Rapport de stage
DEA Algorithmique 2003/2004

Aux frontières de la décidabilité...

Agnès ROBIN

Sous la direction de Béatrice Bérard et de Patricia Bouyer

Laboratoire Spécification et Vérification, ENS Cachan

Table des matières

Introduction

La modélisation et la vérification de systèmes sont aujourd'hui de plus en plus utilisées par les industriels pour éviter ou repérer des erreurs et des fonctionnements inattendus. Depuis une dizaine d'années, cette activité s'est étendue de manière significative aux systèmes dits temporisés, dans lesquels le temps est manipulé de façon explicite.

Cette avancée est principalement due aux travaux d'Alur et Dill [AD90], qui ont montré un important résultat de décidabilité pour une classe particulière de modèles temporisés : des automates finis auxquels sont ajoutées des variables à valeurs réelles appelées horloges. Ces horloges évoluent de manière synchrone avec le temps, peuvent être comparées à des constantes et remises à zéro.

Le résultat d'Alur et Dill est un algorithme qui répond à la question : "le langage reconnu est-il vide ou non ?" Ce problème peut être associé à celui de l'accessibilité ; en effet, un état final donné d'un système est accessible si et seulement si le langage reconnu par celui-ci n'est pas vide. Or ce problème est l'un des problèmes intéressant du model-checking : cet algorithme permet de savoir rapidement si un système peut atteindre un état dangereux ou instable.

Depuis, plusieurs enrichissements de ce modèle ont été proposés afin de modéliser des systèmes plus complexes. Pour ces modèles à base d'automates temporisés, les questions de décidabilité ont été largement explorées. Cependant, la frontière séparant les sous-classes décidables des autres reste encore floue dans plusieurs cas. En particulier, une sous-classe d'une classe indécidable n'ayant qu'un petit nombre d'horloges peut se révéler être décidable. C'est le cas par exemple lorsque des contraintes additives sont autorisées : la classe correspondante est décidable pour deux horloges et indécidable pour quatre.

Pour savoir si un problème est décidable, la méthode communément utilisée est de le ramener à un problème similaire sur un modèle fini. Ainsi, l'existence d'une bisimulation finie constructible permet de prouver la décidabilité : la construction d'un automate fini en bisimulation (time-abstract) avec l'automate temporisé original permet de tester de manière équivalente l'accessibilité sur l'automate temporisé et sur l'automate fini. Inversement, un système est indécidable s'il permet de simuler une machine de Minsky. Mais il existe des problèmes pour lesquels aucune bisimulation finie n'est possible, et qui sont a priori trop "faibles" pour pouvoir simuler une machine de Minsky. C'est à de tels problèmes que nous allons nous intéresser.

Nous commencerons par présenter le modèle des automates temporisés que nous allons étudier. Il s'agit d'une version étendue par rapport au modèle original étudié par Alur et Dill[AD94]. Puis nous rappellerons rapidement les méthodes utilisées pour prouver la décidabilité de certaines sous-classes et l'indécidabilité d'autres.

Dans une deuxième partie, nous verrons en quoi la méthode permettant de conclure à la décidabilité échoue pour la classe des automates à contraintes positives ayant trois horloges.

Alors, dans une troisième partie, nous chercherons à étendre cette méthode, en utilisant une bisimulation qui, bien qu'infinie, permet de conclure dans certains cas.

Chapitre 1

L'accessibilité pour les automates temporisés

1.1 Automates temporisés

En 1990, Alur et Dill ont présenté un modèle, les automates temporisés [AD90, AD94], obtenus à partir des automates finis en introduisant des contraintes sur le temps par l'intermédiaire d'horloges et de gardes sur ces horloges.
La définition que nous donnons ici diffère du modèle original par l'ajout de contraintes et de mises à jour plus variées. Le système obtenu est plus à même de représenter la réalité, mais n'est plus toujours décidable. Ce modèle étendu des automates temporisés est inspiré des travaux [BDFP04].
Le modèle pourrait être rendu encore plus complexe en utilisant des horloges de pentes différentes, voire en faisant varier leurs pentes ou en leur permettent de s'arrêter. On obtiendrai alors le cadre des automates hybrides [Hen96] que nous n'aborderons pas ici.

Les horloges

Le *domaine du temps*, noté $\mathbb{T}$, sera l'ensemble $\mathbb{Q}^+$ des rationnels positifs ou nuls, ou l'ensemble $\mathbb{R}^+$ des réels positifs ou nuls. Une *suite de dates* est une suite croissante $(t_i)_{i\in\mathbb{N}},\ t_i \in \mathbb{T}$. Nous considérons un ensemble fini d'*horloges* X. Intuitivement, il s'agit de variables avançant à la même vitesse et pouvant être mises à certaines valeurs indépendamment les unes des autres. Une *valuation* est une application $v \ :\ X \to \mathbb{T}$ qui associe à chaque horloge sa valeur. L'ensemble des valuations est noté $\mathbb{T}^X$. Soit $t \in \mathbb{T}$, la valuation $v + t$ est définie par : $(v + t)(x) = v(x) + t\ \forall x \in X$. On note 0 la valuation associant à chaque horloge la valeur nulle.

Contraintes d'horloges

Étant donné un ensemble d'horloges X, on définit l'ensemble $\mathcal{C}(X)$ des *contraintes linéaires*. Un *terme linéaire* est une expression de la forme $k_1x_1 + ... + k_mx_m$, où les x_i sont des horloges de X et les k_i des constantes de $\mathbb{Q}$. Ces contraintes sont des conjonctions de termes linéaires comparés à une constante :

$$\varphi := l \sim k_0 \text{ où } l \text{ est un terme linéaire sur } X,\ k_0 \in \mathbb{Q} \text{ et } \sim \in \{<, \leq, =, \geq, >\}.$$

Une valuation v satisfait une contrainte φ (ce qui se note "$v \models \varphi$") si les conditions suivantes sont remplies :

$$\begin{cases} v \models l \sim k_0 & \text{si } k_1 v(x_1) + ... + k_m v(x_m) \sim k_0 \\ v \models \varphi_1 \wedge \varphi_2 & \text{si } v \models \varphi_1 \text{ et } v \models \varphi_2 \end{cases}$$

On note $[\![\varphi]\!] = \{v \in \mathbb{T}^X \mid v \models \varphi\}$ l'ensemble des valuations vérifiant une contrainte φ.

exemples Soit X un ensemble d'horloges (par la suite, lorsque X ne sera pas donné par le contexte, il sera juste supposé fini). Voici des exemples d'ensemble de contraintes de $\mathcal{C}(X)$:

- Contraintes rectangulaires
  $\mathcal{C}_{rect} = \{x \sim c \mid x \in X,\ \sim \in \{<, \leq, =, \geq, >\},\ c \in \mathbb{Q}\}$
- Contraintes diagonales
  $\mathcal{C}^- = \{x - y \sim c \mid x \in X,\ \sim \in \{<, \leq, =, \geq, >\},\ c \in \mathbb{Q}\} \cup \mathcal{C}_{rect}$
- Contraintes additives
  $\mathcal{C}^+ = \{x + y \sim c \mid x \in X,\ \sim \in \{<, \leq, =, \geq, >\},\ c \in \mathbb{Q}\} \cup \mathcal{C}_{rect}$
- On notera $\mathcal{C}^\pm = \mathcal{C}^+ \cup \mathcal{C}^-$.

Mises à jour

Les contraintes d'horloges permettent de tester les valeurs de ces horloges. Pour pouvoir changer ces valeurs, on utilise des *mises à jour*, fonctions de $\mathbb{T}^X$ vers $\mathcal{P}(\mathbb{T}^X)$[1]. Une mise à jour associe à une valuation un ensemble de valuations. Ces mises à jour sont définies par des conjonctions d'opérations de la forme

$$\bigwedge_{x \in X} up(x),\ \text{où } up \in \{x :\sim ky + c \mid c \in \mathbb{Q},\ x, y \in X,\ k \in \mathbb{Q} \text{ et } \sim \in \{<, \leq, =, \geq, >\}\}.$$

Soit v une valuation et up une mise à jour de l'horloge x. Une valuation v' est dans $up(v)$ si $v'(z) = v(z)$ pour toutes les horloges $z \neq x$ et si $v'(x)$ satisfait :

$$v'(x) \sim kv(y) + c \wedge v'(x) \geq 0 \text{ si } up = x :\sim ky + c$$

Étant donné un ensemble d'horloges X, on note $Up(X)$ l'ensemble des mises à jour sur X.

exemple $\mathcal{U}_0$ est l'ensemble $\{x := 0 \mid x \in X\}$ des remises à zéro .

Automates temporisés

Un *automate temporisé*[2] sur $\mathbb{T}$ est un n-uplet $\mathcal{A} = (\Sigma, X, Q, \Delta, I, F)$, où :
- Σ est un alphabet fini

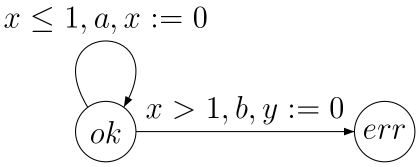
[1] $\mathcal{P}(\mathbb{T}^X)$ représente les parties de $\mathbb{T}^X$.

[2] La définition présentée ici est plus générale que celle généralement utilisée, afin d'inclure les extensions étudiées au cours de ce stage.

– X est un ensemble fini d'horloges
– Q est un ensemble fini d'états
– $\Delta \subseteq Q \times [\mathcal{C}(X) \times \Sigma \times Up(X)] \times Q$ est un ensemble fini de transitions
– $I \subseteq Q$ est le sous-ensemble des états initiaux
– $F \subseteq Q$ est le sous-ensemble des états finaux

Soit $q \xrightarrow{\varphi,a,up} q' \in \Delta$ une transition : φ est la garde sur les horloges, up la mise à jour.

exemple



Un *chemin* de $\mathcal{A}$ est une suite finie de transitions :

$$P = q_0 \xrightarrow{\varphi_1,a_1,up_1} q_1 \xrightarrow{\varphi_2,a_2,up_2} q_2 ..., \text{ où } q_{i-1} \xrightarrow{\varphi_i,a_i,up_i} q_i \in \Delta \ \forall i > 0$$

Ce chemin est *acceptant* s'il part d'un état initial ($q_0 \in I$) et s'il est fini et finit dans un état final ($q_f \in F$). Une *exécution* du chemin P est une suite de la forme :

$$\langle q_0, 0 \rangle \xrightarrow[t_1]{a_1} \langle q_1, v_1 \rangle \xrightarrow[t_2]{a_2} \langle q_2, v_2 \rangle ...$$

où $\tau = (t_i)_{i \geq 1}$ est une suite de dates et $(v_i)_{i \geq 0}$ des valuations d'horloges telles que :

$$\begin{cases} v_{i-1} + (t_i - t_{i-1}) \models \varphi_i \\ v_i \in up_i(v_{i-1} + (t_i - t_{i-1})) \end{cases}$$

Une telle exécution est étiquetée par le *mot temporisé* $w = (a_1, t_1)(a_2, t_2)...$ Si le chemin P est acceptant, alors ce mot temporisé est dit accepté par $\mathcal{A}$. L'ensemble de tous les mots temporisés acceptés par $\mathcal{A}$ est noté $L(\mathcal{A})$.

exemple Pour l'automate présenté ci-dessus, un chemin possible est :

$$\begin{bmatrix} ok \\ 0 \\ 0 \end{bmatrix} \xrightarrow{a,0.5} \begin{bmatrix} ok \\ 0 \\ 0.5 \end{bmatrix} \xrightarrow{a,0.3} \begin{bmatrix} ok \\ 0 \\ 0.8 \end{bmatrix} \xrightarrow{a,1} \begin{bmatrix} ok \\ 0 \\ 1.8 \end{bmatrix} \xrightarrow{b,2.2} \begin{bmatrix} err \\ 2.2 \\ 0 \end{bmatrix}.$$

Ce chemin est étiqueté par le mot temporisé $(a, 0.5), (a, 0.8), (a, 1.8), (b, 4)$.

Soit $\mathcal{C} \subseteq \mathcal{C}(X)$ un ensemble de contraintes et $Up \subseteq Up(X)$ un ensemble de mises à jour. On notera $\mathbf{AT}(\mathcal{C}, Up)$ l'ensemble de tous les automates temporisés dont les transitions n'utilisent que des contraintes de $\mathcal{C}$ et des mises à jours de Up.

Nous allons étudier le problème du vide, fondamental en vérification, qui est présent dans les problèmes de sûreté (lorsqu'un système ne doit pas se trouver dans certains états), ainsi que pour les questions d'inclusion de langages (savoir si $L_1 \subseteq L_2$ revient à se demander si $L \cap \overline{A} = \emptyset$.)

Ce problème est le suivant : « existe-t-il un algorithme qui, étant donné un automate temporisé, répond OUI ou NON suivant que le langage reconnu par l'automate est vide ou pas ? »
Cette question revient à se demander si l'un des états finaux est accessible. Un algorithme permettant de résoudre ce problème peut alors donc servir pour certaines questions de model-checking sur les systèmes modélisés par des automates temporisés.
Il est donc intéressant de se demander pour quelles classes d'automates temporisés le problème du vide est décidable (on parlera à présent de *classes décidables*), afin de savoir jusqu'à quel point on peut enrichir le modèle utilisé.
Nous allons commencer par rappeler la technique utilisée par Alur et Dill pour montrer la décidabilité de la classe la plus simple d'automates temporisés, puis nous verrons comment l'étendre. Enfin nous rappellerons une technique souvent utilisée pour montrer l'indécidabilité d'une classe d'automates temporisés : la réduction à une machine de Minsky.

1.2 Classes décidables

Le modèle d'Alur et Dill

Les automates temporisés étudiés par Alur et Dill correspondent à la classe $\mathbf{AT}(\mathcal{C}_{rect}, \mathcal{U}_0)$ ($\mathcal{C}_{rect}$ est l'ensemble des contraintes rectangulaires défini page 6 et $\mathcal{U}_0$ l'ensemble des remises à zéros.) La propriété principale de ce modèle est la suivante :

Théorème 1.2.1 *[AD90, AD94] La classe $\boldsymbol{AT}(\mathcal{C}_{rect}, \mathcal{U}_0)$ est décidable.*

La preuve repose sur la construction pour tout automate temporisé $\mathcal{A}$ d'un automate fini $\mathcal{B}$ reconnaissant exactement le langage $\textsc{Untime}(L(\mathcal{A}))$ défini par :

$$\textsc{Untime}(L(\mathcal{A})) = \{\sigma \in \Sigma^* \mid \exists \tau \in \mathbb{T}^* \text{ tel que } (\sigma, \tau) \in L(\mathcal{A})\}$$

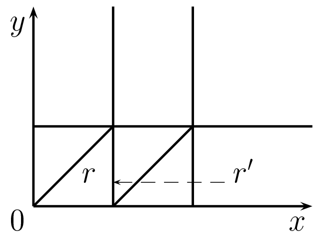
Il est évident que le vide de $L(\mathcal{A})$ est équivalent au vide de $\textsc{Untime}(L(\mathcal{A}))$, donc le résultat découle de la décidabilité du vide pour les automates finis.
Nous allons rappeler la construction de $\mathcal{B}$ en l'adaptant à notre modèle plus général d'automates temporisés [BDFP04].

Régions

Soit X un ensemble fini d'horloges. On considère une partition R de $\mathbb{T}^X$.

exemple

$r = \{0 < x < 1,\ 0 < y < 1,\ x > y\}$, $r' = \{x = 1,\ 0 < y < 1\}$ sont des régions de cette partition.



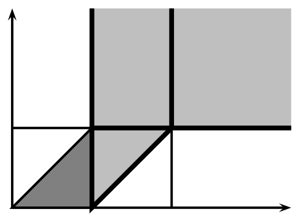
Pour toute valuation $v \in \mathbb{T}^X$, l'unique élément de R qui contient v est noté $[v]_\mathrm{R}$. On définit naturellement les successeurs stricts de $r \in \mathrm{R}$ par :

$$r' \in Succ(r) \text{ si } r \neq r' \text{ et } \exists v \in \mathrm{R},\ \exists t \in \mathbb{T} \text{ tels que } [v+t]_\mathrm{R} = r'$$

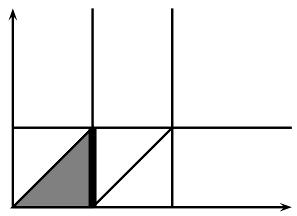
Un successeur immédiat est une région telle que :

$$r' \in Succ_{\texttt{immédiat}}(r) \text{ si } r' \in Succ(r) \textit{ et } \forall r'' \in Succ(r),\ r' \notin Succ(r'')$$

Remarque Cette notion de successeur immédiat est introduite ici afin de suivre précisément l'écoulement du temps et les changements de régions associés. Ainsi, le passage à un successeur immédiat correspond au passage à une nouvelle région sans aucun intermédiaire : on exclut le cas de région étant leur propre successeur.



Partie gris clair : successeurs de r



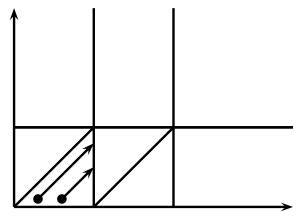
r' est successeur immédiat de r

On dira d'une telle partition qu'elle est *compatible* lorsqu'elle vérifie les conditions suivantes.
$\forall r, r' \in \mathrm{R}$,

– compatibilité avec le temps :

$$r' \in Succ(r) \iff \forall v \in r,\ \exists t \in \mathbb{T} \text{ tel que } [v+t]_\mathrm{R} = r' \qquad (\star)$$

Cette condition impose à la relation d'équivalence définie par la partition R d'être stable par rapport à l'écoulement du temps. Intuitivement, cela signifie que deux valuations équivalentes restent équivalentes lorsque le temps s'écoule : si v est dans la même région que v', et si on peut laisser t unités de temps s'écouler à partir de v, alors on peut laisser s'écouler t' unités de temps à partir de v' de façon à ce que $v+t$ et $v'+t'$ soient dans la même région.



Deux valuations de r arrivant dans r'

– compatibilité avec $Up \subseteq Up(X)$, ensemble fini de mises à jour :

$$r' \in \underline{up}(r) \implies \forall v \in r,\ \exists v' \in r' \text{ tel que } v' \in up(v) \qquad (\dagger)$$

$\underline{up} : \mathrm{R} \to \mathcal{P}(\mathrm{R})$ étant une fonction naturellement induite sur le quotient, qui associe à une région de R l'ensemble $\{r' \in \mathrm{R} \mid up(r) \cap r' \neq \emptyset\}$. Cette condition a une interprétation similaire à la condition $(\star)$: l'équivalence doit également être stable par rapport aux mises à jour.

– compatibilité avec $\mathcal{C} \subseteq \mathcal{C}(X)$, ensemble fini de contraintes :

$$\forall \varphi \in \mathcal{C},\ \forall r \in \mathrm{R},\ r \subseteq [\![\varphi]\!] \text{ ou } r \subseteq [\![\neg\varphi]\!] \tag{‡}$$

Pour toute contrainte d'horloge $\varphi \in \mathcal{C}$, chaque région de R est contenue dans l'ensemble des valuations vérifiant φ ou en est exclue.

Automate des régions

Soient $Up \subseteq Up(X)$, $\mathcal{C} \subseteq \mathcal{C}(X)$ finis et R une partition de $\mathbb{T}^X$ compatible avec Up et $\mathcal{C}$. On peut définir le *graphe des régions* $\mathcal{R}$ associé à R, Up et $\mathcal{C}$, dont les états sont les régions de R et les arêtes sont de deux types distincts :

$r \xrightarrow{\varepsilon} r'$ $\quad \forall r, r'$ tels que $r' \in Succ_{\texttt{immédiat}}(r)$

$r \xrightarrow{\varphi,a,up} r'$ $\quad \forall \varphi \in \mathcal{C},\ a \in \Sigma,\ up \in Up,\ \forall r, r'$ tels que $r \models \varphi$ et $r' \in \underline{up}(r)$

Tous les états de $\mathcal{R}$ sont considérés comme finaux. L'état initial correspond à la région contenant la valuation 0.

Définition 1.2.1 (Produit synchronisé) *Soient A, B deux automates finis sur l'alphabet Λ.*
On note $A \otimes B$ le produit synchronisé *de A et B. Il s'agit de l'automate fini ayant pour alphabet Λ, pour états $Q_A \times Q_B$ et pour transitions $(q_A, q_B) \xrightarrow{a} (q'_A, q'_B)$ chaque fois que $q_A \xrightarrow{\varepsilon^* a} q'_A$ [3] est un chemin de A et que $q_B \xrightarrow{\varepsilon^* a'} q'_B$ est un chemin de B. Un état (q_A, q_B) est initial si q_A et q_B le sont, il est final si q_A et q_B le sont.*

Définition 1.2.2 *Soit $\pi_\Sigma : \mathcal{C} \times \Sigma \times Up \to \Sigma^*$ tel que $\pi_\Sigma(\varphi, a, up) = a$, prolongé en tant que morphisme sur $(\mathcal{C} \times \Sigma \times Up)^*$ la* projection sur Σ^*.

Théorème 1.2.2 *Soient $\mathcal{A}$ un automate temporisé de $\boldsymbol{AT}(\mathcal{C}, Up)$ où $\mathcal{C}$ (resp. Up) est un ensemble fini de contraintes (resp. de mises à jour), et R un ensemble fini de régions compatible. Soit A une copie de $\mathcal{A}$ vue comme un automate fini pour l'alphabet $\mathcal{C} \times \Sigma \times Up$. Alors l'automate fini $A \otimes \mathcal{R}$ (appelé automate des régions) reconnaît un langage L tel que $\pi_\Sigma(L) = \text{UNTIME}(L(\mathcal{A}))$.*

Remarque En fait, nous venons de construire une bisimulation (abstraction faite du temps) entre un automate temporisé et l'automate des régions. Cette bisimulation respecte les états initiaux et finaux.
Reconnaître un langage non vide est une propriété respectée par une telle bisimulation, d'où la conclusion : R est fini, donc le produit synchronisé est un automate fini, donc on sait décider si son langage reconnu est vide, ce qui revient à décider le vide pour l'automate temporel initial.
Cette construction permet de conclure pour tous les autres problèmes qui passent par bisimulation et qui sont résolus dans le cas d'un automate fini, mais pas pour ceux qui ne le sont pas. Par exemple "l'état final peut être atteint en moins de 10 secondes" est un problème que la bisimulation faite ci-dessus ne permet pas de décider.

[3] $\varepsilon^* a$ correspond à un chemin empruntant un nombre fini d'ε-transitions suivi d'une transition étiquetée par a.

Notons que c'est l'existence d'une partition compatible qui permet d'avoir cette bisimulation. Le fait qu'elle soit finie permet juste de conclure pour le problème du vide.

Cette technique a permis entre autre de montrer la décidabilité des classes $\mathbf{AT}(\mathcal{C}_{rect}, \{x := c,\ x \in X\})$, $\mathbf{AT}(\mathcal{C}_{rect}, \{x := y + c,\ x, y \in X,\ c \in \mathbb{Q}\})$, $\mathbf{AT}(\mathcal{C}_{rect}, \{x := x + 1,\ x \in X\})$, $\mathbf{AT}(\mathcal{C}^- \cup \mathcal{C}_{rect}, \{x := c,\ x \in X,\ c \in \mathbb{Q}\})$ [BDFP04].

1.3 Classes indécidables

De même que la décidabilité des classes ci-dessus est montrée en se ramenant au même problème pour un automate fini, l'indécidabilité d'autres classes peut-être montrée en réduisant un problème indécidable au problème du vide pour ces classes.

Définition 1.3.1 (Machine à compteurs) *Une machine à compteurs est un n-uplet* (I, C, i_n) *où*

- I *est un ensemble fini d'instructions*
- C *est un ensemble fini de compteurs*
- $i_n \in I$ *est l'instruction finale*

Chaque instruction est de l'une des formes suivantes :

- Incrémenter c. Aller en i_k.
- Si $c = 0$, aller en i_k. Sinon, décrémenter c et aller en i_l.

où $c \in C$ *et* $i_k,\ i_l \in I$.

Théorème 1.3.1 *[Min67] Il n'existe pas d'algorithme qui, étant donnée une machine à deux compteurs, répond* OUI *si et seulement si la machine s'arrête.*

Remarque On appelle *machine de Minsky* une machine à deux compteurs.

Pour montrer l'indécidabilité d'une classe d'automates temporisés $\mathbf{AT}$, on peut montrer qu'il est possible de construire pour toute machine de Minsky $\mathcal{M}$ un automate de cette classe la simulant : pour tout $\mathcal{M}$ on construit $\mathcal{A}_{\mathcal{M}}$, automate de la classe $\mathbf{AT}$ tel que $\mathcal{M}$ s'arrête si et seulement si $L(\mathcal{A}_{\mathcal{M}}) \neq \emptyset$.

Une telle méthode permet de montrer l'indécidabilité des classes $\mathbf{AT}(C_{rect}, \{x := x - 1,\ x \in X\})$, $\mathbf{AT}(C_{rect} \cup \mathcal{C}^-, \{x := x + 1,\ x \in X\})$ [BDFP04].

1.4 Exemples

Nous allons étudier à présent plus en détails deux exemples d'étude de la décidabilité d'une classe d'automates temporisés tirés des travaux de [BD00], et qui sont à l'origine de ce stage.

Définition 1.4.1 *On notera* $\boldsymbol{AT}_n(\mathcal{C}^+, \mathcal{U}_0)$ *la classe des automates temporisés à contraintes additives avec n horloges. (définition de* $\mathcal{C}^+$ *page 6)*

Un intérêt de cette classe est que pour deux horloges, le problème du vide est décidable, alors que pour quatre, il est indécidable. Pour l'instant, le cas à trois horloges est un problème ouvert.

Cette classe permet par exemple de reconnaître le langage $\{(a^n, t_1 \ldots t_n) \mid n \geq 1 \text{ and } t_i = 1 - \frac{1}{2^i}\}$ (Figure 1.1).

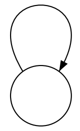


FIG. 1.1 – Un automate de $\mathbf{AT}_n(\mathcal{C}^+, \mathcal{U}_0)$

Classe $\mathbf{AT}_2(\mathcal{C}^+, \mathcal{U}_0)$

Théorème 1.4.1 *[BD00] La classe* $\boldsymbol{AT}_2(\mathcal{C}^\pm, \mathcal{U}_0)$ *est décidable.*

En effet, soit M la plus grande constante apparaissant dans les contraintes d'horloges. Alors on sait construire une partition finie de $\mathcal{R}_+^2$ qui est compatible avec l'écoulement du temps, l'ensemble des contraintes et l'ensemble des remises à zéro.
La figure 1.2 montre la partition obtenue pour $M = 2$.

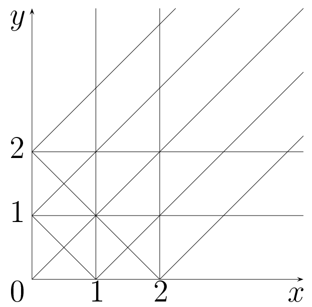


FIG. 1.2 – Régions pour des contraintes de $\mathcal{C}^\pm$

Classe $\mathbf{AT}_4(\mathcal{C}^+, \mathcal{U}_0)$

Théorème 1.4.2 *[BD00] La classe* $\boldsymbol{AT}_4(C^+, \mathcal{U}_0)$ *est indécidable.*

Pour montrer cela, on réduit le problème de l'arrêt d'une machine de Minsky au problème du vide pour un automate de cette classe. La réduction que nous allons voir est reprise de [BD00], mais a été simplifiée afin de vérifier que les quatre horloges sont nécessaires. (Rappelons que le problème "Étant donné un automate temporisé à n horloges, construire un automate à $n-1$ horloges reconnaissant le même langage." est indécidable [Tri03].)

Soit $M = (I, C, i_n)$, $C = \{c, d\}$ une machine de Minsky. Soit $\mathcal{A} = (X, Q, \Delta, I, F)$ l'automate temporisé défini ainsi :

Alphabet : $\Sigma = \{\varepsilon\}$.

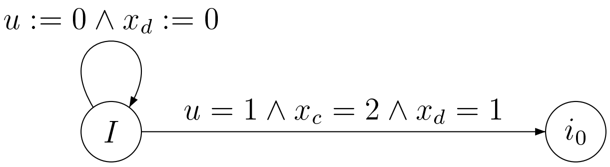
(En pratique, nous ne mettrons pas les ε sur les transitions.)

Horloges : $X = \{x_c, x_d, u, t\}$.

u est une horloge qui compte les unités de temps. Une instruction va être codée sur deux unités de temps : la première met à jour c, la deuxième met à jour d. x_c et x_d codent les valeurs des compteurs de la façon suivante : lorsque la première transition d'une instruction est franchie, $u = 1$, $x_c = 1 + \frac{1}{2^c}$ $(= 2 - \sum_{k=1}^{c} \frac{1}{2^k})$ et $x_d = \frac{1}{2^d}$. t est une horloge "de travail", qui permettra de décrémenter les compteurs.

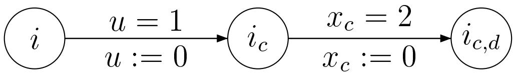
États : A chaque instruction $i \in I$ correspondent un état de départ i, un état intermédiaire $i_{c,d}$ (si c est testé à zéro, on a également $i_{0,d}$) dans lequel c aura été éventuellement modifié mais pas d, et trois (resp. deux) états de travail si l'instruction incrémente (resp. décrémente) un compteur.

On ajoute un état initial I et une transition entre I et i_0 (instruction initiale de $\mathcal{M}$) mettant u à 1, x_c à 2 et x_d à 1.

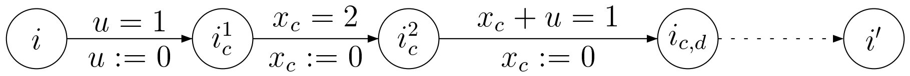


Transitions : Nous représentons ci-dessous les modules simulant les instructions. Seule la moitié de chaque instruction (correspondant au premier compteur) est représentée : la seconde moitié se construit sur le même principe en partant de $i_{c,d}$, en remplaçant c par d et en arrivant dans l'état correspondant à l'instruction suivante. Nous donnons également une idée de comment marchent les modules pour exécuter la première moitié d'une instruction i envoyant vers i' (et éventuellement vers i'').

– i ne modifie pas c



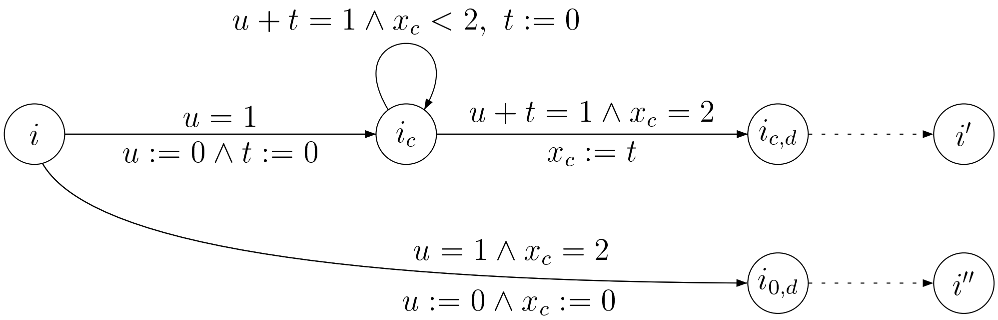
– i incrémente c



							instruction suivante
$u =$	1		$1 - \frac{1}{2^n}$		$1 - \frac{1}{2^{n+1}}$		1
$x_c =$	$1 + \frac{1}{2^n}$		2		$\frac{1}{2^{n+1}}$		$1 + \frac{1}{2^{n+1}}$
délais :		$1 - \frac{1}{2^n}$		$\frac{1}{2^{n+1}}$		$\frac{1}{2^{n+1}}$	

Les valeurs de u et x_c données sous l'automate correspondent aux valeurs des valuations au moment où chaque transition est franchie, pour le cas $c = n$. Les délais sont ceux passés dans chaque état.

– i teste c à zéro et le décrémente si $c > 0$



		$l^{\text{ème}}$ itération, $l < n$		instruction suivante
$u =$	1	$1 - \frac{1}{2^l}$	$\sum_{k=1}^{n-1} \frac{1}{2^k}$	1
$t =$		$\frac{1}{2^l}$	$\frac{1}{2^{n-1}}$	
$x_c =$	$1 + \frac{1}{2^n}$	$2 - \sum_{k=l+1}^{n} \frac{1}{2^k}$	2	$1 + \frac{1}{2^{n-1}}$

Les conventions concernant les valeurs de u, x_c et t sont les mêmes que précédemment. Seule la partie haute de l'automate est considérée.

La décrémentation utilise une mise à jour de la forme $x := y$, qui n'appartient pas à celles autorisées dans cette classe. Mais les travaux de [BDFP04] montrent qu'une telle mise à jour peut toujours[4] être simulée sans augmenter le nombre d'horloges.

[4]Le résultat cité concerne des classes sans contraintes additives, mais il peut être adapté sans difficultés au cas étudié ici.

Qu'en est-il de $\mathbf{AT}_3(\mathcal{C}^+, \mathcal{U}_0)$?

L'objectif de la présentation de ce codage d'une machine de Minsky par un automate de la classe $\mathbf{AT}_3(\mathcal{C}^+, \mathcal{U}_0)$, simplification de la construction faite dans [BD00], était de voir s'il n'était pas possible d'en obtenir un avec trois horloges seulement. Mais la quatrième horloge, t, semble être nécessaire. Nous n'avons pas trouvé d'autre moyen de décrémenter les compteurs en utilisant uniquement les contraintes et mises à jour de cette classe.

Par contre, il est possible de faire l'économie de cette horloge en ajoutant des mises à jour ou des gardes de types différents :

- Il pourrait être intéressant d'étudier une mise à jour de la forme $x := 1 - u$ qui permet de faire la même chose (Figure 1.3). Peut-on alors se passer de $x + y = 1$ pour faire l'incrémentation ? Si oui, une telle mise à jour rend une classe à trois horloges indécidable.
- Une autre solution consiste à rajouter à $\mathcal{C}^+$ la garde $u = 2x - 1$. Alors avec trois horloges, on parvient à simuler une machine de Minsky (Figure 1.4) : $\mathbf{AT}_3(\{x + y = 1,\ x + z = 1,\ x = 2y - 1,\ x = 2z - 1\} \cup \mathcal{C}_{rect}, U_0)$ est une classe indécidable.

Une autre piste serait de chercher un autre codage.

Enfin, on est en droit de se demander si cette classe n'admet pas une bisimulation finie, ce qui permettrait de prouver qu'elle est décidable. C'est ce que nous allons voir dans le chapitre suivant.

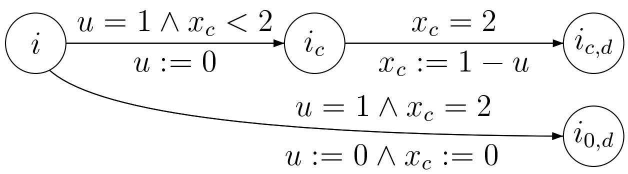


FIG. 1.3 – Décrémentation de c en utilisant $x := 1 - u$

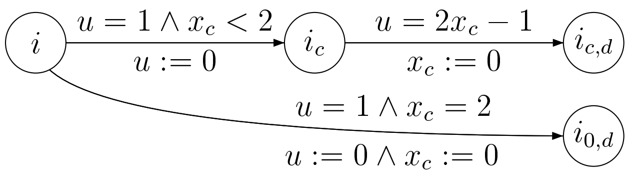


FIG. 1.4 – Décrémentation de c en utilisant $u = 2x - 1$

Chapitre 2

Ensemble infini de régions

2.1 Nécessité d'une bisimulation d'indice fini

Les résultats de décidabilité obtenus jusqu'à présent reposent sur l'existence d'une partition finie de l'ensemble des valuations d'horloges. En fait, cette partition permet d'établir une bisimulation entre l'automate temporisé et le produit synchronisé de l'automate des régions et d'un automate fini. Le fait que la partition soit finie sert uniquement à conclure que l'accessibilité est décidable, car c'est le cas pour les automates finis. Pour savoir quand ce résultat n'est pas applicable, il faudrait avoir un moyen d'être sûr qu'il n'existe pas de bisimulation finie.

Définition 2.1.1 (bisimulation timed-abstract) *Soient $\mathcal{A}$ un automate temporisé, $\mathcal{B}$ un système de transitions. $\mathcal{A}$ et $\mathcal{B}$ sont* bisimilaires *s'il existe une relation d'équivalence $\approx$ entre les configurations de $\mathcal{A}$ ($(q,v) \in Q \times \mathbb{T}^X$) et celles de $\mathcal{B}$ ($c \in C_{\mathcal{B}}$) vérifiant :*
$\forall\ (q,v) \in Q \times \mathbb{T}^X,\ c \in C_{\mathcal{B}}$, si $(q,v) \approx c$ alors

- *si $(q,v) \xrightarrow{\alpha} (q',v')$, alors il existe $c' \in C_{\mathcal{B}}$ tel que $c \xrightarrow{\alpha} c'$ et $(q',v') \approx c'$;*
- *si $(q,v) \xrightarrow{\tau} (q, v+\tau)$, où $\tau > 0$ et $q' \in Q_{\mathcal{A}}$, alors il existe $\tau' > 0$ et $c' \in C_{\mathcal{B}}$ tel que $c \xrightarrow{\tau'} c'$ et $(q, v+\tau) \approx c'$.*

Réciproquement,

- *si $c \xrightarrow{\alpha} c'$, alors il existe $q' \in Q_{\mathcal{A}},\ v' \in \mathbb{T}^X$ tels que $(q,v) \xrightarrow{\alpha} (q',v')$ et $(q',v') \approx c'$.*

Définition 2.1.2 *Une bisimulation $\approx$ est* finie *si elle a un nombre fini de classes d'équivalence. Elle* correspond à une partition *$\mathcal{P}$ de Q si tout $P \in \mathcal{P}$ est une union de classes d'équivalence de $\approx$.*
Soit P une classe d´équivalence de $\approx$. Alors $Pre(P) = \{q \in Q \mid (\exists q' \in P\ q \xrightarrow{\alpha} q')$ ou $(\exists q' \in P\ \exists \tau > 0\ q \xrightarrow{\tau} q')\}$. Il s'agit d'une union de classes d'équivalence.

Remarque En fait, on distinguera pour la suite Pre_{temp} qui correspond aux antécédents pour une progression dans le temps, et $\forall up \in Up,\ Pre_{up}$ qui correspond aux antécédents pour la une mise à jour up.

Algorithme 2.1.1 (Bisimulation la plus grossière respectant la partition $\mathcal{P}_0$)
Initialisation $\mathcal{P} := \mathcal{P}_0$;
Tant que *il existe $P, P' \in \mathcal{P}$ tels que $\varnothing \subsetneq P \cap Pre(P') \subsetneq P$, faire*

$P_1 := P \cap Pre(P'), P_2 := P \setminus Pre(P')$;
$\mathcal{P} := (\mathcal{P} \setminus \{P\}) \cup \{P_1, P_2\}$.

RENVOYER $\mathcal{P}$.

Théorème 2.1.1 *[BBR04] Soit $\mathcal{A}$ un automate temporisé. L'algorithme 2.1.1 termine si et seulement si la plus grossière bisimulation de $\mathcal{A}$ respectant une partition $\mathcal{P}$ est finie.*

Pour montrer que le résultat de décidabilité utilisant un automate des régions fini ne s'applique pas à la classe $\mathbf{AT}_3(\mathcal{C}^+, U_0)$, il suffit donc de montrer que l'algorithme 2.1.1 ne termine pas lorsqu'on l'applique à la partition $\mathcal{P}_0$ correspondant à $\mathcal{C}^+$.

2.2 Partition compatible pour $\mathbf{AT}_3(\mathcal{C}^+, \mathcal{U}_0)$?

Intuition

Sur un croquis du cube représentant les contraintes $x = 1,\ y = 1,\ z = 1$ (Figure 2.1), on a l'impression qu'en alternant $Pre_{[x:=0]}$ et Pre_{temps}, on crée à chaque fois de nouvelles régions. Regardons par exemple ce que donne le droite $(x = 0 \wedge z = 1)$:

On pose $P = \{x + y = 1\},\ P' = \{x = 0 \wedge z = 1\}$ pour obtenir par $Pre_{[x:=0]}$ une nouvelle région $\Delta_0 = P \cap Pre_{[x:=0]}(P')$ à ajouter à $\mathcal{P}$.
Puis on pose $P = \{x = 0\},\ P' = \Delta_0$, et on obtient par Pre_{temps} une nouvelle région D_1. Et on recommence avec $P = \{x + y = 1\},\ P' = D_1$[1].

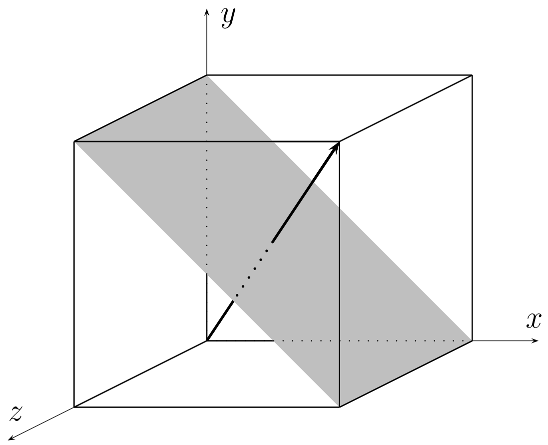


FIG. 2.1 – le cube et la contrainte $\{x + y = 1\}$

[1]Nous ne sommes pas exactement dans le cadre de l'algo 2.1.1, vu que $\{x+y = 1\}$ et $\{x = 0\}$ ne sont pas des régions de $\mathcal{P}$ mais des unions de régions. Cepandant, on pourrait à chaque fois extraire de ces unions deux régions correspondant plus précisemment à l'algorithme.

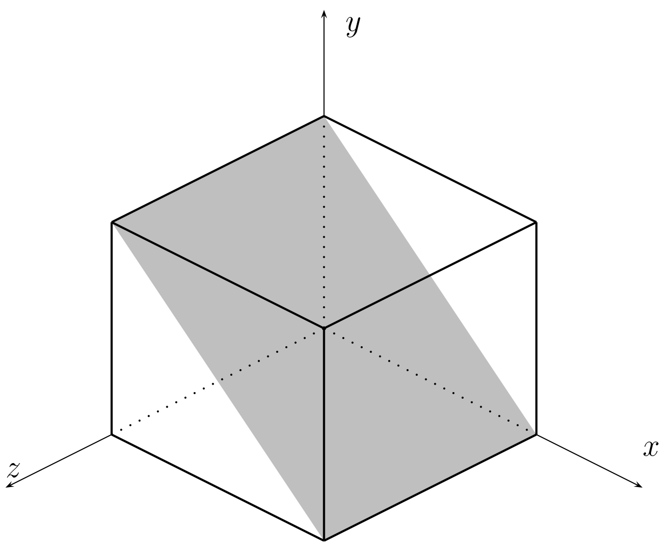


FIG. 2.2 – le même, vu dans l'axe du temps

Simulation sous HYTECH

L'algorithme ne termine pas si les droites Δ_n et D_n sont toutes distinctes. Afin d'avoir une idée de leurs équations, nous avons simulé le début du calcul sous HYTECH [HHWT95]. Les résultats sont dans le tableau 2.1.

On part de :
x = 0 & y >= 0 & z >= 0 & y < 1 & z < 1
...Number of iterations required for reachability : 3
....Number of iterations required for reachability : 4
....Number of iterations required for reachability : 4

Les chemins sortant d'abord du cube (z=1) partent de :
z = 0 & y>=0 & z < 1 & 2z >= y + 1
Les chemins passant d'abord par x+y=1 partent de :
x = 0 & y >= 0 & z >= 0 & z < 1 & y < 1 & 2z <= y + 1
..Number of iterations required for reachability : 2
..Number of iterations required for reachability : 2

D'où la nouvelle région de départ :
y >= 0 & z >= 0 & z < 1 & y < 1 & 2z <= y + 1
et la nouvelle contrainte :
y >= 0 & 2z >= y + 1 & 2z < 2

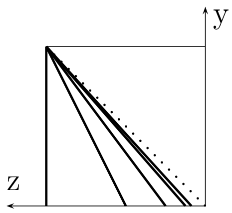
On part de :
x = 0 & y >= 0 & z >= 0 & z < 1 & y < 1 & 2z <= y + 1
...Number of iterations required for reachability : 3
....Number of iterations required for reachability : 4
....Number of iterations required for reachability : 4

Les chemins rencontrant d'abord C partent de :
x = 0 & 4z >= 3y + 1 & y >= 0 & 4z < 4 & 4z <= 2y + 2
Les chemins passant d'abord par z+y=1 partent de :
x = 0 & y >= 0 & z >= 0 & z < 1 & y < 1 & 4z <= 3y + 1
..Number of iterations required for reachability : 2
..Number of iterations required for reachability : 2

D'où la nouvelle région de départ :
y >= 0 & z >= 0 & z < 1 & y < 1 & 4z <= 3y + 1
et la nouvelle contrainte :
4z >= 3y + 1 & 2z <= y + 1 & z < 1 & y >= 0

TAB. 2.1: Résultats sous HYTECH

HYTECH donne donc l'intuition des équations des droites obtenues. Si cette intuition se vérifie, nous aurons bien une infinité de régions distinctes.



Démonstration mathématique

Posons $D_0 = \{z = 1 \wedge x = 0\}$. Il s'agit bien d'une région.
Définissons récursivement

$$D_{n+1} = Pre_{temps}[([x := 0]^{-1} D_n) \cap P] \cap \{x = 0\}$$

Il est évident que si D_n est une union de régions, alors D_{n+1} doit en être une aussi.
Hypothèse :

$$\forall n, D_n = \{(2^n - 1)y = 2^n z - 1\} \cap \{x = 0\} \qquad (\clubsuit)$$

Si ($\clubsuit$) est vérifié pour tout n, alors toute bisimulation respectant la partition donnée directement par les contraintes de départ sera infinie (théorème 2.1.1).
Montrons ($\clubsuit$) par recurrence :
D_0 vérifie ($\clubsuit$).
Soit $n \geq 0$. Supposons que D_n vérifie ($\clubsuit$).
Alors

- $[x := 0]^{-1} D_n = \{(2^n - 1)y = 2^n z - 1\}$
- $([x := 0]^{-1} D_n) \cap P = \left\{ \begin{array}{l} (2^n - 1)y = 2^n z - 1 \\ x + y = 1 \end{array} \right.$
- $Pre_{temps}[([x := 0]^{-1} D_n) \cap P] = \{(x, y, z) \mid \exists t, \left\{ \begin{array}{l} (2^n - 1)(y + t) = 2^n (z + t) - 1 \\ x + y + 2t = 1 \end{array} \right. \}$
- $Pre_{temps}[([x := 0]^{-1} D_n) \cap P] \cap \{x = 0\} = \{(0, y, z) \mid \exists t, \left\{ \begin{array}{l} (2^n - 1)(y + t) = 2^n (z + t) - 1 \\ y + 2t = 1 \end{array} \right. \}$
- $D_{n+1} = \{(2^{n+1} - 1)y = 2^{n+1} z - 1\} \cap \{x = 0\}$ $\square$

Conclusion

Il n'existe donc pas de partition finie de l'ensemble des valuations correspondant à un automate temporisé à 3 horloges x, y, z ayant pour contraintes $\{z = 1\}$, $\{x = 0\}$, $\{x + y = 1\}$ et pour mise à jour $[x := 0]$, qui respecte les conditions de compatibilité ($\star$), ($\dagger$) et ($\ddagger$).

Théorème 2.2.1 *En général, il n'existe pas de bisimulation finie d'un automate temporisé de la classe* $\boldsymbol{AT}_3(\mathcal{C}^+, \mathcal{U}_0)$.

Chapitre 3

Automates à compteur

Jusqu'à présent, le comportement temporisé était simulé par un automate fini. L'avantage d'une telle simulation est sa simplicité (les automates finis forment une classe décidable), mais c'est aussi un inconvénient : dès que les possibilités offertes par les contraintes et les mises à jour sont trop complexes, un automate fini ne permet plus de simuler le comportement d'un automate temporisé.
Pourtant, nous connaissons des modèles décidables plus riches que les automates finis : par exemple les automates à un compteur, ou les automates à une pile. L'idée que nous allons développer ici consiste à considérer le cas où le graphe des régions peut être simulé par un automate à un compteur.

3.1 Définitions et propriétés

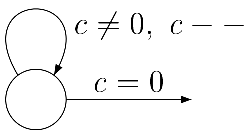
Résultat fondamental

Théorème 3.1.1 *Considérons le problème suivant : « Étant donnée une machine à compteurs, l'état final est-il accessible ? »*
Ce problème est décidable pour les machines à un compteur (résultats cités dans [HU79]), indécidable pour les machines à plus de deux compteurs [Min67].

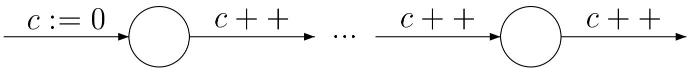
Automate à un compteur

On peut représenter une machine à un compteur par un automate dont les transitions contiendraient des instructions sur le compteur : test à zéro, incrémentation, décrémentation.
Pour alléger la suite, nous allons montrer que l'on peut également simuler certaines instructions.

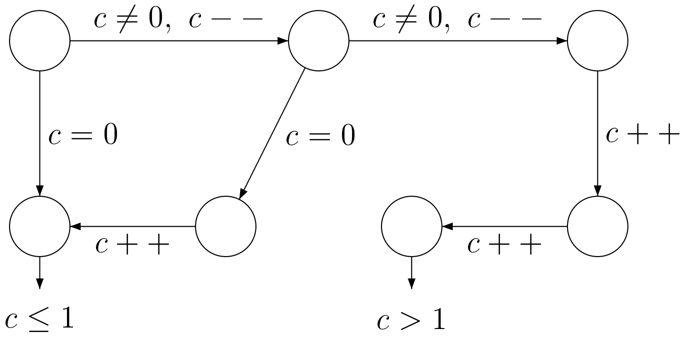
– Remise à zéro du compteur



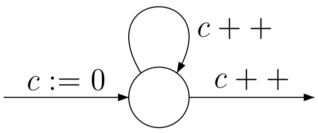
– Remise à une constante donnée k



– Test >1



– Mise à une valeur strictement positive



3.2 Bisimulation

Introduction

Nous avons vu en partie 2 qu'il arrivait qu'on ne puisse pas conclure à la décidabilité d'une classe quand celle-ci possédait un graphe des régions infini, mais qu'il ne semblait pas y avoir non plus de réduction vers une machine de Minsky.

L'idée de la preuve qui suit est d'essayer de construire alors un automate des régions qui ne soit pas fini, mais un automate à un compteur. Cela semble possible lorsque le graphe des régions est infini, mais régulier : si sa partie infinie consiste en un motif répété régulièrement, il semble faisable de la simuler par un compteur précisant dans quel motif on se trouve.

Nous allons à présent poser les hypothèses suffisantes pour que le graphe des régions soit bisimilaire à un automate à pile, puis nous allons vérifier qu'il l'est effectivement.

Notations

Soit $\mathcal{A}$ un automate temporisé ($= (\Sigma, X, \mathcal{C}_{\mathcal{A}}, Up, I, F)$, $\mathcal{C}_{\mathcal{A}}$ sous-ensemble fini de $\mathcal{C}(X)$, Up sous-ensemble fini de $Up(X)$). Soit R une partition de l'ensemble des valuations compatible avec les contraintes, l'avancée dans le temps et les mises à jours de $\mathcal{A}$.
Si R existe et est dénombrable, on sait que[1] $\mathcal{A}$ est en bisimulation avec $A \times \mathcal{R}$, où :

A est un automate fini, copie exacte de $\mathcal{A}$ pour l'alphabet $\mathcal{C}_{\mathcal{A}} \times \Sigma \times Up(X)$

$\mathcal{R}$ est un automate infini sur cet alphabet, avec pour états les éléments de R, et pour transitions :

[1] Voir la remarque page 10

- $r_1 \xrightarrow{\varepsilon} r_2$ chaque fois que r_2 est le successeur **immédiat** de r_1,
- $r_1 \xrightarrow{\varphi,\Sigma,up} r_2$ tels que $r_1 \models \varphi,\ \varphi \in \mathcal{C}_\mathcal{A}$ et $r_2 \in up(r_1),\ up \in Up(X)$.

Il peut arriver qu'un élément r_0 de R n'ait aucun successeur immédiat, mais ait des successeurs que $\mathcal{A}$ permet d'atteindre en traversant une infinité de régions (exemple sur la figure 3.2). Dans ce cas, pour que la bisimulation soit possible, on doit ajouter les transitions :

- $r_0 \xrightarrow{\varepsilon^\omega} r$ entre r_0 et ses successeurs.

(on agit de même lorsqu'une région n'est successeur immédiat d'aucune autre mais peut-être atteinte en laissant passer du temps à partir d'une infinité de régions.)

On note $\Delta_\mathcal{R}$ l'ensemble des transitions de $\mathcal{R}$, $S_\mathcal{R}$ l'ensemble de ses états initiaux.

Hypothèses

Par la suite, on notera $\sqcup$ l'union disjointe de deux ensembles.
On suppose que R est de la forme : $P \sqcup \bigsqcup_{n\geq 0} f^n(M)$, où P et M sont des unions finies de régions de R, $P \supseteq S_\mathcal{R}$, et f une fonction injective de $\mathrm{R}\backslash P$ dans $\mathrm{R}\backslash P$.
Pour qu'un compteur suffise à représenter le passage d'un motif au suivant, il faut que ceux-ci respectent une certaine compatibilité avec la progression du temps, les contraintes et les mises à jour. En particulier, on ne considérera que deux sortes de mises à jour : les mises à jour *locales*, qui restent à l'intérieur d'un même motif ou vont d'une région d'un motif à la même région d'un motif voisin, et les *globales*, qui amènent dans la partie finie.
Plus formellement, on suppose que $Up = Up_{loc} \sqcup Up_0$, où $\forall up \in Up_0,\ \forall p \in \mathrm{R},\ up(p) \in P$, et que f vérifie les conditions suivantes :
$\forall\, r, r' \in \mathrm{R}\backslash P, \forall\, p \in P,$

$$\begin{array}{ll}
(\bullet) & r' \in Succ_{\texttt{immédiat}}(r) \iff f(r') \in Succ_{\texttt{immédiat}}(f(r)) \\
(\bullet_\omega) & r' \in Succ_{\texttt{infini}}(p) \iff f(r') \in Succ_{\texttt{infini}}(p) \\
 & p \in Succ_{\texttt{infini}}(r) \iff p \in Succ_{\texttt{infini}}(f(r)) \\
(\circ) & \forall up \in Up_{loc}(X), \quad r' \in up(r) \iff f(r') \in up(f(r)) \\
 & \forall up \in Up_0(X), \quad p \in up(r) \iff p \in up(f(r)) \\
(\star) & \forall \varphi \in \mathcal{C}_\mathcal{A}, r \subseteq [\![\varphi]\!] \iff f(r) \subseteq [\![\varphi]\!]
\end{array}$$

Enfin, on suppose que $(\diamond)$ $\forall up,\ \forall r \in M \cup P,\ \forall r' \in M,\ r \neq r',\ \forall n > 0,\ f^n(r') \notin up(r),\ f^n(r') \notin Succ_{\texttt{immédiat}}(r),\ r \notin Succ_{\texttt{immédiat}}(f^n(r'))$ et si $r \in M$ ou $up \in Up_{loc}$, $r \notin up(f^n(r'))$ (par contre on peut avoir $r \in P$, $up \in Up_0$ et $r \in up(f^n(r'))$),
et que $(\triangleleft)$ il n'y a pas de transitions étiquetées ε^ω entre deux éléments de $\mathrm{R}\backslash P$.

Construction de $\mathcal{A}_\mathcal{R}$

À toute région r de $P \sqcup M$ on associe un état $\overline{r}$ de $\mathcal{A}_\mathcal{R}$.
Les configurations initiales de $\mathcal{A}_\mathcal{R}$ sont : $S_{\mathcal{A}_\mathcal{R}} = \{(\overline{r}, 0) | r \in S_\mathcal{R}\}$.
On considère toutes les paires (r, r') de $P \cup M$. Si $\mathcal{R}$ possède une transition de la forme $r \xrightarrow{\alpha} r'$, alors on construit la transition suivante dans $\mathcal{A}_\mathcal{R}$:

	$r' \in P$	$r' \in M$	$r' = f(r'_0), r'_0 \in M$
$r \in P$	$\overline{r} \xrightarrow{\alpha} \overline{r'}$	*si* $\alpha = \varepsilon^\omega$, $\overline{r} \xrightarrow[n:>0]{\alpha} \overline{r'}$; sinon, $\overline{r} \xrightarrow[n:=1]{\alpha} \overline{r'}$.	$\emptyset$
$r \in M$	si $up_\alpha \in Up_0$, $\overline{r} \xrightarrow[n:=0]{\alpha} \overline{r'}$ sinon, $\overline{r} \xrightarrow[n=1,\ n:=0]{\alpha} \overline{r'}$	$\overline{r} \xrightarrow{\alpha} \overline{r'}$	$\overline{r} \xrightarrow[n++]{\alpha} \overline{r'_0}$
$r = f(r_0), r_0 \in M$	$\emptyset$	$\overline{r_0} \xrightarrow[n>1,\ n--]{\alpha} \overline{r'}$	$\emptyset$

$\mathcal{R}$ et $\mathcal{A}_\mathcal{R}$ sont en bisimulation

Définition 3.2.1 (bisimulation) *Soient $\mathcal{A}$ et $\mathcal{B}$ deux systèmes de transitions. $\mathcal{A}$ et $\mathcal{B}$ sont* bisimilaires *s'il existe une relation d'équivalence $\sim$ entre les configurations de $\mathcal{A}$ ($C_\mathcal{A}$) et celles de $\mathcal{B}$ ($C_\mathcal{B}$) vérifiant :*
$\forall\ p \in C_\mathcal{A},\ q \in C_\mathcal{B}$, si $p \sim q$ alors
- *si $p \xrightarrow{\alpha} p'$, alors il existe $q' \in C_\mathcal{B}$ tel que $q \xrightarrow{\alpha} q'$ et $p' \sim q'$;*
- *si $q \xrightarrow{\alpha} q'$, alors il existe $p' \in C_\mathcal{A}$ tel que $p \xrightarrow{\alpha} p'$ et $p' \sim q'$.*

On veut montrer que : $\forall r \in \mathrm{R},\ r \sim (\overline{p}, n)$ où
- $r = p \in P$ et $n = 0$
  ou bien
- $r = f^{n-1}(p)$, $p \in M$ et $n > 0$.

Lemme 3.2.1 *Soit $(\overline{p}, n)$ une configuration accessible de $\mathcal{A}_\mathcal{R}$. Alors $p \in P \iff n = 0$.*

Preuve par induction sur la longueur d'un plus court chemin menant à $(\overline{p}, n)$:
- si $(\overline{p}, n)$ est une configuration initiale, alors $p \in P$ et $n = 0$;
- sinon, supposons que toutes les configurations $(\overline{p'}, n')$ accessibles par un chemin plus court que $(\overline{p}, n)$ sont telles que $p' \in P \iff n' = 0$. Considérons la transition qui finit un plus court chemin menant à $(\overline{p'}, n')$. Elle est de la forme : $(\overline{p_0}, n_0) \xrightarrow[n:=up(n_0)]{\alpha} (\overline{p}, n)$.
  Supposons que $p \in P$. Alors soit $p_0 \in P$, et alors $up = Id$, $n = n_0$ et par induction $n_0 = 0$, donc $n = 0$. Soit $p_0 \in M$, et alors $up = [n := 0]$, donc $n = 0$;
  Supposons que $n = 0$. Alors $up = Id$ ou $up = [n := 0]$. Dans le premier cas, $n_0 = 0$ donc par induction $p_0 \in P$: on se trouve dans la case $p_0, p \in P$. Dans le second cas, on est dans la case $p_0 \in M,\ p \in P$. Donc $p \in P$;
- donc par récurrence, si $(\overline{p}, n)$ est accessible, $p \in P \iff n = 0$. $\square$

$\mathcal{R}$ simule $\mathcal{A}_\mathcal{R}$ pour $\sim$ On considère une transition entre deux configurations dans l'exécution de $\mathcal{A}_\mathcal{R}$:

$$(\overline{p}, n) \xrightarrow[n':=up(n)]{\alpha} (\overline{p'}, n'),\ \alpha = \varepsilon,\ \alpha = \varepsilon^\omega \ ou\ \alpha = [\varphi_\alpha, \Sigma, up_\alpha]$$

On cherche à montrer que :

$$\exists\ r, r' \in \mathrm{R},\ r \sim (\overline{p}, n),\ r' \sim (\overline{p'}, n')\ tels\ que\ r \xrightarrow{\alpha} r' \in \Delta_\mathcal{R}$$

Si le compteur n'est pas mis à jour,

Par construction, on sait que $p \xrightarrow{\alpha} p' \in \Delta_{\mathcal{R}}$, avec soit $p, p' \in P$, soit $p, p' \in M$.
Le premier cas est immédiat.
Dans le deuxième cas, $n > 0$ (lemme 3.2.1) et $f^{n-1}(p) \xrightarrow{\alpha} f^{n-1}(p') \in \Delta_{\mathcal{R}}$ (par $\bullet$, $\circ$ et $\star$ lues dans le sens $\Longrightarrow$). En effet :

- $\alpha \neq \varepsilon^\omega$ ($\triangleleft$).
- Si $\alpha = \varepsilon$, alors $p' \in Succ_{\texttt{immédiat}}(p)$, et donc ($\bullet \Longrightarrow$) $f^{n-1}(p') \in Succ_{\texttt{immédiat}}(f^{n-1}(p))$.
- Si $\alpha = [\varphi_\alpha, \Sigma, up_\alpha]$, on a alors $p' \in up_\alpha(p)$, $up_\alpha \in Up_{loc}$ d'où ($\circ \Longrightarrow$) $f^{n-1}(p') \in up(f^{n-1}(p))$, et $p \subseteq [\![\varphi_\alpha]\!]$ d'où ($\star \Longrightarrow$) $f^{n-1}(p) \subseteq [\![\varphi_\alpha]\!]$.

Si le compteur est remis à 1 ($n := 1$),

Par construction, on sait que $p \in P$ et $p \xrightarrow{\alpha} p' \in \Delta_{\mathcal{R}}$. Or c'est ce qu'on voulait (car $n = 0$ (lemme 3.2.1) et $n' = 1$).

Si le compteur est mis à une valeur strictement positive ($n :> 0$),

Par construction, on sait que $p \in P$ et $p \xrightarrow{\varepsilon^\omega} p' \in \Delta_{\mathcal{R}}$. $(\overline{p}, n)$ est accessible donc $n = 0$. On est alors dans le cas $n' > 0$.
On sait (par $\bullet_\omega$ lue dans le sens $\Longrightarrow$) que $p \xrightarrow{\varepsilon^\omega} f^{n'-1}(p') \in \Delta_{\mathcal{R}}$.

Si le compteur est remis à 0 ($n := 0$),

Par construction, $p' \in P$, $p \in M$ et $p \xrightarrow{\alpha} p' \in \Delta_{\mathcal{R}}$. On a $n > 0$.
Si $\alpha = \varepsilon$, $n = 1$ (par $\diamond$).
Si $\alpha = \varepsilon^\omega$, $p' \in Succ_{\texttt{infini}}(f^{n-1}(p))$ (par $\bullet_\omega \Longrightarrow$).
Sinon, par $\circ$ et $\star$ lues dans le sens $\Longrightarrow$, $f^{n-1}(p) \in [\![\varphi_\alpha]\!]$ et $p' \in up_\alpha(f^{n-1}(p))$.
Donc $f^{n-1}(p) \xrightarrow{\alpha} p' \in \Delta_{\mathcal{R}}$.

Si le compteur est décrémenté ($n > 1, n--$),

Par construction, $p, p' \in M$ et $f(p) \xrightarrow{\alpha} p' \in \Delta_{\mathcal{R}}$.
On a $n \geq 2$, d'où (par $\bullet$, $\circ$ et $\star$ lues dans le sens $\Longrightarrow$), $f^{n-1}(p) \xrightarrow{\alpha} f^{n-2}(p') \in \Delta_{\mathcal{R}}$.

Si le compteur est incrémenté ($n++$),

Par construction, $p, p' \in M$ et $p \xrightarrow{\alpha} f(p') \in \Delta_{\mathcal{R}}$.
On a $n \geq 1$, d'où (par $\bullet$, $\circ$ et $\star$ lues dans le sens $\Longrightarrow$), $f^{n-1}(p) \xrightarrow{\alpha} f^{n-1}(p') \in \Delta_{\mathcal{R}}$.

$\mathcal{A}_{\mathcal{R}}$ simule $\mathcal{R}$ pour $\sim$ On considère une transition de $\mathcal{R}$:

$$r \xrightarrow{\alpha} r', \alpha = \varepsilon,\ \alpha = \varepsilon^\omega \ ou \ \alpha = [\varphi_\alpha, \Sigma, up_\alpha].$$

Si $r \in \mathrm{R} \backslash P$, il existe par hypothèse un unique $p \in M$, $n \geq 0$ tels que $r = f^n(p)$. Si $r \in P$, on pose $p = r$ et $n = -1$. On définit de même p' et n' à partir de r'.
Montrons qu'il existe une transition étiquetée α qui permet de passer de $(\overline{p}, n+1)$ à $(\overline{p'}, n'+1)$ dans l'exécution de $\mathcal{A}_{\mathcal{R}}$.

Cas $n = n'$:

Si $n = n' = -1$, $p, p' \in P$ et $p \xrightarrow{\alpha} p' \in \Delta_{\mathcal{R}}$.
Sinon, $p, p' \in M$ et $f^n(p) \xrightarrow{\alpha} f^n(p') \in \Delta_{\mathcal{R}}$. Par hypothèse, $\alpha \neq \varepsilon^\omega$. Par $\bullet$, si $f^n(p') \in Succ_{\texttt{immédiat}}(f^n(p))$, $p' \in Succ_{\texttt{immédiat}}(p)$, et alors $p \xrightarrow{\varepsilon} p' \in \Delta_{\mathcal{R}}$. Par $\circ$ et $\star$ lues dans le sens $\Longleftarrow$, si $f^n(p) \xrightarrow{\alpha} f^n(p')$, $p \xrightarrow{\alpha} p' \in \Delta_{\mathcal{R}}$.
Dans les deux cas, $\overline{p} \xrightarrow{\alpha} \overline{p'} \in \Delta_{Ap}$ par construction. On a donc bien $(\overline{p}, n+1) \xrightarrow{\alpha} (\overline{p'}, n+1)$ dans l'exécution de $\mathcal{A}_{\mathcal{R}}$.

Cas $n = -1$ $(p \in P)$ **:** $p \xrightarrow{\alpha} f^{n'}(p')$

Par $\diamond$, $n' > 0 \Longrightarrow \alpha = \varepsilon^\omega$.

Si $\alpha = \varepsilon^\omega$, $f^{n'}(p') \in Succ_{\texttt{infini}}(p)$ donc $(\bullet_\omega \Longleftarrow)$ $p' \in Succ_{\texttt{infini}}(p)$ et $p \xrightarrow{\varepsilon^\omega} p' \in \Delta_{\mathcal{R}}$.

Par construction, $p \xrightarrow[n:>0]{\varepsilon^\omega} p' \in \Delta_{Ap}$, donc $(\overline{p}, 0) \xrightarrow{\varepsilon^\omega} (\overline{p'}, n' + 1)$ est dans l'exécution de $\mathcal{A}_{\mathcal{R}}$.

Si $n' = 0$, $p \xrightarrow{\alpha} p' \in \Delta_{\mathcal{R}}$ donc par construction, $p \xrightarrow[n:=1]{\alpha} p' \in \Delta_{Ap}$. On a donc bien $(\overline{p}, 0) \xrightarrow{\alpha} (\overline{p'}, 1)$ dans l'exécution de $\mathcal{A}_{\mathcal{R}}$.

Cas $n' = -1$ $(p' \in P)$ **:** $f^n(p) \xrightarrow{\alpha} p'$

Si $\alpha = \varepsilon$, par $\diamond$, $n = 0$. Donc $p \xrightarrow{\varepsilon} p' \in \Delta_{\mathcal{R}}$.

Si $\alpha = \varepsilon^\omega$, $p' \in Succ_{\texttt{infini}}(f^n(p))$ donc $(\bullet_\omega \Longleftarrow)$ $p' \in Succ_{\texttt{infini}}(p)$ et $p \xrightarrow{\varepsilon^\omega} p' \in \Delta_{\mathcal{R}}$.

Si $\alpha = [\varphi_\alpha, \Sigma, up_\alpha]$, $up_\alpha \in Up_0$, $f^n(p) \subseteq [\![\varphi_\alpha]\!]$ et $p' \in up_\alpha(f^n(p))$ donc $(\star \Longleftarrow)$ $p \subseteq [\![\varphi_\alpha]\!]$ et $(\circ \Longleftarrow)$ $p' \in up_\alpha(p)$. D'où $p \xrightarrow{[\varphi_\alpha, \Sigma, up_\alpha]} p' \in \Delta_{\mathcal{R}}$.

Par construction, $\overline{p} \xrightarrow[n:=0]{\alpha} \overline{p'} \in \Delta_{Ap}$. On a donc bien $(\overline{p}, n + 1) \xrightarrow{\alpha} (\overline{p'}, 0)$ dans l'exécution de $\mathcal{A}_{\mathcal{R}}$.

Sinon, $\alpha = [\varphi_\alpha, \Sigma, up_\alpha]$, $up_\alpha \in Up_{loc}$. Par $\diamond$, $n = 0$. $p \xrightarrow{[\varphi_\alpha, \Sigma, up_\alpha]} p' \in \Delta_{\mathcal{R}}$ donc par construction $\overline{p} \xrightarrow[n=1, n:=0]{\alpha} \overline{p'} \in \Delta_{Ap}$. On a donc bien $(\overline{p}, 1) \xrightarrow{\alpha} (\overline{p'}, 0)$ dans l'exécution de $\mathcal{A}_{\mathcal{R}}$.

Cas $n' = n + 1$ **:** $f^n(p) \xrightarrow{\alpha} f^{n+1}(p')$, $n \geq 0$

Par $\triangleleft$, $\alpha \neq \varepsilon^\omega$. Par $\bullet$, $\circ$ et $\star$ lues dans le sens $\Longleftarrow$, $p \xrightarrow{\alpha} f(p') \in \Delta_{\mathcal{R}}$. Par construction, $\overline{p} \xrightarrow[n{+}{+}]{\alpha} \overline{p'} \in \Delta_{Ap}$.

On a donc bien $(\overline{p}, n + 1) \xrightarrow{\alpha} (\overline{p'}, n + 2)$ dans l'exécution de $\mathcal{A}_{\mathcal{R}}$.

Cas $n' = n - 1$ **:** $f^n(p) \xrightarrow{\alpha} f^{n-1}(p')$, $n \geq 1$

Par $\triangleleft$, $\alpha \neq \varepsilon^\omega$. Par $\bullet'$, $\circ$ et $\star$ lues dans le sens $\Longleftarrow$, $f(p) \xrightarrow{\alpha} p' \in \Delta_{\mathcal{R}}$. Par construction, $\overline{p} \xrightarrow[n>1, n{-}{-}]{\alpha} \overline{p'} \in \Delta_{Ap}$.

On a donc bien $(\overline{p}, n + 1) \xrightarrow{\alpha} (\overline{p'}, n)$ dans l'exécution de $\mathcal{A}_{\mathcal{R}}$

Sinon... on a $p, p' \in M$ et $|n - n'| > 1$.

Supposons que $n' > n$. Alors $\exists k > 1$, $p \xrightarrow{\alpha} f^k(p') \in \Delta_{\mathcal{R}}$. Mais on ne peut avoir ni $\alpha = \varepsilon$, ni $\alpha = [\varphi, \Sigma, up]$ (par $\diamond$). Donc $\alpha = \varepsilon^\omega$. Absurde (car $p, p' \in M$). On raisonne de même si $n > n'$.

Conclusion

$\mathcal{A}_{\mathcal{R}}$ et $\mathcal{R}$ sont donc en bisimulation.
Or $\textsc{Untime}(L(\mathcal{A}) = \pi_\Sigma(L(A \times \mathcal{R}))$, d'où $Untime(L(\mathcal{A})) = \pi_S igma(L(A \times \mathcal{A}_{\mathcal{R}}))$. Et comme $\mathcal{A}_{\mathcal{R}}$ est un automate à un compteur et A un automate fini, le vide et l'accessibilité sont décidables pour $\mathcal{A}$.

3.3 Exemples

Cas d'une mise à jour $x := x - 1$

Comme nous l'avons vu dans la partie 1.3, le fait d'ajouter une mise à jour de la forme $x := x - 1$ au modèle d'Alur et Dill le rend indécidable. Mais la construction employée par [BDFP04] utilise trois horloges. D'autre part, une seule horloge suffit à rendre toute partition compatible infinie.

Considérons le cas où X contient deux horloges, et où la mise à jour ne concerne que l'une d'entre elles. Nous obtenons alors le graphe des régions infinis représenté ci-dessous (Figure 3.1, 1 est pris comme constante maximale).

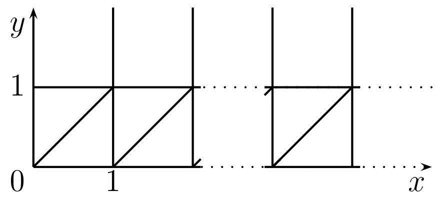


FIG. 3.1 – $\mathcal{C} = \{x = 0,\ x = 1,\ y = 0,\ y = 1\},\ Up = \{x := x - 1,\ y := 0\}$

Ce cas de figure rentre exactement dans la description de R donnée au début de la partie 3.2 : la partie finie correspond aux régions telles que $x \leq M$, où M est la constante maximale employée, le motif correspond aux $4 \times M + 2$ régions qui se trouvent entre $x = M$ et $x > M + 1$, et f est la translation de vecteur (1,0).

Théorème 3.3.1 *La classe* $\boldsymbol{AT}_2(\mathcal{C}_{rect}, \mathcal{U}_0 \cup \{x := x - 1\})$ *est décidable.*

Cas de $y = 2x$

Cette classe est un peu plus compliquée, mais elle est aussi présentée comme un exemple simple de cas où le graphe des régions est infini. La figure 3.2 permet d'avoir une idée de ce à quoi il ressemble avec la mise à jour $y := 0$.

Il est moins évident de voir le motif et la fonction f dans ce cas, mais R vérifie encore les hypothèses suffisantes pour construire un automate à compteur des régions.
Notons que le point $(0, 0)$ n'a pas de successeur immédiat, et justifie donc l'ajout des ε_ω-transitions.

Théorème 3.3.2 *La classe* $\boldsymbol{AT}(\{y \sim c,\ x \sim 0,\ y \sim 2x \mid c \in \mathbb{Q}\},\ \mathcal{U}_0)$ *est décidable.*

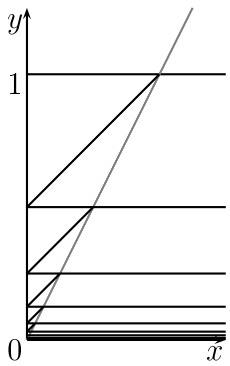


FIG. 3.2 – $\mathcal{C} = \{x = 0,\ y = 0,\ y = 1,\ y = 2x\},\ Up = \{x := 0\}$

Conclusion

L'objectif de ce stage était d'étudier certaines classes étendues d'automates temporisés, afin de montrer que le problème du vide y était décidable ou non. Nous nous sommes penchés en particulier sur le nombre d'horloges à partir duquel une classe est indécidable. Cette question est motivée par l'existence d'une classe ($\mathbf{AT}(\mathcal{C}^+,\mathcal{U}_0)$) pour laquelle le problème est décidable pour deux horloges et indécidable pour quatre. Le tableau ci-dessous présente un aperçu des résultats que nous avons obtenus en étudiant le modèle de base défini par Alur et Dill [AD90] auquel sont ajoutées des gardes et des mises à jour. Nous sommes partis des résultats connus grâce aux travaux de [BD00, BDFP04].

Gardes $\mathcal{C}_{rect} \cup \ldots$	Mises à jour $\mathcal{U}_0 \cup \ldots$	Nombre d'horloges pour une partition finie	classe décidable	classe indécidable
	$\exists! x \in X$, $x := x - 1$	0	≤ 2	≥ 3
$x - y \sim c$	$x := x - 1$	0	≤ 1	≥ 3
	$x := x + 1$	≥ 0	≥ 0	–
$x - y \sim c$	$x := x + 1$	1	≤ 1	≥ 4
$\exists k,\ y \sim kx$ pas de $x \sim c$ pour $c > 0$		1	2	?
$x + y \sim c$		2	≤ 2	≥ 4

Nous avons donc obtenu un résultat qui augmente le nombre des classes décidables connues d'automates temporisés. Il devrait être possible, en revenant sur la dernière hypothèse faite page 23, d'étendre le résultat obtenu aux mises à jour non déterministes [BDFP04].

D'autre part, la technique employée pour obtenir ce résultat (bisimulation entre graphe des régions infini et système différent d'un automate fini dont l'accessibilité est décidable) peut être poussée plus loin. Par exemple, les automates à une pile sont plus puissants (en terme de langage reconnus) que les automates à un compteur. On peut également considérer des automates à deux compteurs, mais dont un seul aurait un test à zéro [Sut00]. Ceci permettrait alors peut-être de trouver des limites précises entre sous-classes décidables et sous-classes indécidables.

Annexe A

Le programme HYTECH

```
-- hytech, test simple

-- Je cherche à déterminer quelle partie futurP de la face F (ancien futurP)
-- du cube rencontre en premier le plan P:x+y=1, et quelle partie rencontre
-- d'abord une contrainte C. J'obtient ainsi une nouvele contrainte newC, et je
-- peux me reposer la question au sujet de futurP : quelle partie de futurP
-- rencontre newC avant le plan et réciproquement.

-- pour ce fichier, C = z = 1 (pour itérer cette méthode, il faudra reécrire
-- ce programme en remplacant C par la nouvelle contrainte obtenue)

--AUTOMATES

var x,y,z,              --horloges utilisees
        h : clock;      --horloge empechant l'ecoulement du temps
        a : discrete;   --variable me permettant de "commander"

automaton A
synclabs:;
initially q0 & x=0 & y=0 & z=0 & h=0;

        --partie simulant l'avancee ds le tps
loc q0: while x=0 wait {}
        when x>=0 goto q1;
loc q1: while z<=1 & x+y<=1 wait {}
        when z=1 goto qA;
        when x+y=1 goto qB;
loc qA: while True wait{}
        when True goto qA;
loc qB: while True wait{}
        when True goto qB;

        --partie simulant la maj x:=0
```

```
loc qz : while h=0 wait {}
        when a=0 do {x'=0} goto q0;
        when a=1 goto qA;
        when a=2 goto qB;

end

--REGIONS

var     F, futurP, futurC,
        futurF,         --Post*(F)
        newC    : region;

F := loc[A]=q0 & x=0
        & y>=0 & z>=0
        & y<1 & z<1;


--Q1
prints "On part de :";
print omit A locations F;

futurF := reach forward from F endreach;

futurC :=  reach
               backward from (futurF & loc[A]=qA)
          endreach
        & loc[A]=q0;
futurP :=  reach
               backward from (futurF & loc[A]=qB)
          endreach
        & loc[A]=q0;

--Q2
prints "Les chemins rencontrant d'abord C partent de :";
print omit A locations futurC;
prints "Les chemins passant d'abord par z+y=1 partent de :";
print omit A locations futurP;

newC := hide a in reach backward from futurC endreach & loc[A]=qz endhide;
newC := hide a,h in
        post(newC & a=1)        --je me ramène à la fin de A
        & loc[A]=qA             --je veux être sure d'y être
        & h=0                   --je ne veux pas que le tps ait pu s'écouler
        endhide;
F := hide a in reach backward from futurP endreach & loc[A]=qz endhide;
```

```
F := hide a,h in post(F & a=2) & loc[A]=qB & h=0 endhide;

--Q3
prints "D'où la nouvelle région de départ :";
print omit A locations F;
prints "et la nouvelle contrainte :";
print omit A locations newC;
```

Bibliographie

[AD90] Rajeev Alur and David Dill. Automata for modeling real-time systems. In *Proc. 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990.

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

[BBR04] Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. Model-checking for weighted timed automata. In *Proc. Joint Conference on Formal Modelling and Analysis of Timed Systems and Formal Techniques in Real-Time and Fault Tolerant System (FORMATS+FTRTFT'04)*, Lecture Notes in Computer Science. Springer, 2004. To appear.

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

[BDFP04] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Updatable timed automata. *Theoretical Computer Science (TCS)*, 2004. To appear.

[Hen96] Thomas A. Henzinger. The theory of hybrid automata. In *Proc. 11th IEEE Annual Symposim on Logic in Computer Science (LICS'96)*, pages 278–292. IEEE Computer Society Press, 1996.

[HHWT95] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. A user guide to HYTECH. In *Proc. 1st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'95)*, volume 1019 of *Lecture Notes in Computer Science*, pages 41–71. Springer, 1995.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[Min67] Marvin Minsky. *Computation : Finite and Infinite Machines.* Prentice Hall International, 1967.

[Sut00] Grégoire Sutre. *Abstraction et accélération de systèmes finis.* PhD thesis, École Normale Supérieure de Cachan, Cachan, France, 2000.

[Tri03] Stavros Tripakis. Folk theorems on the determinization and minimization of timed automata. In *Proc. 1st International Workshop on Formal Modeling and Analysis of Timed Systems (FORMATS'03)*, Lecture Notes in Computer Science. Springer, 2003. To appear.