentes selon les états. Voir [26] pour un excellent survey sur ces questions, où sont aussi décrites des sous-classes décidables de AHL.

Notons que la recherche de sous-classes des AHL est une tâche importante. En effet, cela permet d'isoler des familles décidables (par exemple les automates rectangulaires initialisés [26]) ou des familles pour lesquelles des optimisations sont possibles. De plus, en pratique, certaines sous-classes sont particulièrement intéressante, c'est le cas de la famille des p-automates [14] pour la description de protocoles telecom. Les automates à chronomètres (i.e. munis de variables à pente dans $\{0, 1\}$) permet de modéliser de nombreux problèmes de scheduling avec préemption.

3.3. *Vérification des automates hybrides linéaires*

Les résultats de la section précédente font qu'il n'existe pas pour les AHL généraux des algorithmes pour vérifier les propriétés d'accessibilité ou plus généralement pour le model checking. Il est néanmoins possible de définir des semi-algorithmes (i.e. qui peuvent ne pas terminer) ou des approximations qui garantissent la terminaison mais qui ne fournissent pas toujours un résultat [2].

Le point clé de ces approches repose sur l'usage des contraintes linéaires pour représenter les ensembles d'états du système à vérifier. En effet, on peut définir des opérations $\texttt{Pre}_e(z)$ ou $\texttt{Post}_e(z)$ pour calculer les états prédécesseurs ou successeurs d'un ensemble z par une transition e ; si z est une contrainte linéaire, alors $\texttt{Pre}_e(z)$ et $\texttt{Post}_e(z)$ peuvent être décrits sous la forme de contraintes linéaires [5]. Cela permet d'utiliser les bibliothèques de calcul sur polyhèdres pour implémenter des algorithmes d'accessibilité dans les AHL. On désigne alors par $\texttt{Post}^*(z)$ l'ensemble des états accessibles depuis z et par $\texttt{Pre}^*(z)$ l'ensemble des prédécesseurs de z, ces ensembles se définissent sous la forme de points fixes à partir des opérateurs Pre et Post.

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 [25] pour une description complète et des exemples (`http://www-cad.eecs.berkeley.edu/~tah/HyTech/`).

Dans le cas du brûleur à gaz, pour vérifier le système, on peut soit calculer l'ensemble [3] $\texttt{Post}^*(x = y = z = 0 \wedge \text{état-}q_0)$ et vérifier que $\texttt{Post}^*(x = y = z = 0) \cap y \geq 60 \cap 20z > y)$ est vide, soit calculer l'ensemble $\texttt{Pre}^*(y \geq 60 \wedge 20z > y)$ et vérifier que $\texttt{Pre}^*(y \geq 60 \wedge 20z > y) \cap (x = y = z = 0 \wedge \text{état-}q_0)$ est vide.

L'outil HyTech permet de faire ce second calcul automatiquement et donne :

$$
\begin{array}{l}
\texttt{Pre}^*(y \geq 60 \wedge 20z > y) = \\
\text{état-}q_0 \wedge \Big(19x + y \leq 20t + 19 \wedge y \geq x + 59 \wedge x \leq 1 \wedge x \geq 0 \\
\vee \quad t \geq x + 2 \wedge x \leq 1 \wedge y \geq 0 \wedge 19x + y \leq 20t + 19 \wedge x \geq 0 \\
\vee \quad t \geq x + 1 \wedge x \leq 1 \wedge y \geq 0 \wedge 19x + y \leq 20t + 8 \wedge x \geq 0 \\
\vee \quad 20t \geq 19x + y + 3 \wedge y \geq 0 \wedge x \leq 1 \wedge x \geq 0 \Big) \\
\text{état-}q_1 \wedge \Big(x \geq 0 \wedge t \geq 3 \wedge y \leq 20t \wedge y \geq 0 \\
\vee \quad x + 20t \geq y + 11 \wedge y \leq 20t + 19 \wedge t \geq 2 \wedge x \geq 0 \wedge y \geq 0 \\
\vee \quad y \geq 0 \wedge t \geq 1 \wedge x + 20t \geq y + 22 \wedge y \leq 20t + 8 \wedge x \geq 0 \\
\vee \quad y \geq 0 \wedge x + 20t \geq y + 33 \wedge 20t \geq y + 3 \wedge x \geq 0 \Big)
\end{array}
$$

Et on a bien : $\texttt{Pre}^*(y \geq 60 \wedge 20z > y) \cap (x = y = z = 0 \wedge \text{état-}q_0) = \emptyset$, la propriété est donc vérifée. Si on lance, avec HyTech, le calcul de $\texttt{Post}^*(x = y = z = 0 \wedge \text{état-}q_0$ alors le calcul ne termine pas ! Mais il est possible de calculer "à la main" les premiers pas du calcul puis de deviner le point fixe résultat (et de vérifier que celui-ci est bien le point fixe) puis bien-sûr on peut montrer que la propriété est aussi vraie avec cette analyse en avant.

---

2. Dans ce cas, le programme peut répondre $S \models \phi$, $S \not\models \phi$ ou "ne sait pas"
3. On utilise des propositions pour différencier les états de contrôle, cela revient à ajouter une variable discrète à notre système.

L'autre possibilité consiste à utiliser des approximations, on peut alors garantir la terminaison des algorithmes. Il s'agit aussi d'éviter des calculs coûteux (par exemple tester l'inclusion d'un polyhèdre dans un autre). D'abord on peut utiliser des enveloppes convexes au lieu d'union de polyhèdres ; ensuite, on peut utiliser un opérateur d'élargissement qui permet d'empêcher la divergence du calcul des points fixes. Voir [5] pour une description de ces aspects qui s'inspirent des travaux sur l'interprétation abstraite [21].

4. Conclusion

Depuis leur définition et leur utilisation pour la vérification, dans les années 90, les automates temporisés et hybrides se sont révélés être de bons modèles pour représenter le comportement des systèmes temps-réel. Beaucoup de résultats théoriques et pratiques ont été obtenus. Des outils existent, ils ont permis de vérifier des études de cas issues du milieu industriel. L'un des principaux problèmes réside dans l'explosion combinatoire due aux contraintes quantitatives. Si les BDD ont permis, dans le cas non temporisé, de traiter des systèmes de taille très importante, la situation est plus difficile dans le cas temporisé dans la mesure où il faut à la fois gérer la complexité due à la mise en parallèle de plusieurs processus (comme dans le cadre classique) et aussi celle due aux contraintes temporelles. Des progrès sont nécessaires dans ce domaine.

Le développement de modèles spécifiques pour certaines applications est aussi une piste de recherche prometteuse. Cela permet de développer des heuristiques particulières et d'éviter ainsi des coûts excessifs.

D'autres modèles intégrant du temps existent, dans les réseaux de Petri notamment, dans les algèbres de processus ... Ces problématiques se rejoignent et favoriser les interactions entre ces différents domaines est un enjeu important.

Remerciements 1 *Je remercie Béatrice Bérard, Patricia Bouyer et Franck Cassez pour leur relecture de ce document et les nombreuses discussions sur ce sujet ...*

Références

[1] L. Aceto, P. Bouyer, A. Burgueño, and K. G. Larsen. The power of reachability testing for timed automata. *Theoretical Computer Science*, 2003. To appear.

[2] L. Aceto and F. Laroussinie. Is your model checker on time ? On the complexity of model checking for timed modal logics. *Journal of Logic and Algebraic Programming*, 52–53 :7–51, 2002.

[3] R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1) :2–34, 1993.

[4] R. Alur, C. Courcoubetis, N. Halbwachs, D. L. 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.

[5] 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.

[6] 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.

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

[8] R. Alur and T. A. Henzinger. A really temporal logic. In *Proc. 30th IEEE Symp. Foundations of Computer Science (FOCS'89), Research Triangle Park, NC, USA, Oct. 1989*, pages 164–169, 1989.

[9] 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.

[10] R. Alur and T. A. Henzinger. Real-time logics : Complexity and expressiveness. *Information and Computation*, 104(1) :35–77, 1993.

[11] E. Asarin, M. Bozga, A. Kerbrat, O. Maler, A. Pnueli, and A. Rasse. Data-Structures for the Verification of Timed Automata. In *Proc. Int. Workshop Hybrid and Real-Time Systems (HART'97), Grenoble, France, Mar. 1997*, volume 1201 of *Lecture Notes in Computer Science*. Springer, 1997.

[12] J. Bengtsson, W. O. D. Griffioen, K. J. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson, and Wang Yi. Verification of an audio protocol with bus collision using UPPAAL. 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 244–256. Springer, 1996.

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

[14] B. Bérard and L. Fribourg. Automated verification of a parametric real-time program : the ABR conformance protocol. In *Proc. 11th Int. Conf. Computer Aided Verification (CAV'99), Trento, Italy, July 1999*, volume 1633 of *Lecture Notes in Computer Science*, pages 96–107. Springer, 1999.

[15] P. Bouyer. Untameable timed automata ! In *Proc. 20th Ann. Symp. Theoretical Aspects of Computer Science (STACS'2003), Berlin, Germany, Feb. 2003*, volume 2607 of *Lecture Notes in Computer Science*, pages 620–631. Springer, 2003.

[16] 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.

[17] 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.

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

[19] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

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

[21] P. Cousot and R. Cousot. Abstract interpretation : A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. 4th ACM Symp. Principles of Programming Languages, Los Angeles, CA, USA*, pages 238–252, January 1977.

[22] David 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*, pages 197–212, June 1989.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

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

[29] 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.

[30] F. Laroussinie, K. G. Larsen, and C. Weise. From timed automata to logic - and back. In *Proc. 20th Int. Symp. Math. Found. Comp. Sci. (MFCS'95), Prague, Czech Republic, Aug.-Sep. 1995*, volume 969 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 1995.

[31] K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Efficient verification of real-time systems : Compact data structure and state-space reduction. In *Proc. of the 18th IEEE Real-Time Systems Symposium, RTSS'97*. IEEE Computer Society Press, December 1997.

[32] K. G. Larsen, J. Pearson, C. Weise, and Wang Yi. Clock difference diagrams. *Nordic Journal of Computing*, 6(3) :271–298, 1999.

[33] K. G. Larsen, P. Pettersson, and Wang Yi. Model-checking for real-time systems. In *Proc. 10th Int. Conf. Fundamentals of Computation Theory (FCT'95), Dresden, Germany, Aug. 1995*, volume 965 of *Lecture Notes in Computer Science*, pages 62–88. Springer, 1995.

[34] K. G. Larsen, P. Pettersson, and Wang Yi. UPPAAL in a nutshell. *Journal of Software Tools for Technology Transfer*, 1(1–2) :134–152, 1997.

[35] Ph. Schnoebelen, B. Bérard, M. Bidoit, F. Laroussinie, and A. Petit. *Vérification de logiciels : Techniques et outils du model-checking*. Vuibert, 1999.

[36] S. Tripakis and S. Yovine. Verification of the fast-reservation protocol with delayed transmission using Kronos. Tech. Report 95-23, Verimag, Grenoble, France, 1995.

[37] S. Yovine. Kronos : A verification tool for real-time systems. *Journal of Software Tools for Technology Transfer*, 1(1–2) :123–133, 1997.