Mémoire de Master 2 : Accessibilité dans les automates temporisé à deux horloges

Guyslain Naves
Sous la direction de Patricia Bouyer et Nicolas Markey

3 septembre 2006

Résumé

Dans ce rapport, nous étudions le problème de l'accessibilité d'un état dans les automates temporisés à deux horloges. Il est connu que le problème de l'accessibilité est PSPACE-complet avec trois horloges, et qu'il est NLOGSPACE avec une seule horloge. Nous montrons qu'il est NP-complet pour une large sous-classe des automates temporisés à deux horloges. Nous donnons aussi une construction permettant d'éliminer les transitions qui ne remettent pas d'horloge à zéro tout en conservant les propriétés d'accessibilité et le nombre d'horloge, cette construction étant polynomiale en la taille de l'automate lorsque le nombre d'horloges est borné. Une piste possible pour montrer que le problème que nous considérons est NP-complet consiste à essayer de compresser les exécutions et de les représenter en espace polynomial. Pour cela, ou pourrait essayer d'utiliser des techniques d'accélération de cycles. Cependant ces techniques ne peuvent pas être utilisée en général. En effet, nous exhibons une classe d'automates temporisés à deux horloges sur lesquels l'accélération de cycles ne permet pas de résoudre le problème de l'accessibilité. Enfin nous montrons que le problème de synthèse de contrôleur sur les automates temporisés à deux horloges est EXPTIME-complet.

Table des matières

Chapitre 1

Introduction

Le model-checking a été introduit dans [12] pour permettre la vérification automatique par la modélisation des programmes ou des systèmes sous forme d'objets mathématiques simples, généralement des extensions d'automates, sur lesquels il est possible de vérifier des propriétés exprimées par des formules logiques. Cette approche a donné beaucoup de résultats et ses applications sont maintenant nombreuses. Le choix du modèle repose sur deux aspects en contradiction : le modèle doit être suffisament expressif pour capturer toute la complexité du programme à vérifier, et doit être suffisamment simple pour permettre de vérifier efficacement les propriétés attendues. Le choix du modèle est donc un compromis entre expressivité et complexité. Il est alors nécessaire d'avoir un grand choix de modèles et de bien les connaître afin de pouvoir choisir, le moment venu, le modèle le plus adapté à un système concret. On pourra notamment consulter [6] et [3] sur ce sujet.
Dans un premier temps, les modèles et les logiques étudiés, généralement des systèmes de transitions, permettaient de vérifier des propriétés sur les états atteints par le système et sur l'ordre de succession des événements. En particulier, le problème de la sécurité : « un état dangereux est-il inaccessible ? » et celui de vérifier qu'un événement donné déclenche systématiquement une réaction appropriée. Parmi ces modèles, on peut en particulier citer les automates finis, les automates à compteurs, les réseaux de Petri. Cependant, ces différents modèles ne permettent pas de prendre en compte le temps de manière quantitative : « combien de temps sépare deux événements donnés ? », « l'événement a déclenche-t-il l'événement b en moins de deux secondes ? ».... Il convient donc de disposer de modèles prenant en compte une datation précise des événements. Une alternative, dévéloppée par exemple dans [4], est alors d'utiliser un modèle en temps discret : les événements ont lieu à des dates entières. Le modèle reconnaît alors non plus une séquence d'événements, un mot, mais une séquence de paires d'une lettre et d'un entier naturel : la date de l'événement codé par cette lettre. Une autre alternative est d'utiliser un modéle en temps réel : les dates des événements appartiennent à l'ensemble des réels, ce qui permet de modéliser des systèmes physiques avec une meilleure précision.

L'un des modèles en temps réel qui ont rencontré le plus de succès est celui proposé dans [2] en 1994 : les automates temporisés. Ce succès s'explique en partie par le fait que le problème du vide du langage reconnu par un automate temporisé est PSPACE-complet, de même que l'accessibilité d'un état. D'un point de vue algorithmique, les problèmes PSPACE-durs sont néanmoins très difficiles à résoudre, et il serait souhaitable d'exhiber des modèles plus efficace. Pour cela, il peut être profitable de chercher à simplifier le modèle, par exemple en réduisant le nombre d'horloges. Dans [7], il est démontré que le problème de l'accessibilité reste PSPACE-dur avec seulement trois horloges, et on sait par ailleurs qu'il est NLOGSPACE pour une seule horloge [10]. Nous nous intéressons au problème ouvert de déterminer la complexité de l'accessibilité dans les automates temporisés à deux horloges. Dans [10], il est montré que ce problème est NP-dur, même lorsque l'automate temporisé est acyclique. Comme un automate temporisé à deux horloges est un cas particulier d'automate temporisé, il est aussi PSPACE.
Après avoir défini de manière précise les automates temporisés, nous montrerons la PSPACE-complétude de l'accessibilité sur plusieurs extensions des automates temporisés à deux horloges. On présentera une technique permettant de simplifier les automates temporisés à deux horloges en supprimant les transitions qui ne remettent pas d'horloge à zéro, ce qui nous permettra de démontrer la NP-complétude du problème de l'accesibilité sur une large sous-classe des automates temporisés à deux horloges. Enfin, nous étudierons la complexité d'un problème de contrôle sur les automates temporisés à deux horloges.

Chapitre 2

Automates temporisés

2.1 Définition

Afin de coder le comportement d'un système temps réel, on introduit le concept de mot temporisé, mot sur l'alphabet des couples formées d'une action du système et du temps auquel a lieu cette action. Ainsi, si Σ est l'alphabet des actions du système, $\Sigma \times \mathbb{R}_+$ est l'alphabet temporisé que nous considérons.

Définition 2.1.1
Un mot temporisé sur l'alphabet Σ est une suite (finie ou non) $(a_1, \tau_1), (a_2, \tau_2), \ldots$, telle que pour tout i inférieur ou égal à la longueur de cette suite, $a_i \in \Sigma$, $\tau_i \in \mathbb{R}_+$ et si $i > 1$, alors $\tau_i \geq \tau_{i-1}$. De plus, si cette suite est infinie, alors la suite $(\tau_i)_{i \in \mathbb{N}}$ diverge.

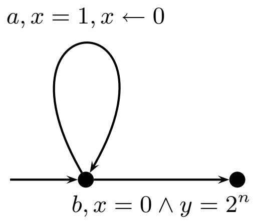
Exemple :
$(a,1)(b,2)(a,3)(b,4)...(a,2n)(b,2n+1)\ldots$ est un mot temporisé, mais $(a,\frac{1}{2})(a,\frac{1}{4})(a,\frac{1}{8})(a,\frac{1}{16})...(a,\frac{1}{2^n})\ldots$ ne l'est pas.

Les automates temporisés sont un modèle de calcul permettant de reconnaître certains langages temporisés. Le mot correspondant à une exécution de l'automate correspond à la séquence des lettres lues sur les transitions associés au temps écoulé lors du passage de la transition depuis le début de l'exécution. Soit X un ensemble de variables, on note $LC(X)$ l'ensemble des combinaisons booléennes d'inéquations linéaires sur X de la forme $x \sim c$ avec $x \in X$, $\sim \in \{=, \geq, >, \leq, <\}$ et $c \in \mathbb{N}$. Une valuation des horloges X est une élément de $\mathbb{R}_+^X$. Soit v une valuation de X, v satifait une inéquation $x \sim c$ si $v(x) \sim c$. Si v satisfait $P \in LC(X)$, on note $v \vDash P$, sinon on note $v \nvDash P$. Soit $R \subseteq X$, on note $v[R \rightarrow 0]$ la valuation qui sur R vaut 0 et sur $X - R$ prend la même valeur que v, et par $v + t$ pour $t \in \mathbb{R}_+$ la valuation qui à $x \in X$ associe $v(x) + t$. On note v_0 la valuation qui associe 0 à chaque horloge. Soit $x \in \mathbb{R}_+$, on note $\lfloor x \rfloor \in \mathbb{N}$ sa partie entière inférieur, $\lceil x \rceil$ sa partie entière supérieure, et $\{x\} \in [0, 1[$ sa partie fractionnaire. Enfin, on note $a = b[c]$ pour a est congru à b modulo c.

Définition 2.1.2
Un automate temporisé $\mathcal{A}$ est un 7-uplet $(Q, q_0, X, \Sigma, \delta, I, F)$, avec :
- Q un ensemble fini, l'ensemble des états de $\mathcal{A}$,
- $q_0 \in Q$ l'état initial,
- X un ensemble fini, l'ensemble des horloges,
- Σ un alphabet fini, les actions de $\mathcal{A}$,
- $\delta \subset Q \times \Sigma \times LC(X) \times \mathcal{P}(X) \times Q$ ensemble fini des transitions de l'automate,
- $I : Q \to LC(X)$ fonction qui a chaque état associe son invariant,
- $E : Q^* \cup Q^\omega \to \{\top, \bot\}$ une condition d'acceptation.

On utilisera la notation $q \xrightarrow{a,P,R} q'$ pour $(q, a, P, R, q') \in \delta$. Il s'agit d'une transition de l'état q vers l'état q' par l'action a, sous la contrainte (ou garde) P, et R est l'ensemble des horloges remises à 0.

Exemple :



Le dessin précédent représente un automate temporisé à deux états et deux horloges, x et y. L'état initial est marqué par une flèche entrante. On note $x \leftarrow 0$ pour dire que l'horloge x est remise à 0 par la transition.

Une configuration de l'automate temporisé $\mathcal{A} = (Q, q_0, X, \Sigma, \delta, I, E)$ est un couple (q, v) où q est un état de $\mathcal{A}$ et v est une valuation de l'ensemble des horloges dans $\mathbb{R}_+$, telle que v satisfait $I(q)$. On définit le système de transition temporisé associé à $\mathcal{A}$ par $T_{\mathcal{A}} = (\mathcal{C}_{\mathcal{A}}, \Sigma, s_0, \to)$ avec $\mathcal{C}_{\mathcal{A}}$ l'ensemble des configurations de $\mathcal{A}$, $s_0 = (q_0, v_0)$ la configuration initiale, et $\to$ est une relation définie par les règles suivantes :
- $(q, v) \xrightarrow{a} (q', v')$ si $q \xrightarrow{a,P,R} q'$ avec $v \vDash I(q)$, $v \vDash P$, $v' = v[R \to 0]$ et $v' \vDash I(q')$
- $(q, v) \xrightarrow{\epsilon(d)} (q', v')$ si $q = q'$, $v' = v + d$ et $\forall 0 \leq d' \leq d, v + d' \vDash I(q)$

Le premier type de transition est appelé transition d'action, et le second transition de délai.

Définition 2.1.3
Une exécution ρ de l'automate temporisé $\mathcal{A} = (Q, q_0, X, \Sigma, \delta, I, E)$ est une séquence de configurations de $\mathcal{A}$ $c_1, c_2, \ldots$ (finie ou non) telle que pour tout $i > 1$ inférieur à la longueur de ρ, $c_{i-1} \to c_i$ et telle que si ρ est infinie, pour $t_i = \sum_{\{j \leq i : c_{j-1} \xrightarrow{\epsilon(d_j)} c_j\}} d_j$, la suite $(t_i)_{i \in \mathbb{N}}$ diverge.
Soit $j_1 < j_2 < \ldots$ l'ensemble des indices tels que la transition $c_{j_i} \xrightarrow{a_i} c_{j_{i+1}}$ soit une transition d'action. ρ définit alors un unique mot temporisé $w(\rho) = (a_1, t_{j_1}), (a_2, t_{j_2}), \ldots$

Définition 2.1.4
Un mot temporisé w sur Σ est accepté par l'automate temporisé $\mathcal{A} = (Q, q_0, X, \Sigma, \delta, I, E)$ s'il existe une exécution ρ de $\mathcal{A}$ tel que $w = w(\rho)$ et $E(\pi(\rho)) = \top$, où π est la projection triviale des configurations sur les états. L'ensemble des mots acceptés par $\mathcal{A}$ est appelé langage reconnu par $\mathcal{A}$ et est noté $l(\mathcal{A})$.

En général, on utilise des conditions d'arrêt simples. Lorsqu'on veut reconnaître un langage de mots temporisés finis, on définit F un sous-ensemble de Q, l'ensemble des états finaux. On définit alors $E(\pi(\rho)) = \bot$ si ρ est infinie, et si ρ est fini, avec (q, v) la dernière configuration de ρ, $E(\pi(\rho)) = \top$ ssi $q \in F$.
Une autre condition classique est la condition de Büchi pour les mots temporisés infinis. Soit F un sous-ensemble de Q, on définit E par $E(\pi(\rho)) = \bot$ si ρ est fini, et sinon $E(\pi(\rho)) = \top$ ssi pour tout $n \in \mathbb{N}$, il existe $j \geq n$ et une valuation v telle que la i^{e}configuration de ρ est de la forme (q, v) avec $q \in F$.

Exemple :
Dans l'exemple précédent, si la condition d'acceptance est d'atteindre l'état de droite, l'automate reconnait le langage formé d'un unique mot temporisé, qui est $(a, 1)(a, 2) \ldots (a, 2^n - 1)(a, 2^n)(b, 2^n)$.

2.2 Automates des régions

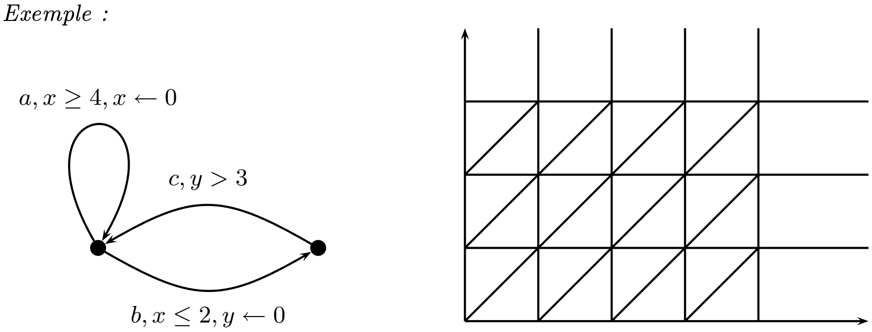
On présente ici une technique usuelle sur les automates temporisés. Cette technique consiste à partitionner l'ensemble des configurations en un nombre fini de classes d'équivalences. Deux configurations équivalentes partageront des propriétés semblables. On passe ainsi d'une quantité indénombrable de configurations à un nombre fini de classes d'équivalence, ce qui permet de décider plusieurs problèmes sur les automates temporisés.
On souhaite que deux configurations soient équivalentes si toute suite de transitions applicable à l'une l'est aussi à l'autre dans le même ordre (mais pas forcément au même moment). Une condition nécessaire pour cela est que les deux configurations soient dans le même état de l'automate. Pour toute horloge x, on note M_x le plus grand entier auquel la valeur de x est comparée dans une garde ou un invariant de l'automate.

Définition 2.2.1
Soient (q, u) et (p, v) deux configurations de l'automate temporisé $\mathcal{A} = (Q, q_0, X, \Sigma, \delta, I, E)$.
$(q, u) \sim (p, v)$ si les trois conditions suivantes sont vraies :
(i) $q = p$,
(ii) pour toute horloge $x \in X$, soit $u(x)$ et $v(x)$ sont strictement supérieurs à M_x, soit les deux conditions suivantes sont vérifiées :
- $\lfloor u(x) \rfloor = \lfloor v(x) \rfloor$,
- $u(x) \in \mathbb{N}$ ssi $v(x) \in \mathbb{N}$,

(iii) pour toute paire d'horloges x et y telle que $u(x) \leq M_x$ et $u(y) \leq M_y$, $\{u(x)\} < \{u(y)\}$ ssi $\{v(x)\} < \{v(y)\}$.

On note $[(q,v)]$ la classe déquivalence de (q,v) et on l'appelle région de (q,v). On note $\mathcal{R}_{\mathcal{A}}$ l'ensemble des régions de $\mathcal{A}$

Exemple :



Un automate temporisé à deux horloges et une représentation des régions de cet automate. Chaque point, segment, demi-droite et face correspond, associé à un état, à une région de l'automate. Son automate des régions a donc 208 états

En particulier, grâce à la condition (ii), si $(q,u) \sim (q,v)$, alors u et v satisfont les mêmes contraintes de $LC(X)$, donc les mêmes transitions pourront être prises dans l'automate. La condition (iii) permet détablir le résultat suivant :

Proposition 2.2.2
Soit (q,v_1) une configuration telle qu'il existe $t \in \mathbb{R}_+$ pour lequel $(q,v_1) \nsim (q,v_1+t)$. Soit $t_1 \in \mathbb{R}_+$ tel que $(q,v) \nsim (q,v+t_1)$ et pour tout $t' \in [0,t]$, $(q,v_1) \sim (q,v_1+t')$ ou $(q,v_1+t_1) \sim (q,v_1+t')$. Alors pour toute configuration (q,v_2) équivalente à (q,v_1), il existe t_2 tel que $(q,v_2) \nsim (q,v_2+t_2)$ et pour tout $t' \in [0,t]$, $(q,v_2) \sim (q,v_2+t')$ ou $(q,v_2+t_2) \sim (q,v_2+t')$. De plus, dans ce cas $(q,v_1+t_1) \sim (q,v_2+t_2)$. On définit alors $\mathrm{succ}([(q,v_1)]) = [(q,v_1+t_1)]$

Proposition 2.2.3
Soient $(q,u),(q,v)$ deux configurations équivalentes. Soit $R \subseteq X$ un sous-ensemble d'horloge. Alors, $(q,u[R \to 0]) \sim (q,v[R \to 0])$. On définit alors $[(q,v)][R \to 0] = [(q,v[R \to 0])]$.

Grâce à ces deux propositions, on sait maintenant que depuis deux configurations équivalentes, en effectuant la même suite de transitions, en choisissant de bons délais, on arrive dans deux configurations appartenant à une même région. On définit donc l'automate des régions par l'automate dont les sommets sont les régions de $\mathcal{A}$, et les arcs relient une région à une région directement accessible par une action où un délai.

Définition 2.2.4
Soit $\mathcal{A} = (Q, q_0, X, \Sigma, \delta, I, E)$ un automate temporisé. On définit l'automate des régions de $\mathcal{A}$ par l'automate $(\mathcal{R}_{\mathcal{A}}, r_0, \Sigma, \to, E)$ suivant :

- $\mathcal{R}_{\mathcal{A}}$ ensemble des états
- $[(q,u)] \xrightarrow{\epsilon} [(q',v)]$ si $[(q',v)] = \mathrm{succ}([(q,u)])$,
- $[(q,u)] \xrightarrow{a} [(q',v)]$ s'il existe une transition $q \xrightarrow{a,P,R} q'$ telle que $u \models P$, $u \models I(q)$ et $v = u[R \to 0]$,
- $r_0 = [q_o, v_0]$ état initial,
- E est la condition d'arrêt que l'on étend de manière triviale.

Proposition 2.2.5
Le nombre de régions d'un automate temporisé $\mathcal{A} = (Q, q_0, X, \Sigma, \delta, I, E)$ est borné par $|Q| \cdot |X|! \cdot M^{|X|}$ où $M = 4 \max_{x \in X} M_x + 1$.

Cette construction trouve son intérêt lorsque l'on veut résoudre le problème du vide du langage reconnu par un automate temporisé donné avec une condition d'arrêt par états finaux. Pour ce faire, il suffit de construire l'automate des régions, qui lui est un automate non-temporisé, et d'appliquer les algorithmes classiques de théorie des automates. Si un mot w est reconnu dans l'automate des régions, alors il existe un mot temporisé reconnu par $\mathcal{A}$ tel que sa projection sur Σ est w. On peut aussi vérifier qu'un état de l'automate temporisé est accessible en trouvant une région sur cet état qui est accessible dans l'automate des régions. De manière plus générale, l'automate des régions possède de nombreuses applications, dont le model checking des automates temporisés pour diverses logiques temporelles. On notera que la taille de l'automate des régions est exponentielle en général par rapport à la taille de l'automate temporisé, à cause du nombre de régions.

2.3 Accessibilité dans un automate temporisé

Le problème de l'accessibilité est un problème fondamental lorsque l'on fait du model checking, car c'est elle qui va permettre de savoir si le système à vérifier peut atteindre un état interdit, comme un *deadlock* par exemple. Grâce à l'automate des régions, on peut décider en PSPACE si un état de l'automate temporisé est accessible ou non depuis une configuration donnée (généralement la configuration initiale). De manière plus précise, on a le résultat :

Proposition 2.3.1
Soit $\mathcal{A}$ un automate temporisé, q un état et c une configuration de $\mathcal{A}$. Décider si q est accessible depuis c dans $\mathcal{A}$ est PSPACE-complet, même lorsque A n'a que trois horloges.

On montre maintenant que la complexité ne vient pas que du nombre détats, mais peut aussi venir du nombre d'horloge. Cette construction originale utilise des idées déjà employées dans [1].

Proposition 2.3.2
Soit $\mathcal{A}$ un automate temporisé, q un état et c une configuration de $\mathcal{A}$. Décider si q est accessible depuis c dans $\mathcal{A}$ est PSPACE-complet, même lorsque A n'a que deux états.

Preuve
Ce problème est équivalent au problème d'accessibilité dans le graphe des régions, l'accessibilité dans un graphe est un problème NLOGSPACE-complet, et le graphe des régions est de taille exponentielle, et puisque NPSPACE = PSPACE, la complexité est donc bien PSPACE, en construisant le graphe des régions à la volée.
On prouve la PSPACE-difficulté en réduisant à partir de LBTM (Linear-space Bounded Turing Machine), autrement dit, l'accessibilité dans une machine de Turing en espace linéairement borné (son espace de travail est celui sur lequel est écrit l'entrée de la machine). Soient donc M une telle machine et un mot $w \in \Sigma^*$, avec $\Sigma = \{a, b\}$. On note m le nombre d'états de M et n la taille de w et donc aussi du ruban de M. On indice les états de M de 1 à m de telle sorte que le m^{e} état soit l'état final et le premier soit l'état initial.
On construit maintenant un automate $\mathcal{A}$ à deux états, u_1 l'état initial et u_2 l'état final. $\mathcal{A}$ possède 4 horloges par case du ruban (x_i, y_i, z_i, p_i), 1 horloge par état de M (s_i), et deux autres horloges, t et t'. x_i, y_i et z_i codent la valeur de la case i du ruban, et p_i est un drapeau permettant de savoir si la tête de lecture est sur la case i. s_i est un drapeau permettant de savoir si l'état actuel de la machine est l'état i. Plus précisément :

- a est stocké sur la case i du ruban se code par $x_i = y_i$ et $z_i \leq 1$
- b est stocké sur la case i du ruban se code par $x_i \neq y_i$ et $z_i \leq 1$ ou par $z_i > 1$
- la machine est à l'état i se code par $s_i \leq 1$ et $\forall j \neq i, s_j > 1$
- la tête de lecture est à la position i se code par $p_i \leq 1$ et $\forall j \neq i, p_j > 1$

Soit $q_i, l \rightarrow q_j, l', \delta$ une transition de la machine de Turing, avec $i, j \in [\![1, m]\!]$, $l, l' \in \{a, b\}$ et $\delta \in \{-1, 1\}$. Soit $k \in [\![1, n]\!]$ telle que $k \neq 1$ si $\delta = -1$ et $k \neq n$ si $\delta = 1$. On définit les contraintes linéaires suivantes :

- POSITION$(k) \hat{=} (p_k = 1) \wedge \bigwedge_{l \in [\![1,n]\!]-\{k\}} p_l > 1$, permettant de vérfier que la tête de lecture est à la place k,
- ETAT$(i) \hat{=} (s_i = 1) \wedge \bigwedge_{l \in [\![1,m]\!]-\{i\}} s_l > 1$, permettant de vérifier que l'état actuel de la machine de Turing est bien l'état i,
- CODE$\hat{=} \bigwedge_{l \in [\![1,n]\!]} (x_l \leq 2 \wedge y_l \leq 2 \wedge z_l = 1)$, permettant de vérifier que la valeur de chaque case du ruban est bien définie,
- LECTURE$(a, k) \hat{=} (x_k = 1 \wedge y_k = 1) \vee (x_k = 2 \wedge y_k = 2)$, permettant de vérifier que la case k du ruban code la valeur a,
- LECTURE$(b, k) \hat{=} (x_k = 1 \wedge y_k = 2) \vee (x_k = 2 \wedge y_k = 1)$, permettant de vérifier que la case k du ruban code la valeur b.

Enfin, suivant les valeurs de l et l', on définit les ensembles d'horloges suivants :

- ECRITURE$(k, _, a) \hat{=} \{x_k, y_k\} \cup \{z_l : l \in [\![1, n]\!]\}$ permet d'écrire a sur la case k,
- ECRITURE$(k, a, b) \hat{=} \{x_k, y_k\} \cup \{z_l : l \in [\![1, n]\!] - \{k\}\}$ permet d'écrire un b à la place d'un a sur la case k du ruban,
- ECRITURE$(k, b, b) \hat{=} \{z_l : l \in [\![1, n]\!]\}$ permet de laisser un b sur la case k du ruban.

On peut maintenant définir une transition de notre automate par :

$$u_1 \xrightarrow{\epsilon,\phi,\mathrm{ECRITURE}(k,l,l')\cup\{s_j,t\}} u_1$$

$$\phi \hat{=} \mathrm{POSITION}(k) \wedge \mathrm{ETAT}(i) \wedge \mathrm{CODE} \wedge \mathrm{LECTURE}(l,k) \wedge t = 1 \wedge t' > 1$$

En plus de toutes les transitions codant des transitions de la machine de Turing, on a besoin de quelques transitions supplémentaires :

- Puisque les x_i sont calculés à un modulo 2 près, on introduit la transition suivante pour tout $i \in [\![1,n]\!]$:

$$u_1 \xrightarrow{\epsilon,t=0\wedge x_i=2,x_i\leftarrow 0} u_2$$

- De même pour y_i pour tout $i \in [\![1,n]\!]$:

$$u_1 \xrightarrow{\epsilon,t=0\wedge y_i=2,y_i\leftarrow 0} u_2$$

- Après l'écriture d'un b à la place d'un a en place i, les deux horloges sont remises à zéros. Pour rétablir la bonne valeur dans la case du ruban, il faut donc en remettre une à 0 après une unité de temps. Cette opération a lieu lorsque $z_i > 1$ et est nécessaire pour faire progresser le calcul, puisque $z_i \leq 1$ est requis pour passer une transition correspondant à une étape de calcul de la machine de Turing. On a donc besoin des transitions suivantes pour tout $i \in [\![1,n]\!]$:

$$u_1 \xrightarrow{\epsilon,t=1\wedge z_i>1,\{y_i,z_i\}} u_2$$

Enfin, il nous faut une transition permettant d'initialiser le calcul, et une autre permettant de vérifier que l'état final est atteint. L'initialisation se fait lorsque $t' = 1$ (remarquez que t' n'est jamais remise à 0). À ce moment, toutes les horloges valent 1. On remet donc à 0 toutes les horloges y_i telle que $i \in [\![1,n]\!]$ et $w_i = b$, ainsi que t, s_1, p_1 et tous les z_j, sous la seule condition $t' = 1$. La transition finale s'écrit $u_1 \xrightarrow{\epsilon,t'>1\wedge s_q=0,\phi} u_2$. La preuve de la réduction se fait sans difficulté, et la réduction se fait bien en temps polynomial. ■

D'autre part, avec une seule horloge, on peut résoudre le problème de l'accessibilité de manière efficace :

Proposition 2.3.3
Le problème de l'accessibilité dans un automate temporisé à une seule horloge est NLOGSPACE-complet.

La preuve se fait en considérant simplement des « régions élargies » qui sont en nombre polynomial. Enfin, on mentionne cette conséquence de la construction de l'automate des régions. Puisque le nombre des régions dépend seulement exponentiellement du nombre d'horloge et polynomialement de la valeur des constantes apparaissant dans l'automate temporisé, on a :

Proposition 2.3.4
Pour tout $k \in \mathbb{N}$, le problème de décider l'accessibilité d'un état dans un automate à k horloges est pseudopolynomial.

Chapitre 3

Accessibilité dans les automates temporisés à deux horloges

Dans le chapitre précédent, le cas de l'accessibilité dans les automates à deux horloges n'est pas étudié. De fait, la complexité de ce problème n'est pas précisement connue. Toujours grâce à la construction de l'automate des régions, on sait que ce problème appartient à PSPACE. Mais il n'y a pas de preuve de PSPACE-difficulté, quel que soit le nombre d'etats. Par la suite, on étudiera seulement des problèmes d'accessibilité, on omettra donc de mentionner les lettres associées aux transitions des automates temporisés, car elle n'ont pas d'importance pour ce problème.
On note que pour les problèmes d'accessibilité, on peut supprimer les invariants en les rajoutant aux transitions sortantes des états. On considèrera donc maintenant uniquement des automates sans invariants.

3.1 NP-difficulté

La NP-difficulté du problème a d'abord été montré par [10] sous la forme suivante, par une réduction au problème du sac-à-dos :

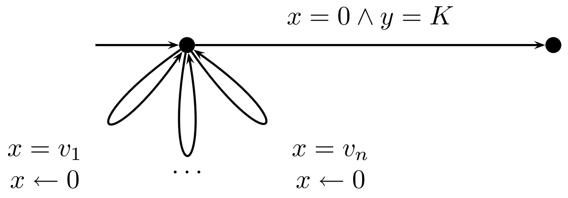
Proposition 3.1.1
L'accessibilité d'un état dans un automate à deux horloges est un problème NP-difficile, même lorsque l'automate est acyclique.

On propose une variante originale de ce résultat :

Proposition 3.1.2
L'accessibilité d'un état dans un automate à deux horloges est un problème NP-difficile, même lorsque l'automate n'a que deux états.

Preuve
On réduit à partir de INTEGER KNAPSACK [8], qui est le problème suivant : Étant donné n entiers $v_1, \dots, v_n$ et un objectif entier K, existe-t-il $a_1, \dots, a_n \in \mathbb{N}$ tels que $\sum_{i \in [\![1,n]\!]} a_i v_i = K$? Ce problème est NP-complet. On propose l'automate suivant :
On affirme que l'accessibilité de l'état de droite dans cette automate est équiva-

FIG. 3.1 – Réduction d'INTEGER KNAPSACK



lente à l'existence d'une solution pour le problème INTEGER KNAPSACK. En effet, il faut pour atteindre l'état de droite que l'horloge x soit égale à zéro, donc que l'exécution passe un nombre entier de fois par chaque transition bouclant sur l'état initial. La valeur de y à la fin d'une telle transition correspond donc au poids d'une certaine combinaison entière de valeurs parmi $v_1, \dots, v_n$. Donc si l'état de droite est atteint, le problème de INTEGER KNAPSACK a bien une solution. Réciproquement, on construit facilement une exécution de l'automate atteignant l'etat de droite si on possède une combinaison entière permettant d'atteindre l'objectif K, en passant par chaque transition testant $x = v_i$ un nombre a_i de fois. ■

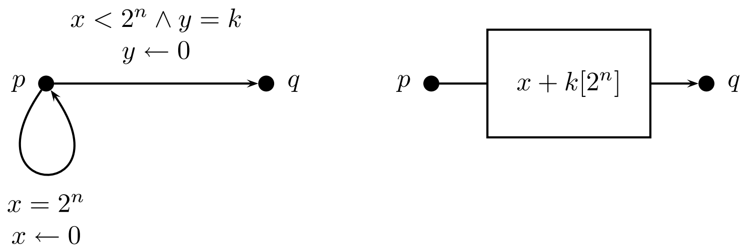
3.2 Gadgets

Dans cette section, on présente plusieurs gadgets, parties d'automates temporisés, qui seront utilisés dans des réductions pour prouver la difficulté de problèmes d'accessibilité. Toutes les horloges seront considérées avoir des valeurs bornées par 2^n pour un n donné, la taille de ces gadgets étant toujours polynomiale en n. Initialement, x possède une valeur entière les autres horloges valent 0, et doivent valoir 0 à la fin du calcul.

3.2.1 Additionneur

On se place dans le contexte suivant. On a deux horloges, disons x et y. L'horloge x code une valeur entière bornée, $x \in [\![0, 2^n - 1]\!]$. L'horloge y est une horloge dont la valeur n'a *a priori* pas d'importance, on peut donc en disposer comme bon nous semble et la rendre égale à 0. En pratique, cette horloge servira d'horloge de calcul. On souhaite réaliser quelques opérations simples, par exemple construire un automate temporisé contenant deux états p et q tel que pour toute valeur de x, il existe une unique configuration (q, x_0, y_0) atteignable depuis $(p, x, 0)$, et de plus elle vérifie $y_0 = 0$ et $x_0 = x + k[2^n]$. L'automate sur la gauche représente le gadget qui permet d'additionner une

FIG. 3.2 – additionneur

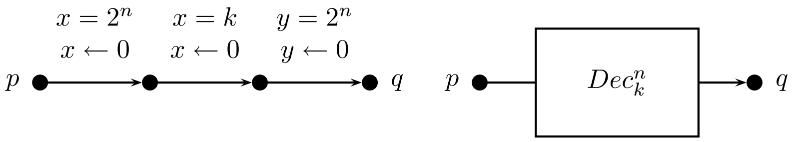


constante k à l'horloge x sous condition que $y = 0$. On le représentera dorénavant par la boîte indiquée à droite.

3.2.2 Soustracteur

On note que soustraire k revient à ajouter $2^n - k$, ce qui nous définit aussi un gadget permettant de soustraire une constante. Nous allons cependant avoir besoin d'un autre gadget pour faire la soustraction, de telle sorte que le gadget fonctionne précisément en temps 2^n pour soustraire k lorsque $x \in [\![k, 2^n - 1]\!]$.

FIG. 3.3 – Soustracteur

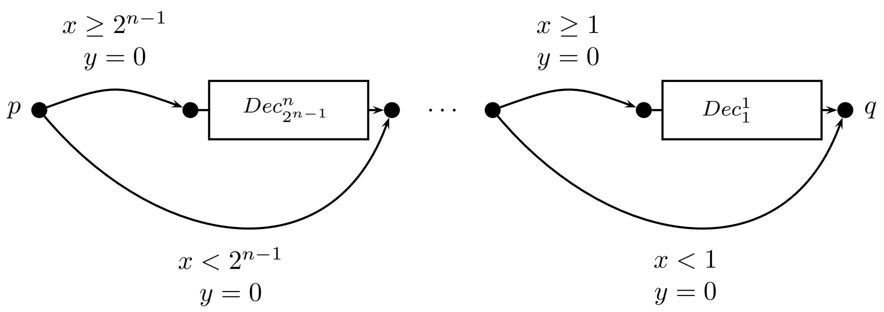


Les gadgets suivants sont des gadgets utilisant trois horloges. On montrera par la suite que pour un automate temporisé à deux horloges utilisant un de ces gadgets, le problème de l'accessibilité est PSPACE-dur.

3.2.3 Multiplieur

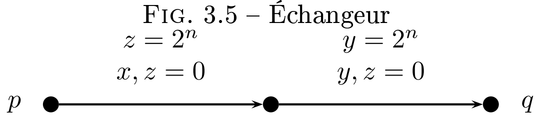
Le gadget suivant permet de passer de la configuration (p, x_0, y_0, z_0) à la configuration (q, x_1, y_1, z_1) avec $y_0 = z_0 = y_1 = x_1$ et $z_1 = 2x_0$ sous la condition $x_0 \in [\![0, 2^n - 1]\!]$.

Fig. 3.4 – Multiplieur



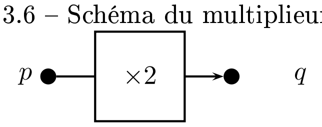
Si $x = \sum_{i \in [\![0,n-1]\!]} x_i 2^i$ avec pour tout i $x_i \in \{0, 1\}$, alors le temps passé dans ce gadget est exactement $\sum_{i \in [\![0,n-1]\!]} x_i 2^{i+1} = 2x$. Ce gadget double la valeur de x lorsque x est un entier inférieur à 2^n. Cependant, le double de la valeur de x est stocké dans l'horloge z et non dans x. On présente donc un gadget qui copie la valeur de z dans x et remet à zéro y et z.

Fig. 3.5 – Échangeur



On notera le multiplieur, qui est la concaténation des deux précédents gadgets, de la manière suivante :

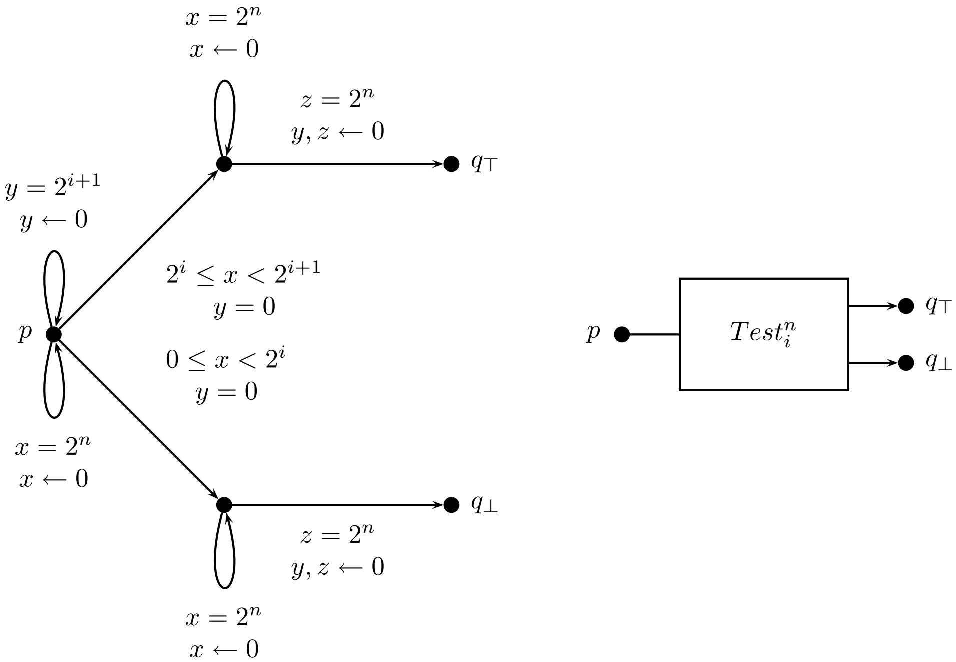
Fig. 3.6 – Schéma du multiplieur



3.2.4 Test d'un bit

On construit maintenant un gadget qui va permettre de distinguer si le i^e bit de l'écriture en binaire de la valeur de l'horloge x est zéro ou 1. Lorsque ce bit sera zéro l'état $q_\perp$ sera seul accessible, et lorsqu'il sera égal à 1, c'est l'état $q_\top$ qui sera accessible. Initialement, les valeurs de y et z sont nulles. On demande comme condition supplémentaire que la valeur de x ne soit jamais changée par l'utilisation de ce gadget.

FIG. 3.7 – Test d'un bit



Le principe est d'ajouter 2^{i+1} à x jusqu'à dépasser 2^n. Grâce aux retenues, on a pu ainsi "chasser" tous les bits de valeur 1 en positions supérieures à i, c'est-à-dire calculer la valeur de x modulo 2^{i+1}. On peut donc facilement tester la valeur du i^e bit. On retrouve la valeur initiale de x en utilisant l'horloge z.

3.3 Automates à deux horloges avec test

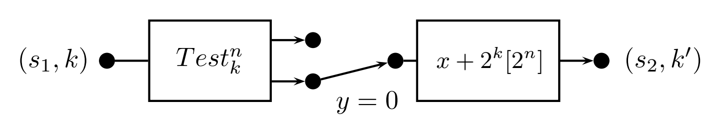
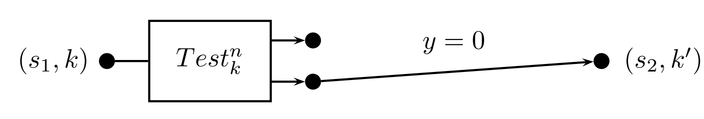
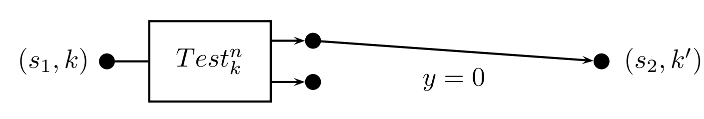
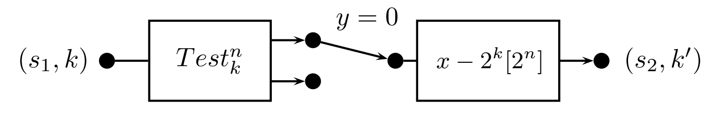
On montre maintenant que l'ajout de ce dernier gadget, permettant de tester la valeur du i^e bit d'une horloge, au modèle des automates temporisés à deux horloges rend le probléme de l'accessibilité PSPACE-complet.

Proposition 3.3.1
L'accessibilité dans les automates temporisés à deux horloges avec test des bits

de l'horloge x est un problème PSPACE-complet.

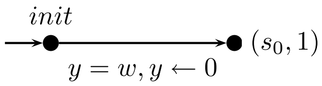
Preuve
À nouveau on réduit à partir de LBTM. Soit M une machine de Turing en place à p états, et soit w un mot sur l'alphabet $\{a, b\}$ de taille n. On construit un automate temporisé à deux horloges, x et y, dont les états sont les couples (s, i) avec s état de M et $i \in [\![0, n-1]\!]$, auxquels on rajoute l'état *init*. x code la valeur du ruban comme un entier borné par 2^n, y sert d'horloge de calcul. Pour toute transition $s_1, l \rightarrow s_2, l', \delta$ avec s_1, s_2 états de M, $l, l' \in \{a, b\}$ et $\delta \in \{-1, 1\}$, pour tout $k \in [\![0, n-1]\!]$ tel que $k' = k + \delta \in [\![0, n-1]\!]$, on ajoute le gadget correspondant du tableau ci-dessous :

FIG. 3.8 – Transitions de l'automate construit

l	l'	gadget
a	b	
a	a	
b	b	
b	a	

Soit s_0 l'état initial de M. On ajoute la transition suivante (on confond w et l'entier représenté en binaire par w) :

FIG. 3.9 – Initialisation



Les états finaux de cet automate sont les états (s_f, k) pour tout s_f final et $k \in [\![0, n-1]\!]$. Alors la machine M termine sur l'entrée w si et seulement si l'automate temporisé ci-dessus possède un état final accessible. ■

3.4 Autres cas PSPACE-complets

On vient de montrer que l'ajout des tests aux automates temporisés rend le problème de l'accessibilité PSPACE-complet. On montre maintenantdeux résultats similaires, où l'ajout d'une opération assez simple permet de démontrer la PSPACE-complétude.

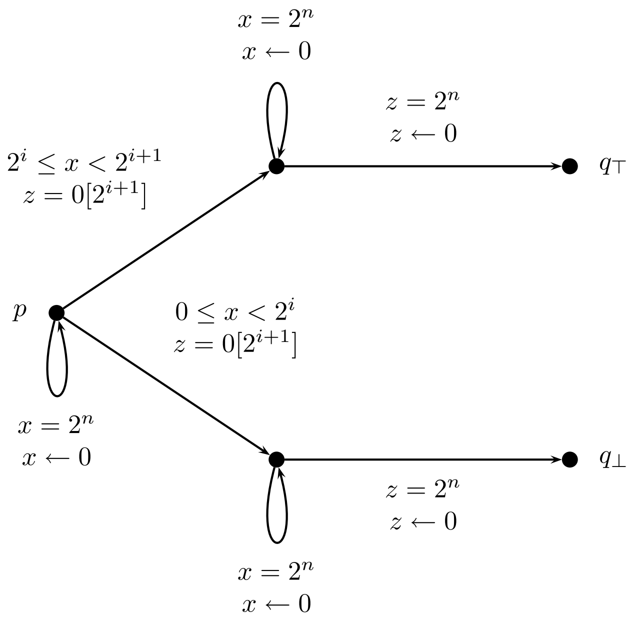
3.4.1 Gardes modulos

On souhaite ajouter un nouveau type de garde, les gardes modulos, pour les transitions des automates temporisés. On permet donc maintenant que les gardes soient des combinaisons booléennes de contraintes de la forme $x \sim c$ ou $x = c[2^k]$ avec x une horloge quelconque, $c \in \mathbb{N}$, $\sim \in \{=, <, >, \leq, \geq\}$ et $k \in \mathbb{N}$. La construction de l'automate des régions est toujours valide sous cette hypothèse. D'autre part, il est connu que l'on peut supprimer les modulos d'un automate temporisé, par une construction qui augmente le nombre d'horloges [5].

Proposition 3.4.1
L'accessibilité dans les automates temporisés à deux horloges avec modulo est PSPACE-complet.

Preuve
C'est PSPACE par la construction de l'automate des régions. On montre la difficulté en implémentant le gadget du test du i^{e} bit.

FIG. 3.10 – Test avec modulo



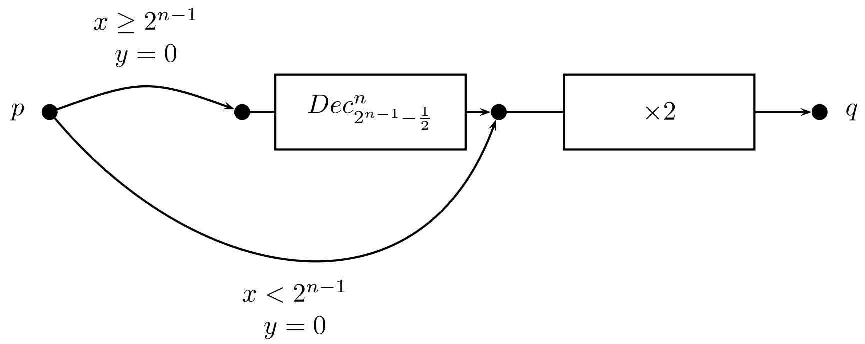
■

3.4.2 Multiplication par 2

Proposition 3.4.2
L'accessibilité dans les automates temporisés à deux horloges avec multiplication par 2 est PSPACE-complet.

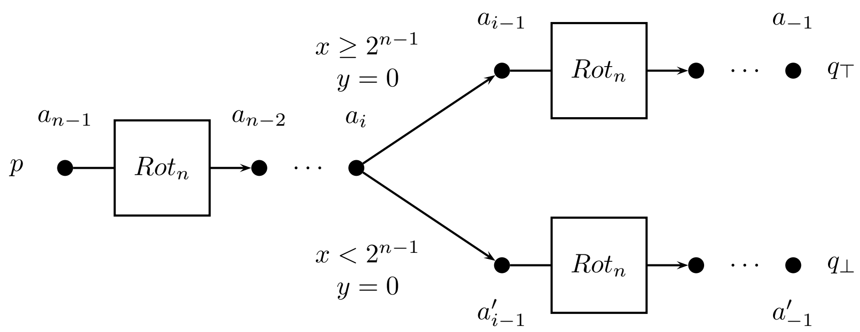
Preuve
Là encore, la multiplication par 2 permet encore la validité de la construction de l'automate des régions, donc le problème est bien PSPACE. On montre que c'est PSPACE-dur en implémentant le test du i^{e} bit. On aura besoin du gadget suivant pour ensuite coder le test du i^{e} bit, qu'on appellera Rot_n, et qui effectue une rotation des bits de l'écriture binaire de x vers la gauche :

FIG. 3.11 – Rotation des bits



Le test consiste donc à faire une rotation des bits, jusqu'à avoir le bit à tester comme bit de poids fort, le tester puis finir la rotation pour revenir à la valeur de départ :

FIG. 3.12 – Test avec multiplieur



■

Ceci prouve aussi que le seul gadget de rotation suffit à prouver que l'accessibilité est un problème PSPACE-complet. Ce n'est pas surprenant, car la multiplication par 2 sur n bits et la rotation sur $n+1$ bits ont le même effet.
Pour conclure ce chapitre, même si on ignore toujours la complexité du problème de l'accessibilité pour deux horloges, on remarque que des ajouts relativement faibles (puisqu'on peut les faire avec trois horloges), permettent de démontrer la PSPACE-complétude. Pourtant, nous n'avons pas plus trouvé de manière d'implémenter un de ces gadgets avec deux horloges que de preuves de ne serait-ce que de la coNP-complétude de l'accessibilité.

Chapitre 4

Élimination des transitions sans remise à zéro

Dans ce chapitre, nous allons montrer qu'il est possible de supprimer les transitions qui ne remettent aucune horloge à zéro sans augmenter le nombre d'horloges, par une transformation dont la complexité est exponentielle en le nombre d'horloges, mais polynomiale par rapport à tous les autres paramètres si on permet les gardes diagonales, c'est-à-dire les gardes de la forme $x - y \sim c$. Les gardes diagonales peuvent être éliminées et remplacées par des gardes non-diagonales, mais cette opération est coûteuse puisque le nombre d'états explose exponentiellement. La transformation double le nombre d'états par gardes diagonales à retirer, mais il existe un transformation plus astucieuse qui multiplie le nombre d'états par $\mathcal{O}(n^{|X|^2})$, avec n la taille de l'automate et $|X|$ le nombre d'horloges. L'élimination des transitions sans remise à zéros préservera les propriétés d'accessibilité, mais pas les propriétés de langage, si bien qu'on continuera d'omettre les références aux lettres reconnues par les transitions des automates temporisés.

4.1 Zones d'un automate temporisé

Soit $C(X) \subseteq LC(X)$ un ensemble de gardes linéaires. On demande en plus que chaque garde de $C(X)$ soit telle que l'ensemble de ces modèles soit un polyèdre P de $\mathbb{R}_+^X$. On sait qu'il est toujours possible de prendre cette restriction pour l'ensemble des gardes d'un automate temporisé sans augmenter significativement la taille de l'automate (on peut transformer l'automate en temps polynomial). Soit $cyl(P)$ le cylindre engendré par P selon la direction $\mathbb{1}$, où $\mathbb{1}$ est le vecteur dont toutes les composantes sont égales à 1. Autrement dit, si P est défini par le système d'inéquation $Ax \prec b$ avec $\prec$ un vecteur dans $\{<, \leq\}$ et $A \in \{-1, 0, 1\}^{m \times n}$ une matrice telle que chaque ligne ne comporte qu'un coefficient non-nul, ce coefficient étant 1 ou -1 :

$$cyl(P) = \{x \in \mathbb{R}_+^X \mid \exists t \in \mathbb{R} \text{ t.q. } A(x + t \cdot \mathbb{1}) \prec b\}$$

On définit la matrice $A' \in \{-1, 0, 1\}^{m \times (n+1)}$ de la manière suivante : pour tout $i \in [\![1, m]\!]$, pour tout $j \in [\![1, n]\!]$, $A'_{i,j} = A_{i,j}$ et $A'_{i,n+1} = \sum_{k \in [\![1,n]\!]} A_{i,k}$ (A' est la concaténation de A avec $A \cdot \mathbb{1}$). Soit $x'(t) \in \mathbb{R}_+^{n+1}$ telle que pour tout $i \in [\![1, n]\!]$, $x'_i(t) = x_i$ et $x'_{n+1}(t) = t$. Alors :

$$cyl(P) = \{x \in \mathbb{R}_+^X \mid \exists t \in \mathbb{R} \text{ t.q. } A'x'(t) \prec b\}$$

On cherche maintenant à supprimer les occurences de t dans le système d'inéquations précédent. Pour cela, on applique la procédure suivante : tant qu'il existe deux inéquations faisant apparaître t, disons $a_1x + a_1t \prec_1 b_1$ et $a_2y + a_2t \prec_2 b_2$, on remplace la première par la somme (si $a_1 \neq a_2$) ou la différence (sinon) des deux inéquations. Si une seule des deux inéquations est stricte, c'est celle-là que l'on remplace. Dans ce cas, le nouveau système obtenu est équivalent à l'ancien. En répétant l'opération, on obtient un système tel qu'une seule inéquation fait référence à t : $A''x'(t) \prec b''$. Alors, on peut supprimer cette équation et la quantification existentielle, puisque $\exists t \in \mathbb{R}$ s.t. $a_1x + a_1t \prec b$ est toujours vrai lorsque $a_1 \neq 0$, donc :

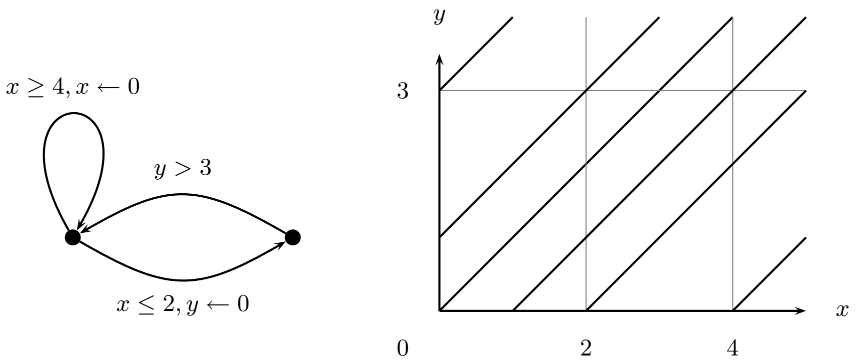
$$cyl(P) = \{x \in \mathbb{R}_+ \mid A_d x \prec b_d\}$$

avec A_d une matrice de contraintes diagonales, c'est-à-dire une matrice à coefficient dans $\{0, 1\}$, telle que chaque ligne co ;porte exactement un coefficient de valeur 1 et un de valeur 0. Toute inéquation du système $A_d x \prec b_d$ s'écrit par construction $x - y \prec a - b$, avec a et b tels que x est comparé à a dans la garde engendrant P, et y à b. Le nombre d'inéquations possibles est donc borné par le produit des nombres de constantes auxquelles x et y sont comparées à un facteur multiplicatif constant près, pour l'ensemble des gardes d'un automate temporisé. Pour chaque couple d'horloges x et y, il existe un nombre polynomial de contraintes diagonales sur ces deux horloges, qui partitionnent l'ensemble des valuations d'horloges en un nombre polynomial de classe pour la relation d'équivalence définie par $v \sim_{x,y} v'$ ssi v et v' vérifient les mêmes contraintes diagonales sur x et y. Si on prend maintenant l'intersection $\sim_d$ de ces classes d'équivalence pour chaque couple d'horloges, cela nous donne une partition de l'espace de valuations d'horloges en cylindres de taille $\mathcal{O}((2c^2 + 1)^{|X|^2})$, avec c le nombre de constantes auxquelles une horloge est comparée dans l'ensemble des gardes. En effet, pour chaque couple d'horloges, il y a au plus c^2 valeurs $a - b$ possibles, avec x et y comparés respectivement à a et b dans les gardes de l'automate. Pour chaque couple d'horloge, la relation $\sim_{x,y}$ contient donc au plus $2c^2 + 1$ classes d'équivalence ($x - y = a - b$ ou $a_1 - b_1 < x - y < a_2 - b_2$ pour des valeurs $a_1 - b_1$ et $a_2 - b_2$ consécutives). Le produit de cette borne pour chaque couple d'horloge donne une borne du nombre de classes d'équivalence pour $\sim_d$. Cette borne est exponentielle en le nombre d'horloges, mais polynomiale dès que le nombre d'horloges est borné, en particulier si on travaille sur les automates à deux horloges.

Définition 4.1.1
On appelle zone[1] d'un automate temporisé une classe d'équivalence pour la relation $\sim_d$, lorsque $C(X)$ est l'ensemble des formules logiques sur les inéquations de la forme $x \sim c$ telles que x est comparée à c dans une garde de $\mathcal{A}$ ou $c = 0$, et $\sim \in \{=, <, \leq, >, \geq\}$. La zone d'une valuation d'horloge v est noté $Z(v)$. On note $\mathcal{Z}_{\mathcal{A}}$ l'ensemble des zones de $\mathcal{A}$.

Définition 4.1.2
On appelle base de la zone Z sur l'horloge x l'intersection de Z avec l'hyperplan d'équation $x = 0$.

Exemple :



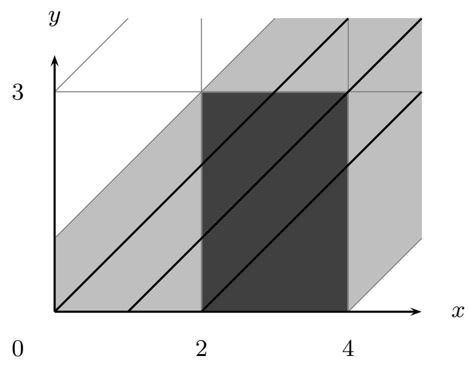
Un automate temporisé à deux horloges et une représentation des zones de cet automate. Il y a 13 zones, dont six demi-droites obliques correspondant aux gardes diagonales $x - y = 4 - 3$, $x - y = 4 - 0$, $x - y = 2 - 0$, $x - y = 0 - 0$, $x - y = 0 - 3$, et $x - y = 2 - 3$.

Lemme 4.1.3
Soient Z une zone et P un polyèdre défini par des contraintes apparaissant dans les gardes de $\mathcal{A}$. Si $Z \cap P \neq \emptyset$, alors $Z \subseteq cyl(P)$

Preuve
Pour $cyl(P) = \{x \in \mathbb{R}_+ \mid A_d x \prec b_d\}$, $cyl(P)$ est l'ensemble des zones dont les valuations vérifiant les inéquations du systèmes $A_d x \prec b_d$, donc est une union de zones, et Z est alors une de ces zones. ■

[1]Ce n'est pas la notion habituelle de zone qui est utilisée ici.

Exemple :



Un polygone ouvert P en gris soutenu, et les zones qu'il intersecte. L'union de ces zones forment le cylindre de P.

Corollaire 4.1.4
Soient Z une zone et P un polyèdre tels que $P \cap Z \neq \phi$. Alors, pour toute valuation $v \in Z$, il existe $t \in \mathbb{R}$ et $v' \in P$ tel que $v = v' + t$.

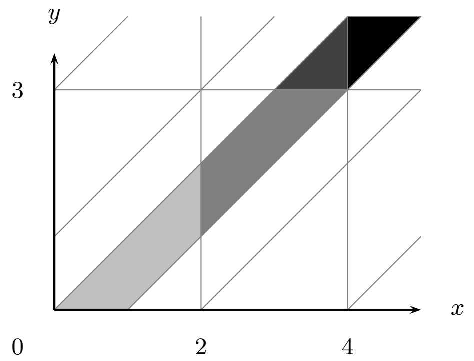
Définition 4.1.5
On appelle futur d'une valuation v l'ensemble $v^{\nearrow}$ des valuations $v+t$ pour tout $t \in \mathbb{R}_+$. On étend cette définition aux ensembles de valuations : le futur de V est l'ensemble $V^{\nearrow} = \{v + t : v \in V, t \in \mathbb{R}_+\}$.

Lemme 4.1.6
Soit Z une zone de l'automate temporisé $\mathcal{A}$. Soit H_1 et H_2 deux hyperplans définis par des contraintes des gardes de $\mathcal{A}$. Alors $Z \cap H_1^{\nearrow}$ et $Z \cap H_2^{\nearrow}$ sont comparables par inclusion.

Preuve
On suppose que ces deux ensembles sont non-vides, sinon le résultat est trivial. Soit $\mathcal{H}$ l'ensemble des hyperplans définis par les équations $x = c$, pour x et c tels que $x \sim c$ apparaisse dans une garde de $\mathcal{A}$. Notons que si $H_1 : x = c_1$ et $H_2 : y = c_2$ sont deux hyperplans distincts de $\mathcal{H}$ vérifiant $H_1 \cap H_2 \cap Z \neq \phi$, alors, pour $H : x - y = c_1 - c_2$, $Z \subseteq H$ (lemme 4.1.3 puisque $H = H^{\nearrow}$), et donc $Z \cap H_1 = Z \cap H_2$.
Soit $v \in Z$, la droite $\Delta : v + \mathbb{R} \cdot \mathbb{1}$ coupe tous les hyperplans de $\mathcal{H}$ en un unique point (car Δ n'est parallèle avec aucun hyperplan de $\mathcal{H}$), en des points $v+\lambda_1\mathbb{1}, \ldots, v+\lambda_k\mathbb{1}$. On peut donc ordonner les hyperplans de $\mathcal{H}$ selon les valeurs $\lambda_1, \ldots, \lambda_k$, deux hyperplans s'intersectant étant égaux pour cet ordre. De plus, cet ordre ne dépend pas de v : en effet, supposons qu'il existe $v' \in Z$, H_1 et H_2 deux hyperplans de $\mathcal{H}$, $t_1, t_2, t_1', t_2' \in \mathbb{R}$ tel que $v + t_1 \in H_1$, $v + t_2 \in H_2$, $v' + t_1' \in H_1$ et $v' + t_2' \in H_2$, $t_1 < t_2$ et $t_2' < t_1'$. $v + t_1, v + t_2, v' + t_1', v' + t_2'$ appartiennent à l'espace vectoriel de dimension 3 définie par les deux points v

et v' et la direction $\mathbb{1}$. Les segments $(v+t_1, v'+t_1')$ et $(v+t_2, v'+t_2')$ admettent donc un point d'intersection, qui par connexité appartient à $H_1 \cap H_2 \cap Z \neq \phi$, donc $H_1 \cap Z = H_2 \cap Z$, $t_1 = t_2$, ce qui contredit les hypothèses.
Soit donc maintenant $H_1, H_2 \in \mathcal{H}$ deux hyperplans ne s'intersectant pas dans Z (si c'est la cas, le lemme est évident) et sans perte de généralité $H_1 < H_2$ pour l'ordre qu'on vient de définir, on montre que $Z \cap H_2^{\nearrow} \subseteq Z \cap H_1^{\nearrow}$ par l'absurde. Supposons que $Z \cap H_2^{\nearrow} \nsubseteq Z \cap H_1^{\nearrow}$. Il existe une valuation $v' \in Z \cap (H_2^{\nearrow} \setminus H_1^{\nearrow})$. Par le lemme 4.1.4, il existe $v_1 \in H_1$ et $t_1 \in \mathbb{R}_+$ tels que $v' = v_1 - t_1$, et par définition il existe $v_2 \in H_2$ et $t_2 \in \mathbb{R}_+$ tels que $v' = v_2 + t_2$. Ceci contredit le fait que notre ordre ne dépend pas du choix de v. ■

Exemple :



Une zone, et les futurs de l'intersection de cette zone avec les différents hyperplans. Ces futurs forment bien une chaîne.

Lemme 4.1.7
Soit Z une zone de l'automate temporisé $\mathcal{A}$. Soit P_1 et P_2 deux polyèdres définies par des contraintes issues des gardes de $\mathcal{A}$. Alors $Z \cap P_1^{\nearrow}$ et $Z \cap P_2^{\nearrow}$ sont comparables par inclusion, et ces deux ensembles sont définissables comme l'intersection de Z avec un demi-espace (ouvert ou fermé).

Preuve
Pour $i \in \{1,2\}$, soit $\mathcal{H}_i' \subseteq \mathcal{H}$ l'ensemble des hyperplans H_i de $\mathcal{H}$ qui définissent des faces de P_i telles que $P_i \subseteq H_i + \mathbb{R}_+ \cdot \mathbb{1}$, et soit $H_i : x_i = c_i$ l'hyperplan maximal de $\mathcal{H}_i'$. Alors $x_i \sim_i c_i$ est une contrainte de P_i, avec $\sim_i \in \{=, >, \geq\}$. Alors $P_i^{\nearrow} \cap Z$ est l'ensemble des valuations de Z vérifiant la contrainte $x_i \sim_i c_i \vee x_i > c_i$. En effet, toute valuation de $P_i^{\nearrow}$ vérifie cette contrainte. Réciproquement soit v un élément de Z tel que $v \vDash x_i \sim_i c_i \vee x_i > c_i$. D'après le lemme 4.1.4, $Z \subseteq cyl(H_i)$, donc il existe $t \in \mathbb{R}$ tel que $v - t \in H_i$, et t est positif puisque sinon v ne vérifierait pas la contrainte $x_i \sim_i c_i \vee x_i > c_i$. De plus, toujours par le lemme 4.1.4, il existe $t' \in \mathbb{R}$ tel que $v + t' \in P_i$. S'il existe un tel t' négatif ou nul, alors $v \in P_1^{\nearrow}$, sinon $(v \notin P_i)$ $v + t'$ est plus grand composant par composant que v, donc v vérifie les

contraintes de la forme $z < c$ ou $z \leq c$ que $v + t'$ vérifie, et $v - t$ est plus petit composant par composant que v, donc v vérifie toutes les inéquations définissant P_i, donc $v \in P_i$, contradiction. Enfin, il suffit de comparer les contraintes $x_i \sim_i c_i$ pour comparer $Z \cap P_1^{\nearrow}$ et $Z \cap P_2^{\nearrow}$. ■

4.2 D'une transition à l'autre

On se pose le problème suivant : soit $t_1 : q \xrightarrow{g_1, R_1} q_0$ et $t_2 : q_1 \xrightarrow{g_2, R_2} q'$ deux transitions de l'automate temporisé $\mathcal{A}$ telles que $R_1 \neq \phi$ et $R_2 \neq \phi$. Sous quelles conditions existe-t-il des configurations (q, v) et (q', v') telles qu'il existe une exécution de $\mathcal{A}$ de (q, v) à (q', v') dont la première transition est t_1, la dernière est t_2 et toutes les transitions intermédiaires ne remettent pas d'horloge à zéro ? Soit $\mathcal{A}'$ l'automate temporisé $\mathcal{A}$ duquel on a retiré toutes les transitions qui remettent à zéro au moins une horloge. Pour toute garde g, on note P_g le polyèdre convexe associé à g.

Entrée : *Un automate temporisé $\mathcal{A}$, une zone Z de $\mathcal{A}$ et sa base $B \neq \phi$, deux états q_0 et q_1.*

Sortie : *Un polyèdre P tel que $v \in Z \cap P$ ssi il existe $v' \in B \cap cyl(v)$ et une exécution de (q_0, v') vers (q_1, v) dans $\mathcal{A}'$.*

Algorithme :

1. **Pour tout** *état* q, $P(q) \leftarrow \phi$
2. $P(q_0) \leftarrow Z$
3. **Tant qu'***il existe* $r \xrightarrow{g, \phi} s \in \mathcal{A}'$ *tel que* $P(s) \subsetneq (P(r) \cap P_g)^{\nearrow}$ **faire**
4. $\quad P(s) \leftarrow (P(r) \cap P_g)^{\nearrow}$
5. **Renvoyer** $P(q_1)$

Algorithme 1: $T(q_0, q_1, Z)$

Proposition 4.2.1
L'algorithme 1 termine.

Preuve
On remarque que les polyèdres attachés aux sommets ne font que croître pendant l'exécution de l'algorithme, et vérifient $P(p) = P(p)^{\nearrow} \subseteq Z$ pour tout sommet p. Donc les polyèdres manipulés sont tous définis par l'intersection d'une garde et de la zone Z, par le lemme 4.1.7. Comme le nombre de garde est fini et inférieur à la taille de l'automate, l'algorithme termine en temps polynomial. ■

Proposition 4.2.2
L'algorithme 1 est correct.

Preuve
Soit $v' \in B \cap cyl(v)$ et $\rho = (q_0, v') = (p_0, v_0) \to \ldots \to (p_k, v_k) = (q_1, v)$ une exécution de (q_0, v') vers (q_1, v) dans $\mathcal{A}'$. On prouve que $v_i \in P(p_i)$ pour tout $i \in [\![0, k]\!]$ par récurrence sur la longueur k de ρ. Si ρ est de longueur nulle, alors $q_0 = q_1$, $v = v' \in P(q_0) = Z$. Soit $i \in [\![0, k-1]\!]$, supposons que $v_i \in P(p_i)$. Si la $(i+1)^{\text{e}}$ transition de ρ est une transition de délai, comme $P(p_i) = P(p_i)^{\nearrow}$, $v_{i+1} \in P(p_{i+1})$. Sinon, c'est une transition d'action, donc il existe une transition $p_i \to g_{i+1}, \not\phi p_{i+1}$, $v_i \models g_{i+1}$ donc $v_i \in P_{g_{i+1}}$ et $v_i \in P(p_i)$, donc $v_i \in P(p_{i+1})$.
Réciproquement, on montre que l'algorithme possède l'invariant suivant : pour tout sommet p, pour toute valuation $u \in P(p)$, il existe une valuation $v \in B$ et une exécution ρ de (q_0, v) vers (p, u). En effet, comme $Z = B^{\nearrow}$, c'est vrai lors de l'initiation de l'algorithme. Ensuite, supposons que l'algorithme ajoute la valuation u dans $P(p)$ à un instant donné de l'exécution de l'algorithme. Alors à cet instant, il existe $r \xrightarrow{g, \not\phi} p \in \mathcal{A}'$ tel que $P(p) \subsetneq P(r) \cap P_g^{\nearrow}$. Alors, $u \in (P(r) \cap P_g)^{\nearrow}$, donc il existe $u' \in P(r) \cap P_g$ et $t \in \mathbb{R}_+$ tels que $u = u' + t$. Par hypothèse de récurrence, il existe une exécution depuis (q_0, v) vers (r, u'), que l'on peut étendre par la transition d'action vers p, puis une transition de délai t, constituant alors une exécution de (q_0, v) vers (p, u). ■

On peut donc calculer l'effet de transitions sans remise à zéro raisonnablement en faisant un calcul zone par zone. Soit Z une zone. D'après le lemme 4.1.7, $T(q_0, q_1, Z)$ peut être décrit comme l'intersection de Z et d'un demi-espace défini par la contrainte $post(q_0, q_1, Z) : y \sim c$. Puisque la transition t_1 remet au moins une horloge à zéro, disons x, prenons B la base de Z sur x. Cette base peut être définie par une garde g avec contraintes diagonales (c'est la conjonction de la garde définissant Z et de la contrainte $x = 0$). On définit $pre(q_0, q_1, Z, R) = g[R \leftarrow 0]$ par la garde g dans laquelle on a remplacé toutes les occurences d'une variable de R par la valeur zéro. $pre(q_0, q_1, Z)$ définit l'ensemble des valuations dont la projection par $\pi_R : v \to \pi_R(v)$ est dans B, avec $\pi_R(v)(z) = v(z)$ si $z \notin R$, zéro sinon. On peut alors simuler l'exécution passant d'abord par t_1, puis un nombre quelconque de transitions sans remise à zéro, puis t_2, en ajoutant les transitions suivantes pour chaque zone $Z : t_1' : q \xrightarrow{g_1 \wedge pre(q_0, q_1, Z, R_1), R_1} r$ et $t_2' : r \xrightarrow{g_2 \wedge post(q_0, q_1, Z), R_2} q'$ où r est un nouvel état. On remarque que dans le cas des automates temporisés à deux horloges, la contrainte $pre(q_0, q_1, Z, R_1)$ n'est pas une contrainte diagonale, puisqu'au moins une des deux horloges n'apparait pas dans la contrainte.

4.3 Construction de l'automate avec remise à zéro systématique

Soit $\mathcal{T}$ l'ensemble des transitions qui remettent au moins une horloge à zéro, et $\mathcal{Z}$ l'ensemble des zones de l'automate. On va définir l'ensemble des états Q_R

du nouvel automate $\mathcal{A}_R$ par $\mathcal{T}^2 \times \mathcal{Z}$. Soit donc $(t_1, t_2, Z) \in Q_R$. Intuitivement, t_1 represente la transition que l'on vient de prendre, t_2 la transition que l'on s'apprête à utiliser avec une valuation dans Z, sans compter les transitions sans remise à zéro. Les transitions se font donc entre deux états (t_1, t_2, Z_1) et (t_2, t_3, Z_2), qui correspond à prendre l'ancienne transition t_2. On note $t_i : q_i \xrightarrow{g_i, R_i} q_i'$, g la contrainte définissant la base de Z_2. On rajoute alors la transition :

$$(t_1, t_2, Z_1) \xrightarrow{g_2 \wedge post(q_1', q_2, Z_1) \wedge pre(q_2', q_3, Z_1, R_2), R_2} (t_2, t_3, Z_2)$$

Pour l'initialisation, on ajoute au sommet initial q_0 de $\mathcal{A}$ la transition t_0 : $q_0 \xrightarrow{v=0, X} q_0$ avant la construction. Comme cette transition n'a aucun effet, on ne change pas le comportement de $\mathcal{A}$. Soit alors Z_0 la zone contenant la valuation v_0, on définit l'état (t_0, t_0, Z_0) comme l'état initial du nouvel automate $\mathcal{A}_R$.
Pour pouvoir vérifier l'accessibilité d'états, on rajoute un nouvel état q_n dans $\mathcal{A}_R$ pour chaque ancien état q dans $\mathcal{A}$. Pour chaque état $(t_1, t_2, Z) \in Q_R$, on calcule l'ensemble des états accessible depuis q_1' avec une valuation de la base de Z, en utilisant l'algorithme 1, l'ensembles des états accessibles étant l'ensemble des états q tels que $P(q) \neq \emptyset$ à la fin de l'exécution de l'algorithme. On ajoute alors une transition $(t_1, t_2, Z) \xrightarrow{\top, X} q_n$. Il n'y a pas de transitions sortant de q_n. Vérifier l'accessibilité de q dans $\mathcal{A}$ consiste donc à vérifier celle de q_n dans $\mathcal{A}_R$.

Proposition 4.3.1
L'automate avec remise à zéro systématique correspondant à l'automate temporisé à k horloges $\mathcal{A}$ se construit en temps $\mathcal{O}(p(n, |\mathcal{Z}_\mathcal{A}|))$, avec $n = |\mathcal{A}|$ et p un polynôme.

Preuve
Il y a $\mathcal{O}(n^3 \cdot |\mathcal{Z}_\mathcal{A}|^2)$ transitions, à construire, chacune se construisant avec une complexité égale à celle de l'algorithme 1. Cet algorithme s'exécute en temps polynomial en n d'après la proposition 4.2.1. Comme une zone est l'intersection d'au plus deux contraintes diagonales (consécutives) par couple d'horloges, on peut construire l'ensemble des zones en temps polynomial en le nombre de zones.

■

Corollaire 4.3.2
Pour les automates temporisés à deux horloges, on peut construire l'automate avec remise à zéro systématique correspondant en temps polynomial.

REMARQUE 4.3.3
Et même dès que le nombre d'horloges est borné, car dans ce cas le nombre de zone est polynomial.

Chapitre 5

Étude d'un cas particulier

Dans cette partie, on montre la NP-complétude du problème de l'accessibilité pour une sous-classe des automates à deux horloges. On supposera que tous les automates considérés ont des gardes convexes. De plus, en vertu du résultat du chapitre précédent, on suppose que toutes les transitions remettent à zéro au moins une horloge.

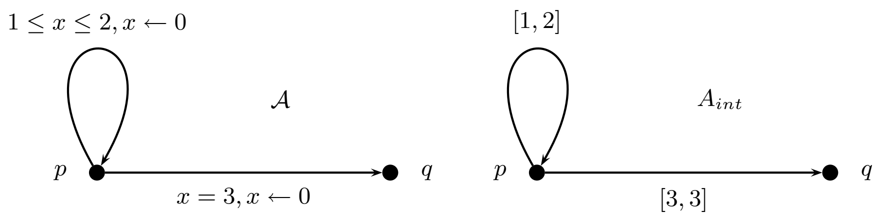
5.1 Délais dans les automates à une horloge

On s'intéresse en premier lieu au problème suivant : étant donné un automate à une seule horloge, qu'on appelle x, et qui est remise à zéro après chaque transition, une configuration $(q_0, 0)$ de cet automate et une valeur $t \in \mathbb{R}_+$, en quelle complexité peut-on décider si la configuration $(q, 0)$ est atteignable depuis $(q_0, 0)$ par une exécution dont la somme des délais est exactement t ? On montre que ce problème est NP-complet.

Soit $\mathcal{A}$ un automate temporisé à une horloge, et C l'ensemble des valeurs apparaissant dans les gardes de $\mathcal{A}$. Puisque les gardes de $\mathcal{A}$ sont supposées convexes, elles sont de la forme $a \sim_1 x \sim_2 b$, avec $\sim_1, \sim_2 \in \{<, \leq\}$, autrement dit $x \in I$, pour un certain intervalle I. Soit A_{int} l'automate (non-temporisé) sur l'alphabet des intervalles de $\mathbb{R}_+$ dont les bornes sont des éléments de $C \cup \{\infty\}$, obtenu en remplaçant chaque transition de garde $x \in I$ dans $\mathcal{A}$ par la transition étiquetée par I. À toute exécution ρ de la forme $(q_0, 0) \rightarrow (q_0, t_0) \rightarrow (q_1, 0) \dots (q_{n-1}, t_{n-1}) \rightarrow (q_n, 0)$ de $\mathcal{A}$, on associe une exécution $\varphi(\rho) = q_0 \xrightarrow{I_0} q_1 \dots q_{n-1} \xrightarrow{I_{n-1}} q_n$ de A_{int} telle que pour tout $i \in [\![0, n-1]\!]$, $t_i \in I_i$, et donc $\sum_{i \in [\![0,n-1]\!]} t_i \in \sum_{i \in [\![0,n-1]\!]} I_i$. Réciproquement, pour tout exécution de A_{int} reconnaissant le mot $I_0, \dots, I_{n-1}$, pour tout $t \in \sum_{i \in [\![0,n-1]\!]} I_i$, il existe une décomposition $t = t_0 + \dots t_{n-1}$ avec $\forall i \in [\![0, n-1]\!], t_i \in I_i$, et donc il existe une exécution ρ de $\mathcal{A}$ de la forme $(q_0, 0) \rightarrow (q_0, t_0) \rightarrow (q_1, O) \dots (q_{n-1}, t_{n-1}) \rightarrow (q_n, 0)$.
Il suffit donc pour résoudre le problème posé, de pouvoir décider s'il existe un

mot reconnu par A_{int} avec état initial q_0 et état final q, et dont la somme des lettres contient la valeur t (on dira plus simplement que l'automate reconnait t). Puisque l'on souhaite résoudre le problème avec un algorithme NP, on pourrait deviner un chemin de A_{int}, faire la somme de ces lettres et vérifier que t est bien un élément de cette somme. Mais la longueur du plus court chemin dont la somme des lettres contient t peut être exponentielle, comme dans l'exemple suivant.

Exemple :



Dans cet exemple, pour passer de la configuration $(p, 0)$ à la configuration $(q, 0)$ avec un délai $2n + 3$, l'exécution la plus courte en nombre de transition prend n fois la transition qui boucle avec un délai 2 puis la transition $p \to q$ avec un délai 3. Cela correspond au mot $([1, 2])^n[3, 3]$ dans l'automate A_{int}. il faut donc au moins $n + 1$ transitions, ce qui constitue un nombre exponentiel de transitions par rapport à la taille du problème.

Cependant, l'opération de sommation sur les intervalles est commutative et associative, on peut donc grouper les intervalles identiques, et les sommer ensemble, ce qui revient à multiplier les bornes de l'intervalle. Ainsi, on n'a pas besoin de retenir l'ordre dans lequel les transitions sont franchies par une exécution, mais seulement le nombre de fois que chaque transition a été franchie. L'algorithme NP devient donc : on devine le nombre de passage dans chaque transition, on vérifie que ces valeurs correspondent à une exécution, puis pour chaque transition, on somme les intervalles, on somme les contributions de chaque transition et on vérifie que t appartient à l'intervalle obtenu. il faut donc décrire comment on fait pour vérifier que le choix du nombre de passage dans chaque intervalle correspond bien à une exécution possible. il faut aussi montrer que le nombre de passages par chaque transition est simplement exponentiel, donc codable en taille polynomiale. On note $I(\tau)$ l'intervalle qui étiquette la transition τ. On rappelle qu'un graphe Eulérien est un graphe connexe dont tous les degrés sont pairs.

Lemme 5.1.1
Soient $\mathcal{A}$ un automate temporisé à une horloge remise à zéro à chaque transition, p et q deux de ses états, et $t \in \mathbb{R}_+$. Alors il existe une exécution de $(p, 0)$ vers $(q, 0)$ reconnaissant t ssi il existe une exécution de $(p, 0)$ vers $(q, 0)$ reconnaissant

Entrée : *A_{int} un automate d'intervalles, p et q deux états de cet automate, $t \in \mathbb{R}_+$.*

Sortie : *vrai ssi il existe un chemin de p vers q, dont la somme des intervalles contient t. (algorithme non-déterministe)*

Algorithme :

1. $k \leftarrow |Q_{A_{int}}| \cdot (\lceil t \rceil + 2)$
2. **Pour toute** *transition* τ, **Choisir** $w(\tau) \in [\![0, k]\!]$
3. *Soit G le graphe sur les sommets de A_{int} dont les arêtes sont les transitions de A_{int} avec multiplicité égale à leur poids w*
4. $E(G) \leftarrow (q, p)$
5. **Si** *G n'est pas Eulérien*
6. **Alors renvoyer** *faux*
7. **Calculer** $I = \sum_{\tau \text{ transition de } A_{int}} w(\tau) I(\tau)$
8. **Renvoyer** *la valeur de vérité de* $t \in I$

Algorithme 2: $delai(p, q, t)$

t utilisant moins de $|Q_{\mathcal{A}}| \cdot (\lceil t \rceil + 2)$ transitions d'actions.

Preuve
Une des directions est triviale. Soit $\rho = (q_0, 0) \to (q_0, t_0) \to (q_1, t_0) \ldots (q_n, 0)$ une exécution minimisant n, avec $q_0 = p$ et $q_n = q$. Puisque n est minimal, il n'y a pas deux transitions de délai successives dans ρ. Supposons que $n > |Q_{\mathcal{A}}| \cdot (\lceil t \rceil + 2) = m$. Alors il existe un état r qui apparait au moins $\lceil t \rceil + 2 = t' + 2$ parmi les états traversés par ρ : $q_{i_1} = \ldots = q_{i_{t'}} = r$ juste avant une transition d'action. On note $\rho_j = (q_{i_j}, 0) \to \ldots (q_{i_{j+1}}, 0)$ la sous-exécution de ρ entre la ${i_j}^{\text{e}}$ configuration et la ${i_{j+1}}^{\text{e}}$ configuration, pour $j \in [\![1, t' + 1]\!]$. Soit $I(\rho_j)$ la somme des intervalles des transitions d'actions empruntées par ρ_j. Si $I(\rho_j) = [0, 0]$, alors la sous-exécution ρ_j peut être retirée de ρ, donnant une exécution strictement plus courte reconnaissant t, ce qui contredit la minimalité de n. Soit $I = \sum_{j \in [\![1, t']\!]} I(\rho_j)$, soient $a = \inf I$ et $b = \sup I$. Alors, $b \geq t'$, puisque la borne supérieure de chacun des $I(\rho_j)$ est au moins 1. Donc $t \in I + I(\rho_{t'+1})$ implique $t \in I$, donc on peut enlever de ρ la sous-exécution $\rho_{t'+1}$, ce qui contredit la minimalité de n à nouveau. ■

Dans l'algorithme 2, on prend en entrée un réel quelconque, mais ce qui est intéressant est de savoir quel est la valeur de sa partie entière et si sa partie fractionnaire est nulle ou non, puisque les intervalles reconnus par automates d'intervalles ont des bornes entières. On peut donc supposer que t est codé en espace $\lceil \log_2 t \rceil + 1$.

Proposition 5.1.2
L'algorithme 2 termine en temps polynomial et est correct.

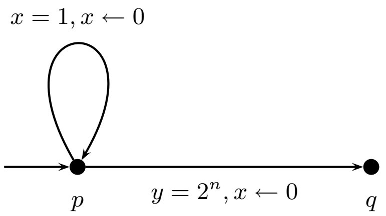
Preuve
La terminaison est triviale : il n'y a pas de boucle, et le caractère polynomial vient du fait que les valeurs utilisées sont de taille polynomiale, et que déterminer qu'un graphe est Eulérien se fait en temps linéaire.
La correction vient du fait qu'un chemin auquel on ajoute une arête entre les deux extrémités est exactement un graphe eulérien et réciproquement. De plus, grâce au lemme 5.1.1, on sait que la contrainte sur le choix du poids des transitions est légitime. ■

5.2 Compression d'une exécution

On suppose que toutes les exécutions d'automates temporisés sont telles qu'il n'y a jamais deux transitions de délais successives. Soit $\mathcal{A}$ un automate temporisé à deux horloges, que l'on appelle x et y. Soit $C_x \subset \mathbb{N}$ l'ensemble des valeurs auxquelles x est comparé dans les gardes de $\mathcal{A}$. On définit similairement C_y. Notons $n = |\mathcal{A}|$, alors d'évidence $|C_x| < n$ et $|C_y| < n$. Soit ρ une exécution de $\mathcal{A}$ qui ne remet jamais l'horloge y à zéro. On souhaiterait pouvoir coder une telle exécution en espace polynomial, de telle sorte qu'un algorithme NP puisse décider s'il s'agit bien du codage d'une exécution de $\mathcal{A}$. Puisque cette exécution ne remet jamais l'horloge y à zéro, on va découper cette exécution en plusieurs morceaux, de telle sorte que dans chaque morceau, les valeurs de y dans les différentes configurations utilisées vérifient les mêmes contraintes des gardes de $\mathcal{A}$.
Formellement, soit $0 = b_0 < b_1 < \ldots < b_m = \infty$ les valeurs de $C_y \cup \{0, \infty\}$. Soit $\rho = (q_0, (0, y_0)) \rightarrow (q_0, (x_0, y_0 + x_0)) \rightarrow (q_1, (0, y_1)) \ldots (q_l, (0, y_l))$ une exécution qui ne remet pas y à zéro (et donc remet systématiquement x à zéro). Soit Y les intervalles de la forme $]b_i, b_{i+1}[$ ou $[b_i, b_i]$ avec $i \in [\![0, m-1]\!]$, Soit $I \in Y$, on note ρ_I la plus grande sous-exécution de ρ de la forme $(q_i, (0, y_i)) \rightarrow \ldots \rightarrow (q_j, (0, y_j))$ avec $y_i, y_j \in I$ (on rappelle que la valeur de y ne fait que croître).

Définition 5.2.1
on appelle compression de ρ, et on note $comp(\rho)$, l'exécution ρ dans laquelle on a remplacé chaque sous exécution $\rho_I : (q_i, (0, y_i)) \rightarrow \ldots \rightarrow (q_j, (0, y_j))$, $I \in Y$, par une unique transition compressée $(q_i, (0, y_i)) \rightsquigarrow (q_j, (0, y_j))$.

Exemple :



Pour atteindre l'état q l'exécution la plus courte est $(p,(0,0)) \to (p,(0,1)) \to \ldots \to (p,(0,2^{n-1})) \to (p,(1,2^n)) \to (q,(0,2^n))$. Cette exécution se compresse en $(p,(0,0)) \rightsquigarrow (p,(0,2^{n-1})) \to (p,(1,2^n)) \to (q,(0,2^n))$, ce qui est nettement plus concis.

Proposition 5.2.2
Pour toute exécution ρ ne remettant pas l'horloge y à zéro, la longueur de $comp(\rho)$ est polynomiale en la taille de $\mathcal{A}$.

Preuve
Il y a au plus deux configurations de la forme $(q,(0,y_0)), y_0 \in I$ pour chaque intervalle $I \in Y$, et comme le nombre de transitions de délai est borné par le nombre de configurations telle que l'horloge x vaut zéro, il y a au plus 4 configurations pour lesquelles la valeur de y est dans I. Comme $|Y| < n$, la longueur de l'exécution compressée aussi. ■

Proposition 5.2.3
Soit $t : (p,(0,y_0)) \rightsquigarrow (q,(0,y_1))$ une transition compressée, où y_0 et y_1 sont des rationnels. On peut vérifier que t est bien la compression d'une exécution de $\mathcal{A}$ par un algorithme NP.

Preuve
Pour cela, on vérifie d'abord qu'il existe $I \in Y$ tel que $y_0, y_1 \in I$. Ensuite, on modifie $\mathcal{A}$ en l'automate temporisé $\mathcal{A}_I$ à une seule horloge, en enlevant les transitions qui ne remettent pas y à zéro et en remplaçant dans les gardes de $\mathcal{A}$ les contraintes $y \in J$ par $\top$ si $I \subseteq J$ et $\bot$ sinon ($I \cap J = \emptyset$). Cet automate correspond à $\mathcal{A}$ lorsqu'on travaille avec une valeur d'horloge y dans l'intervalle I. Il suffit alors de vérifier que dans $\mathcal{A}_I$ on peut atteindre la configuration $(q,0)$ depuis $(p,0)$ avec un délai $y_1 - y_0$, ce qui se fait en NP grâce à l'algorithme 2. ■

Corollaire 5.2.4
Soit σ une exécution de $\mathcal{A}$ avec transitions compressées sur x ou y. On peut vérifier en NP s'il existe une exécution ρ de $\mathcal{A}$ telle que $\sigma = comp(\rho)$.

Preuve
Il suffit de vérifier chaque transition compressée. ■

5.3 Taille d'une exécution compressée

Proposition 5.3.1
Soit $(q_0, x_0, y_0), \ldots, (q_k, x_k, y_k)$ une exécution compressée de $\mathcal{A}$ depuis l'état initial, avec $\mathcal{A}$ un automate à deux horloges. Alors il existe $\overline{x_0}, \ldots, \overline{x_k}, \overline{y_0}, \ldots, \overline{y_k}$ tels que $(q_0, \overline{x_0}, \overline{y_0}), \ldots, (q_k, \overline{x_k}, \overline{y_k})$ est une exécution compressée et pour tout $i \in [\![0, k]\!]$, les conditions suivantes sont vérifiées :

(i) $[(q_i, (x_i, y_i))] = [(q_i, (\overline{x_i}, \overline{y_i}))]$

(ii) $\overline{x_i}, \overline{y_i} \in 2^{-i-1}\mathbb{N}$.

Preuve
Sans perte de géneralité, on suppose qu'il n'y a pas deux transitions de délais successives, et que toute transition remet une des horloges à zéro. Par récurrence sur k.
Si $k = 0$, alors $\overline{x_0} = x_0 = \overline{y_0} = y_0 = 0$, et les conditions sont trivialement vraies. Soit $k \geq 1$ un naturel. Supposons que la proposition est vraie au rang $k - 1$. Soit $(q_0, x_0, y_0), \ldots, (q_k, x_k, y_k)$ une exécution compressée de $\mathcal{A}$ depuis l'état initial. Par hypothèse de récurrence, on peut construire une exécution compressée $(q_0, \overline{x_0}, \overline{y_0}), \ldots, (q_k, \overline{x_k}, \overline{y_k})$ telle que (i) et (ii) soient vérifiées. Dans l'exécution initiale, la transition $(q_{k-1}, (x_{k-1}, y_{k-1})) \to (q_k, (x_k, y_k))$ est :

- ou bien une transition d'action, sans perte de généralité, $x_k = 0$, $y_k = y_{k-1}$. On pose $\overline{x_k} = 0$, $\overline{y_k} = \overline{y_{k-1}}$. Par hypothèse de récurrence, on a que $[(q_{k-1}, (x_{k-1}, y_{k-1}))] = [(q_{k-1}, (\overline{x_{k-1}}, \overline{y_{k-1}}))]$, donc $(q_{k-1}, \overline{x_{k-1}}, \overline{y_{k-1}}) \to (q_k, \overline{x_k}, \overline{y_k})$ est une transition d'action pour $\mathcal{A}$ et $[(q_k, (x_k, y_k))] = [(q_k, (\overline{x_k}, \overline{y_k}))]$. (ii) est évident pour le rang k.
- ou bien une transition de délai, sans perte de généralité, $x_{k-1} = 0$. Dans ce cas, il y a cinq possibilités.
  (a) Si $[(q_k, (x_k, y_k))]$ est un singleton, soit $\overline{x_k} = x_k$ and $\overline{y_k} = y_k$. (i) est vraie. x_k et y_k sont des naturels, donc (ii) est vraie aussi. Alors $y_{k-1} \in \mathbb{N}$, donc $\overline{y_{k-1}} = y_{k-1}$ car $[(q_{k-1}, (x_{k-1}, y_{k-1}))] = [(q_{k-1}, (\overline{x_{k-1}}, \overline{y_{k-1}}))]$ par hypothèse de récurrence. Il existe donc bien une transition de délai $(q_{k-1}, \overline{x_{k-1}}, \overline{y_{k-1}}) \to (q_k, \overline{x_k}, \overline{y_k})$.
  (b) Si $[(q_k, (x_k, y_k))]$ est une région de la forme $\{(q_k, (x, y) \mid x = a, b < y < b+1\}$ pour a et b deux naturels, posons $\overline{x_k} = a$ et $\overline{y_k} = \overline{y_{k-1}} + a$. La condition (ii) est vérifiée puisque $b \in \mathbb{N}$, et il existe bien une transition de délai $(q_{k-1}, (\overline{x_{k-1}}, \overline{y_{k-1}})) \to (q_k, (\overline{x_k} \overline{y_k}))$. On montre que $\lfloor y_k \rfloor = \lfloor \overline{y_k} \rfloor$ pour prouver (i) :

$$\begin{array}{rcl} \overline{y_k} & = & \lfloor \overline{y_{k-1}} \rfloor + a + \{\overline{y_{k-1}}\} \\ & = & \lfloor y_{k-1} \rfloor + a + \{\overline{y_{k-1}}\} \\ y_k & = & \lfloor y_{k-1} \rfloor + a + \{y_{k-1}\} \end{array}$$

  On en déduit que $\overline{y_k}$ et y_k ont la même partie entière $\lfloor y_{k-1} \rfloor + a$.
  (c) Si $[(q_k, (x_k, y_k)]$ est une région de la forme $\{(q_k, (x, y) \mid a < x < a+1, y = b\}$ pour a et b deux naturels, on pose $\overline{x_k} = b - \overline{y_{k-1}}$ et $\overline{y_k} = b$.

Comme dans le cas (b), il suffit de montrer que $\lfloor x_k \rfloor = \lfloor \overline{x_k} \rfloor$:

$$\begin{array}{rcl} \overline{x_k} & = & b - \lfloor \overline{y_{k-1}} \rfloor - \{\overline{y_{k-1}}\} \\ & = & b - \lfloor y_{k-1} \rfloor - \{\overline{y_{k-1}}\} \\ x_k & = & b - \lfloor y_{k-1} \rfloor - \{y_{k-1}\} \end{array}$$

Comme $\{y_{k-1}\} = 0$ ssi $\{\overline{y_{k-1}}\} = 0$, on a bien $\lfloor x_k \rfloor = \lfloor \overline{x_k} \rfloor$.

(d) Si $[(q_k, (x_k, y_k))]$ est une région de la forme $\{(q_k, (x, y) \mid a < x < a+1, b < y < b+1, \{x\} = \{y\}\}$ pour a et b deux naturels, posons $\overline{x_k} = a + 1 - 2^{-1-k}$ et $\overline{y_k} = b + 1 - 2^{-1-k}$. Les conditions (i) et (ii) sont donc satisfaites. Par hypothèse d'induction, $x_{k-1} - y_{k-1} = \overline{x_{k-1}} - \overline{y_{k-1}} = a - b$ et $\overline{x_{k-1}} < \overline{x_k}$ car $a > \overline{x_k - 1} \in 2^{-k}\mathbb{N}$, donc il existe bien une transition $(q_{k-1}, (\overline{x_{k-1}}, \overline{y_{k-1}})) \to (q_k, (\overline{x_k}, \overline{y_k}))$ de délai $\overline{x_k} - \overline{x_{k-1}}$.

(e) Si $[(q_k, (x_k, y_k))]$ est une région de la forme $\{(q_k, (x, y) \mid a < x < a+1, b < y < b+1, \{x\} < \{y\}\}$ pour a et b deux naturels, on pose $\overline{y_k} = b + 1 - 2^{-1-k}$ et $\overline{x_k} = \overline{y_k} - \overline{y_{k-1}}$. La condition (ii) est donc vérifiée. Il existe une transition de délai $(q_{k-1}, (\overline{x_{k-1}}, \overline{y_{k-1}})) \to (q_k, (\overline{x_k}, \overline{y_k}))$ puisque $\overline{y_k} > \overline{y_{k-1}} \in 2^{-k}\mathbb{N}$. De plus, $0 < \{\overline{x_k}\} < \{\overline{y_k}\} = 1 - 2^{-1-k}$ et :

$$\begin{array}{rcl} \overline{x_k} & = & \lfloor \overline{y_k} \rfloor - \lfloor \overline{y_{k-1}} \rfloor + \{\overline{y_k}\} - \{\overline{y_{k-1}}\} \\ & = & \lfloor y_k \rfloor - \lfloor y_{k-1} \rfloor + \{\overline{y_k}\} - \{\overline{y_{k-1}}\} \\ x_k & = & \lfloor y_k \rfloor - \lfloor y_{k-1} \rfloor + \{y_k\} - \{y_{k-1}\} \end{array}$$

Or, $0 < \{\overline{y_k}\} - \{\overline{y_{k-1}}\} < 1$ car $0 \neq \{\overline{y_k}\} \in 2^{-k}\mathbb{N}$ et $\{\overline{y_{k-1}}\} = 1 - 2^{-1-k}$, et comme $\lfloor y_{k-1} \rfloor = \lfloor y_{k-1} - x_{k-1} \rfloor = \lfloor y_k - x_k \rfloor = b - a$, on a $\lfloor x_k \rfloor + \{x_k\} = \lfloor y_k \rfloor + \{y_k\} - \lfloor y_{k-1} \rfloor - \{y_{k-1}\} = \lfloor x_k \rfloor + \{y_k\} - \{y_{k-1}\}$, on en déduit $\{x_k\} = \{y_k\} - \{y_{k-1}\}$, donc $\lfloor x_k \rfloor = \lfloor \overline{x_k} \rfloor$, et finalement $[(q_k, (x_k, y_k))] = [(q_k, (\overline{x_k}, \overline{y_k}))]$. La preuve est symétrique lorsque $\{x_k\} > \{y_k\}$.

– ou bien une transition compressée, $x_k = x_{k-1} = 0$. Soit $l = y_k - y_{k-1}$. Supposons que l est un entier naturel, alors on pose $\overline{y_k} = \overline{y_{k-1}} + l$, alors $[(q_k, (x_k, y_k))] = [(q_k, (\overline{x_k}, \overline{y_k}))]$ (puisque les parties entières des valeurs pour y sont les mêmes), et donc il existe bien une transition compressée de $(q_{k-1}, (\overline{x_{k-1}}, \overline{y_{k-1}}))$ vers $(q_k, (\overline{x_k}, \overline{y_k}))$. On suppose donc maintenant que $a < l < a + 1$ pour un entier a. Par la correction de l'algorithme 2, on sait que si l est accepté pour l'automate d'intervalle construit à partir de $\mathcal{A}$, alors tout $l' \in]a, a+1[$ est accepté.

Si y_{k-1} est un entier, alors $\overline{y_{k-1}}$ aussi et ces deux entiers sont égaux. Soit $l' = a + \frac{1}{2}$, on pose $\overline{y_k} = \overline{y_{k-1}} + l$, et les conditions (i) et (ii) sont vérifiées de même que l'existence d'une transition compressée. Il reste à traiter le cas $c < y_{k-1} < c + 1$ pour un entier c. On distingue 3 sous-cas :

(a) Si $\{y_{k-1}\} + \{l\} > 1$, alors $c + l + 1 < y_k < c + l + 2$. On pose $\overline{y_k} = \overline{y_{k-1}} + l + 1 - 2^{-k-1} \in 2^{-k-1}\mathbb{N}$, alors $\lfloor y_k \rfloor = \lfloor \overline{y_k} \rfloor$, donc les régions sont les mêmes, et il existe bien une transition compressée.

(b) Si $\{y_{k-1}\} + \{l\} = 1$, alors $c + l + 1 = y_k$, on pose donc $\overline{y_k} = y_k$, et il existe bien une transition compressée de longueur $c + l + 1 - \overline{y_{k-1}} = l + 1 - \{\overline{y_{k-1}}\} \in]l, l+1[$.

(c) Si $\{y_{k-1}\} + \{l\} < 1$, alors $c + l < y_k < c + l + 1$. On pose $\overline{y_k} = \overline{y_{k-1}} + l + 2^{-k-1} \in 2^{-k-1}\mathbb{N}$, alors $\lfloor y_k \rfloor = \lfloor \overline{y_k} \rfloor$, donc les régions sont les mêmes, et il existe bien une transition compressée.

■

5.4 Applications

Théoreme 5.4.1
Le problème de l'accessibilité dans les automates temporisés à deux horloges est NP-complet sur la classe des automates qui ne remettent jamais l'un des deux horloges à zéro.

Preuve
Pour la NP-difficulté, il suffit de remarquer que la réduction de la section 3.1 crée un automate qui ne remet jamais y à zéro.
Le caractère NP vient du fait qu'on peut deviner une exécution codable en taille polynomiale par la proposition 5.3.1, puis on peut vérifier que cette exécution est correcte par le corollaire 5.2.4. ■

De manière plus générale et pour les mêmes raisons :

Théoreme 5.4.2
Le problème de l'accessibilité dans les automates temporisés à deux horloges est NP-complet sur la classe des automates qui vérifient que dans toute composante connexe, une seule des deux horloges est remise à zéro.

Avec cette méthode, le résultat le plus général que l'on puisse écrire est le suivant (et les deux résultats précédents en sont des conséquences) :

Théoreme 5.4.3
Soit $\mathcal{C}$ une classe d'automate et $p \in \mathbb{R}[X]$ un polynôme tels que pour tout automate $\mathcal{A}$ de cette classe, pour tout état q de cet automate, l'état q est accessible depuis la configuration initiale ssi il est accessible par une exécution vérifiant que le nombre de remise à zéro de x suivie d'une remise à zéro de y est inférieur à $p(|\mathcal{A}|)$. Alors l'accessibilité est un problème NP sur cette classe.

Preuve
En compressant tant que possible une telle exécution, on obtient une exécution compressée de taille polynomiale en $|\mathcal{A}|$, puisque chaque sous-exécution qui ne remet qu'une horloge à zéro peut être compressée avec une taille polynomiale, et que le nombre de telles sous-exécution est inférieur à $2p(|\mathcal{A}|) + 1$. ■

Chapitre 6

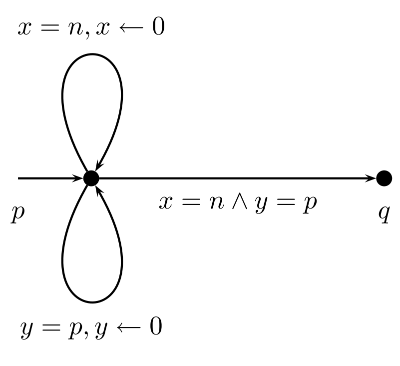
Pistes

Dans ce chapitre, nous allons présenter de manière assez informelle les différentes pistes que nous avons testées afin de résoudre le problème de l'accessibilité des automates temporisées à deux horloges dans le cas le plus général, et nous tenterons d'expliquer pourquoi ces idées n'ont pas donné de résultat.

6.1 Limites du théorème 5.4.3

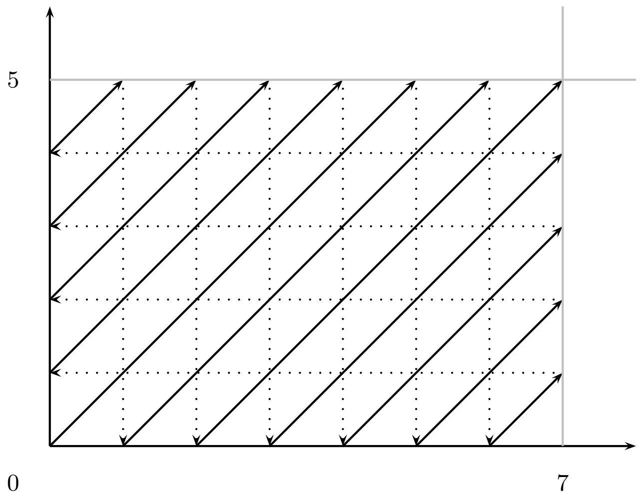
Nous commençons par montrer que le théorème 5.4.3 ne suffit par pour résoudre le problème de l'accessibilité dans le cas le plus général. Pour cela, il suffit de trouver une classe infinie d'automates telle que pour chaque automate de cette classe, un des états est accessible depuis la configuration initiale, mais que tout chemin permettant l'accès à cet état possède un nombre d'alternance entre remise à zéro sur x et sur y qui est exponentiel en la taille de l'automate. Nous présentons une telle classe d'automates, chacun possédant deux états, seule la valeur des constantes utilisées varient entre deux de ces automates (figure 6.1).

FIG. 6.1 – Automate de Bezout



Soient n et p deux entiers naturels, le plus petit délai t tel que l'état q est accessible depuis $(p,(0,0))$ vérifie $t = \text{ppcm}(p,n)$. Notamment, si n et p sont premiers entre eux, $t = n \cdot p$. Dans ce cas, sans perte de généralité, supposons $n < p$. Entre deux remises à zéro de y, il y a au moins une remise à zéro de x sur chaque exécution atteignant q. Comme y est remise à zéro au moins $n-1$ fois, il y a au moins $2n-2$ alternances entre remises à zéro de x et de y, ce qui est exponentiel en la taille du codage de n. Pour $n \in \mathbb{N}$ et $p = n+1$, c'est exponentiel en la taille de l'automate temporisé, on a donc une classe d'automates qui vérifie les conditions recherchées.

Exemple :



Pour $n = 7$ et $p = 5$, une représentation graphique de la plus courte exécution atteignant p. Les transitions de délai sont en trait continu, les transitions d'action en pointillés. L'exécution passe dans toutes les régions entières sauf $(7,0)$ et $(0,5)$. Les remises à zéro successives sont faites sur les horloges $y, x, y, x, y, y, x, y, x$, et enfin y.

6.2 Exécutions régulières

À cause de ces alternances, il n'est pas possible de compresser les exécutions accédant à l'état q de manière efficace. Pourtant, dans ce cas très précis, il est facile de décider pour n et p donnés, si q est accessible, même si on change la garde de la transition de p vers q en n'importe quelle garde, cela dépend juste de $\text{pgcd}(p,q)$. De plus les exécutions possèdent des régularités, qui permettent de les réécrire simplement : notons t_1 la transition qui remet x à zéro, t_2 celle qui remet y à zéro, et t_3 la transition de p vers q. La plus courte exécution utilise successivement les transitions $t_2, t_1, t_2, t_1, t_2, t_2, t_1, t_2, t_1$, ce que l'on peut réécrire $(t_2, t_1)^2 t_2 (t_2, t_1)^2$. Pour $n = 11$ et $p = 7$, cela donne $(t_2 t_1 t_2 t_2 t_1)^3 t_2$. De manière plus générale, pour n'importe quel couple (n,p), on peut ainsi écrire

une suite de transition menant à q avec une expression utilisant des puissances en une taille polynomiale en le codage de n et p.
On pourrait espérer généraliser ce résultat à n'importe quel automate à deux horloges : trouver un petit nombre de cycles de transitions de l'automate, afin de raccourcir l'exécution en accélérant ces cycles, c'est-à-dire de calculer l'effet sur les horloges d'un certain nombre d'itérations de ces cycles, ce qui permettrait de ne coder dans l'exécution qu'une seule fois le cycle avec le nombre de fois que ce cycle est utilisé successivement. S'il est tout à fait possible de faire cette transformation, il existe cependant des automates pour lesquels l'accélération n'est pas assez efficace pour obtenir une exécution avec accélération de taille polynomiale.
Étant donné un paramètre $n \in \mathbb{N}^*$, on construit un automate TM_n qui reconnait le préfixe du mot de Thue-Morse de longueur 2^{n-1}. Le mot de Thue-Morse est un mot w infini sur l'alphabet $\{a, b\}$ défini par $f^\omega(b)$ où $f : \{a, b\}^* \to \{a, b\}^*$ est le morphisme défini par $f(a) = ab$ et $f(b) = ba$. Le préfixe de w de longueur 2^n est $f^n(b)$

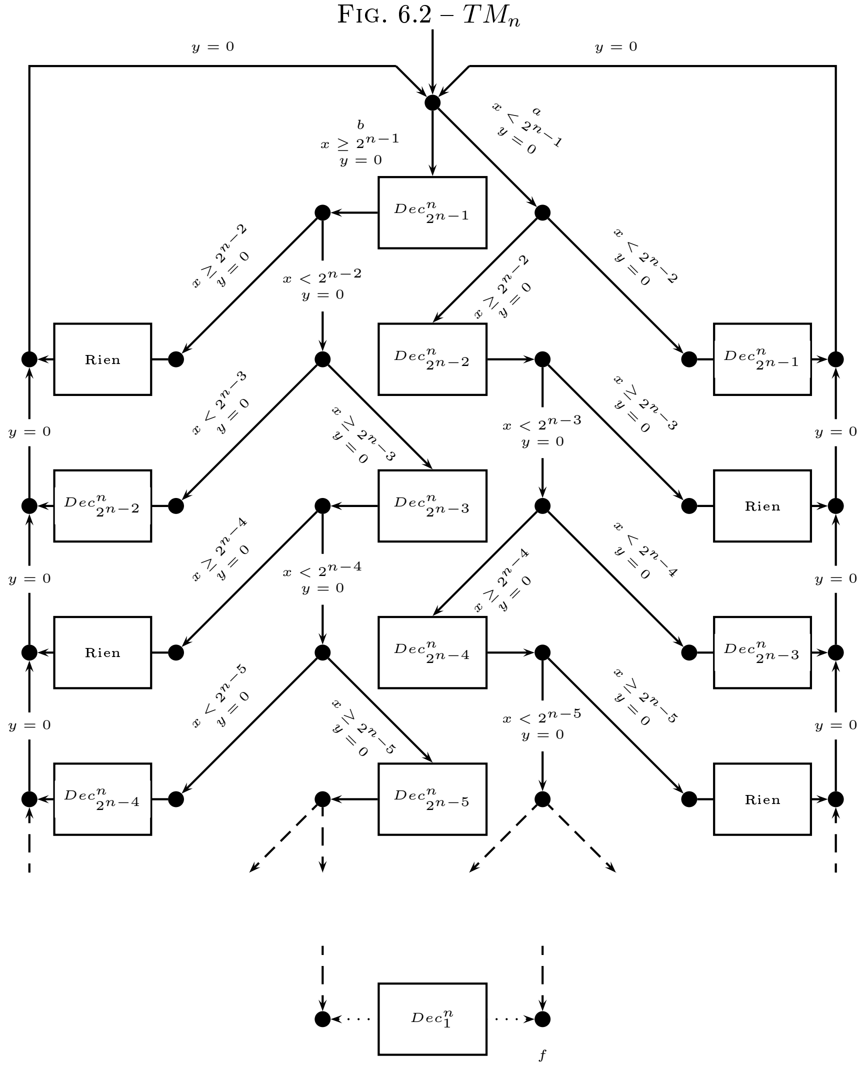
Exemple :

$$\begin{aligned} f(b) &= ba \\ f^2(b) &= baab \\ f^3(b) &= baababba \\ f^4(b) &= baababbaabbabaab \\ f^5(b) &= baababbaabbabaababbabaabbaababba \end{aligned}$$

L'une des propriété importante du mot de Thue-Morse est qu'il ne possède pas de cube : il n'existe pas de mot $v \in \{a, b\}^*$ tel que v^3 est un facteur de w (voir par exemple [11]). TM_n possèdera une unique arête d'étiquette a et une autre unique arête d'étiquette b. Toute exécution atteignant un état final devra donc passer par ces deux arêtes dans un ordre respectant le mot de Thue-Morse, et cette exécution ne pourra pas être accélérée de manière efficace à cause de cette propriété du mot w.

L'automate de la figure 6.2 possède deux horloges, x et y. y est une horloge de travail, valant généralement 0 sauf dans les gadgets, et permettant d'empêcher le temps de s'écouler. Il possède un unique état final noté f. Les gagdets nommés *Rien* n'ont aucun effet sur les horloges, on peut les remplacer par des transitions de garde $y = 0$. L'horloge x code une suite de n lettres a ou b par un entier binaire, obtenu en remplaçant a par 0 et b par 1. Donc x est borné par 2^n. Le mot de Thue-Morse $w = w_1 \ldots w_n \ldots$ possède la propriété suivante : pour tout $n \in \mathbb{N}$, $w_{2n-1} = w_n \neq w_{2n}$, donc pour connaître w_i, il suffit de connaître $w_{\lceil \frac{i}{2} \rceil}$. Imaginons qu'on veuille calculer w_{n+1} sachant que l'on connait $w_{\lceil \frac{n}{2} \rceil}$ et w_n sans connaitre précisement n. Remarquons que $n+1$ est pair si et seulement si $w_{\lceil \frac{n}{2} \rceil} = w_n$. Si $n + 1$ est pair, $w_{n+1} \neq w_{\lceil \frac{n}{2} \rceil}$ donc on connait w_{n+1} puisque l'alphabet ne comporte que deux lettres. Si $n + 1$ est impair, il faut connaître $w_{\lceil \frac{n}{2} \rceil+1}$ pour connaître la valeur de w_{n+1}. Si donc on connait les valeurs de

Fig. 6.2 – TM_n



y = 0
y = 0
a
x < 2^{n-1}
y = 0
b
x ≥ 2^{n-1}
y = 0
Dec^n_{2^n-1}
x ≥ 2^{n-2}
y = 0
x < 2^{n-2}
y = 0
x ≥ 2^{n-2}
y = 0
x < 2^{n-2}
y = 0
Rien
Dec^n_{2^n-2}
Dec^n_{2^n-1}
Dec^n_{2^n-2}
Dec^n_{2^n-3}
Rien
Rien
Dec^n_{2^n-4}
Dec^n_{2^n-3}
Dec^n_{2^n-4}
Dec^n_{2^n-5}
Rien
Dec^n_1
f

TAB. 6.1 – Calcul de Thue-Morse

n	$w_{\lceil n \rceil}$	$w_{\lceil \frac{n}{2} \rceil}$	$w_{\lceil \frac{n}{4} \rceil}$	$w_{\lceil \frac{n}{8} \rceil}$
1	$\boldsymbol{a}$	a	a	a
2	$\boldsymbol{b}$	$\boldsymbol{a}$	a	a
3	$\boldsymbol{b}$	b	a	a
4	$\boldsymbol{a}$	$\boldsymbol{b}$	$\boldsymbol{a}$	a
5	$\boldsymbol{b}$	b	b	a
6	$\boldsymbol{a}$	$\boldsymbol{b}$	b	a
7	$\boldsymbol{a}$	a	b	a
8	$\boldsymbol{b}$	$\boldsymbol{a}$	$\boldsymbol{b}$	$\boldsymbol{a}$

$w_n, w_{\lceil \frac{n}{2} \rceil}, w_{\lceil \frac{n}{4} \rceil}, \dots w_1$, on peut itérer le raisonnement jusqu'à obtenir que $\lceil \frac{n}{2^i} \rceil +$ 1 soit pair, ce qui permet de calculer les valeurs de $w_{n+1}, w_{\lceil \frac{n+1}{2} \rceil}, w_{\lceil \frac{n+1}{4} \rceil}, \dots w_1$. Plus précisément, sans faire de démonstration, regardons comment évoluent les valeurs prises par cette suite lorsqu'on incrémente n (on prend $w_0 = a$ par convention), tableau 6.1. D'après les remarques précédentes, il suffit de chercher dans chaque suite le plus long préfixe de cette suite avec une stricte alternance de a et b, par exemple *ababa* ou *babababab*. En effet, si $w_{\lceil \frac{n}{2^i} \rceil} = w_{\lceil \frac{n}{2^{i+1}} \rceil}$, alors $\lceil \frac{n}{2^i} \rceil$ est impair, et si on prend i minimum, alors pour tout $j < i$, $\lceil \frac{n}{2^j} \rceil$ est donc pair, et donc $\lceil \frac{n}{2^j} \rceil \neq \lceil \frac{n+1}{2^j} \rceil$. Il faut donc recalculer tout ce préfixe, et comme pour tout $j \leq i$, $\lceil \frac{n+1}{2^j} \rceil = \lceil \frac{n}{2^j} \rceil + 1$, on a $w_{n+1} = w_{\lceil \frac{n+1}{2} \rceil} = \dots w_{\lceil \frac{n+1}{2^i} \rceil} \neq w_{\lceil \frac{n}{2^i} \rceil}$. Une fois calculé le préfixe, il suffit donc de remplacer chaque lettre de ce préfixe par a si le préfixe fini par b, et par b si le préfixe fini par a (en fait on peut même remarquer que la longueur de ce préfixe est le successeur de la valuation 2-adique de n, *i.e.* le nombre de 2 dans la décomposition en nombres premiers de n plus 1). L'automate proposé implémente cette idée, en codant sur l'horloge x la suite $w_n, w_{\lceil \frac{n}{2} \rceil}, w_{\lceil \frac{n}{4} \rceil}, \dots w_1$, le bit correspondant à w_n en tête. La partie centrale détecte le plus long préfixe alternant, en remplaçant tous les bits par 0, *i.e.* par a. Lorsque l'alternance est finie, ou bien il fallait remplacer le préfixe par des a, on prend donc un gagdet *Rien*, ou bien il fallait remplacer par des b, ce qui peut alors être fait par un simple décrément (puisque le décrément est fait modulo 2^n). Le calcul s'achève lorsque toute la suite est alternante, ce qui arrive à la 2^{i-1}eétape, d'après la remarque sur les valuations 2-adiques. Enfin, à chaque étape, la première opération consiste à déterminer w_n, on peut donc reconnaître à ce moment la lettre a ou b correspondante.

Chapitre 7

Synthèse de contrôleurs dans les automates temporisés à deux horloges

On s'intéresse maintenant à un problème de synthèse de contrôleurs. On étudie une système modélisable par un automate temporisé à deux horloges. Le système subit un certain nombre d'événements (qui correspondent aux lettres de l'alphabet de notre automate). Un certain nombre de ces évènements sont contrôlables (actions du contrôleur), les autres sont incontrôlables (actions de l'environnement). Le problème de la synthèse de contrôleurs est de provoquer les actions contrôlables pour forcer le comportement du système, malgré les événements contrôlables. Ce problème peut aussi être vu comme un problème de jeu à deux joueurs, l'un étant le contrôleur et l'autre l'environnement.

7.1 Définitions

Soient Σ_c et Σ_e deux alphabets, le premier est l'alphabet des actions du contrôleur, le second celui de l'environnement. On note Σ_c^ϵ et Σ_e^ϵ les alphabets $\Sigma_c \uplus \{\epsilon\}$ et $\Sigma_e \uplus \{\epsilon\}$ respectivement . Plutôt que de travailler avec un unique alphabet $\Sigma = \Sigma_c \uplus \Sigma_e$, on va choisir de prendre comme alphabet des actions de l'automate $\Sigma = \Sigma_c^\epsilon \times \Sigma_e^\epsilon$. Ainsi, à tout instant, le contrôleur et l'environnement choisissent une de leurs actions, ou bien choisissent de ne rien faire (ϵ). Si les deux choisissent ϵ, l'automate prend une transition de délai jusqu'à ce que l'un des deux fasse une autre action. Sinon, le système prend une transition d'action correspondant à la lettre de Σ qui est le couple des choix du contrôleur et de l'environnement.

Définition 7.1.1
Un jeu temporisé concurrent est un automate temporisé $\mathcal{A} = (Q, q_0, X, \Sigma_c^\epsilon \times \Sigma_e^\epsilon, \delta, I, E)$, dont l'alphabet est défini comme le produit des actions du contrô-

leur, aussi appelé joueur existentiel, et de l'environnement, aussi appelé joueur universel. De plus, aucune transition n'a pour étiquette (ϵ, ϵ). On appelle E la condition de gain du contrôleur. Une partie de ce jeu est une exécution ρ de l'automate $\mathcal{A}$, cette partie est gagnante pour le contrôleur si $E(\pi(\rho)) = \top$ sinon elle est perdante (π est la projection triviale des configurations sur les états de $\mathcal{A}$).

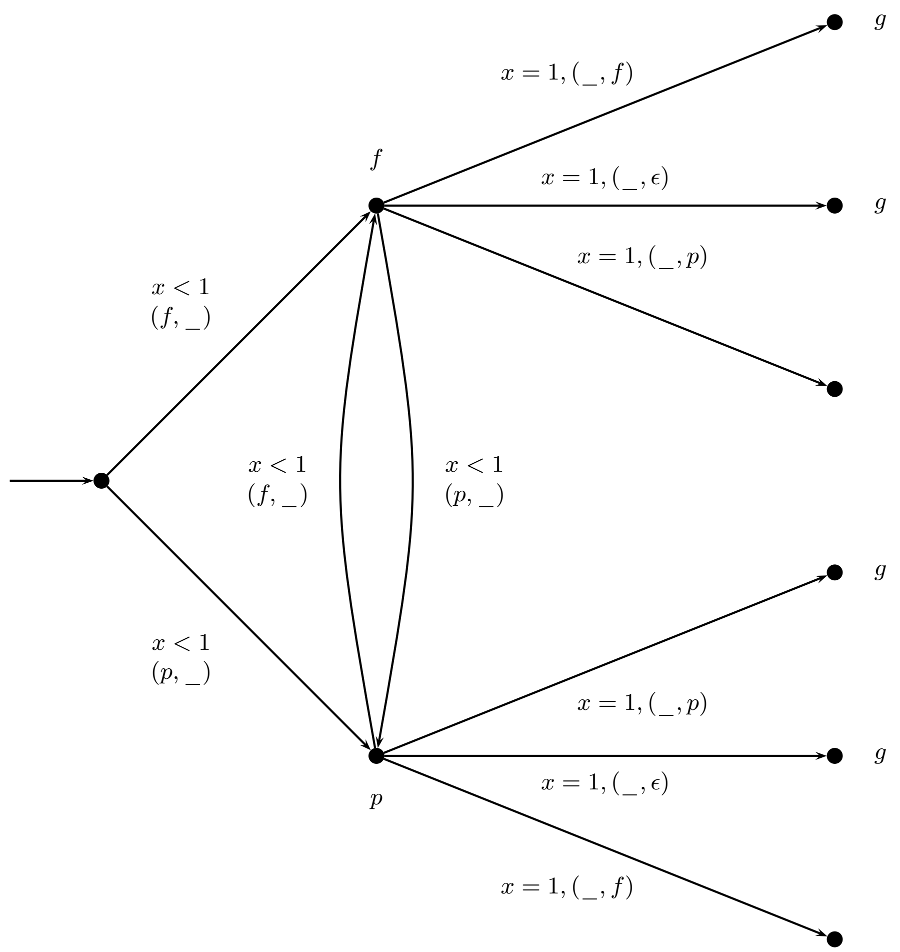
On n'impose pas d'avoir pour tout état et pour tout choix d'actions une transition sortant de cet état avec ce couple d'action. Si le contrôleur et l'environnement font un choix qui ne provoque pas de transitions, on considère que le système reste dans le même état. Une autre possibilité serait d'interdire à chacun des joueurs de choisir une action qui n'étiquette aucune transition sortant de l'état actuel. Les deux représentations sont équivalentes.

Définition 7.1.2
Une stratégie gagnante pour le contrôleur d'un jeu $\mathcal{A}$ est une fonction φ des exécutions finies de $\mathcal{A}$ dans Σ_c^ϵ qui vérifie la condition suivante : Pour toute exécution ρ de $\mathcal{A}$ vérifiant les deux conditions suivantes, $E(\pi(\rho)) = \top$:

- pour tout découpage $\rho = \rho', (p, u) \xrightarrow{(a,b)} (q, v), \rho''$ où $a \in \Sigma_c^\epsilon$, $\varphi(\rho', (p, u)) = a$
- pour tout découpage $\rho = \rho', (p, u) \xrightarrow{\epsilon(t)} (p, u + t), \rho''$, pour tout $t' \in [0, t[$, $\varphi(\rho', (p, u), (p, u + t')) = \epsilon$.

Autrement dit, une stratégie gagnante donne la manière de jouer du contrôleur pour que l'exécution vérifie la condition de gain E. Si le contrôleur suit cette stratégie, il est sûr de gagner. Le problème de la synthèse de contrôleur est de décider s'il existe une stratégie gagnante, et s'il en existe une, la calculer. Comme dans le cas de la condition d'arrêt d'une exécution d'un automate temporisé, on choisit généralement une condition de victoire simple. Nous nous concentrerons sur la plus simple, la condition d'accessibilité : le contrôleur gagne la partie si l'exécution passe par un état de l'ensemble des états finaux, et la condition duale : le contrôleur gagne la partie si l'exécution ne passe par aucun des états finaux, qu'on appelle jeu de sureté. Il existe beaucoup d'autres conditions étudiées dans la littérature, on peut par exemple définir une formule dans une logique modale, et donner comme condition de gain que l'exécution soit un modèle de la formule. Il est connu que le problème de la synthèse de contôleur est EXPTIME dans le cas général [9]. Nous allons montrer que dans le cas très simple de l'accessibilité dans les automates temporisés à 2 horloges, le problème est déjà EXPTIME-complet.

Exemple :



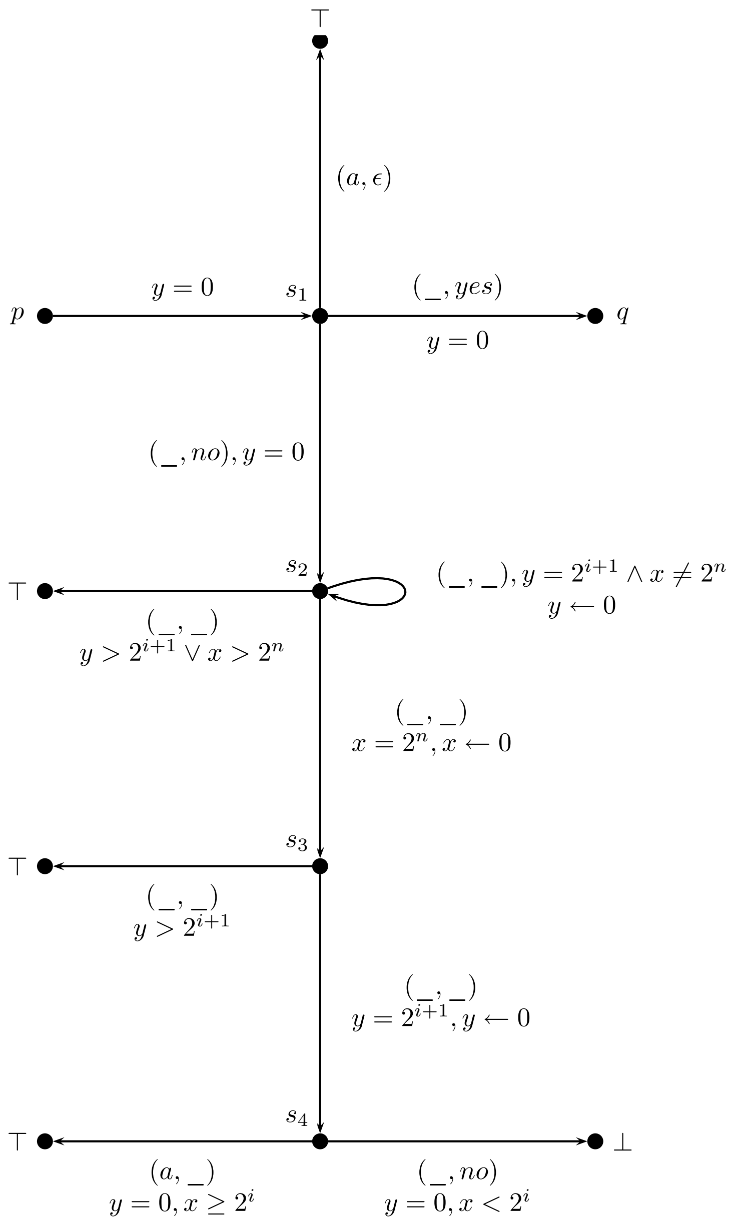
Un jeu de pile ou face. Le contrôleur choisit « pile » ou « face » et peut changé d'avis tant que la pièce (l'environnement) n'est pas retombée, ce qui arrive lorsque $x = 1$. Le contrôleur gagne s'il atteint un des états g (il a fait le bon choix ou bien l'environnement n'a pas choisi de côté). '_' dénote n'importe quelle action permise. Le contrôleur n'a pas de stratégie gagnante, l'environnement en a une : jouer p si la configuration est $(f, 1)$, f si c'est $(p, 1)$, ϵ sinon.

7.2 EXPTIME-difficulté

Proposition 7.2.1
Décider s'il existe une stratégie gagnante pour le contrôleur dans un jeu temporisé à deux horloges est EXPTIME-difficile.

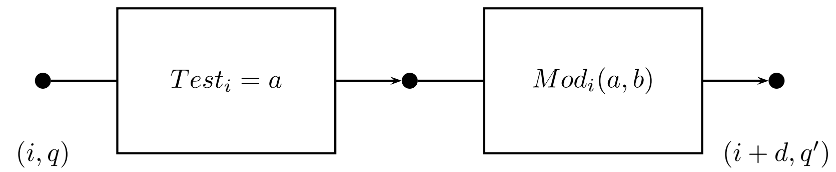
Preuve
On réduit au problème de l'acceptance dans les machines de Turing alternantes en espace polynomial (LBATM), c'est-à-dire les machines de Turing alternantes qui travaillent sur un ruban de la même longueur que l'entrée. Il est connu que APSPACE = EXPTIME, donc cela prouvera le rśultat cherché.
Soit $M = (Q, q_0, \Sigma = \{0,1\}, \delta, g)$ une LABTM, $\delta \subseteq Q \times \Sigma \times \Delta \times \Sigma \times Q$ est la fonction de transition, avec $\Delta = \{\text{left}, \text{right}\}$, et $g : Q \to \{\wedge, \vee, \top\}$. Un état d'étiquette $\top$ est un état de succès. Le problème de l'acceptance est de déterminer pour un mot w s'il existe une exécution de M acceptant w. On rappelle qu'une exécution de M est un arbre fini de configurations, dont chaque branche correspond à une transition valide de la machine de Turing depuis la configuration du père vers la configuration du fils de cette branche, la racine étant la configuration initiale, tel que tout nœud dont l'état a pour étiquette $\vee$ a un seul fils, tout nœud dont l'état a pour étiquette $\wedge$ a pour fils toutes les configurations accessibles depuis la configuration de ce nœud en une transition, et toutes les feuilles sont d'étiquette $\top$. Pour toute transition $(q, a, d, b, q') \in \delta$, on note $q \xrightarrow{a,d,b} q'$. Les configurations de M sont des couples (s, i) où s est un état de M et i la position de la tête de lecture sur le ruban, $i \in [\![0, n-1]\!]$. Soit $w \in \{0,1\}^n$ une entrée de la machine M.
Le codage va reposer sur les mêmes principes que les preuves de PSPACE-complexité de la section 3.3. L'automate possède comme ensemble d'états l'ensemble des couples $(i, q) \in [\![1, n]\!] \times Q$. L'horloge x contiendra la valeur du ruban $v(x) = \sum_{i \in [\![0,n-1]\!]} w_i 2^i$, et y servira pour accomplir les opérations diverses, notamment l'addition et la soustraction. Avec deux horloges, il ne semblait pas possible de faire un gadget de test d'un bit de x, on va pouvoir l'accomplir. On note $\top$ les états gagnants pour le contrôleur, et $\perp$ les états perdants du contrôleur. On utilisera une condition de victoire de type accessibilité pour le contrôleur. Les états $\top$ correspondent donc aux états que le contrôleur doit atteindre pour gagner, les états $\perp$ sont des puits, donc si la partie atteint une configuration dont l'état est un de ces puits, la partie est perdue pour le contrôleur. Tous les états (i, q) avec $g(q) = \top$ sont considérés comme des états $\top$. On note que dans ce cas particulier des jeux d'accessibilités sur les automates temporisés, il existe une stratégie gagnante si et seulement si il existe une stratégie gagnante sans mémoire, c'est-à-dire ne dépendant que de la configuration actuelle.
Le gadget de test d'un bit est présenté sur la figure 7.1. L'entrée du gadget est l'état p et sa sortie est q. On montre que depuis la configuration (p, x_0, y_0), si le bit de poids i dans x est 1, le contrôleur a une stratégie permettant d'atteindre q ou de gagner, et si ce bit est 0, l'environnement a une stratégie gagnante.

FIG. 7.1 – Test d'un bit dans un jeu temporisé



Premier cas, si le bit est 1, dans l'état s_1, le contrôleur choisit l'action a, qui lui permet de gagner si le controlêur ne choisit pas yes ou no. S'il choisit yes, l'état q est atteint. Sinon, on arrive dans l'état s_2. Si l'environnement ne force pas les transitions possibles dans les états s_2 et s_3, le contrôleur peut atteindre les états $\top$ sur la gauche et gagner. On suppose donc que l'environnement force les transitions, on arrive alors dans l'état s_4 avec dans l'horloge x la valeur de x_0 modulo 2^{i+1}. Il suffit alors au contrôleur de jouer a pour gagner. Si le bit de poids i est 0, dans s_1, l'environnement joue no, puis force les transitions jusqu'à s_4, puis l'environnement joue no, et la partie est perdu pour le contrôleur. Ce gadget permettant de tester si le bit de poids i est 1 est donc correct. Pour tester si un bit vaut 0, il suffit de permuter les gardes $x \geq 2^i$ et $x < 2^i$ des deux transitions du bas, la preuve de correction est similaire. Ce gadget correspond donc à faire prouver par le contrôleur que le bit de poids i à la bonne valeur si l'environnement le souhaite. Il est aisé de le modifier pour que ce soit l'environnement qui doivent le prouver si le contrôleur demande une preuve en permutant les rôles.
On peut facilement réécrire les différents gadgets, additionneur et soustracteur, en ajoutant des transitions qui forcent l'environnement à prendre les transitions dès que possible. On construit donc maintenant l'automate temporisé pour lequel il existe une statégie gagnante pour le contrôleur si et seulement si la machine de Turing alternante M accepte le mot w de taille n. Soit $q, a \to q', b, d$ une transition de M, avec $a, b \in \{0,1\}$, $d \in \Delta$. On ajoute à l'automate le gadget suivant (lorsque l'état $(i+d, q')$ existe) :

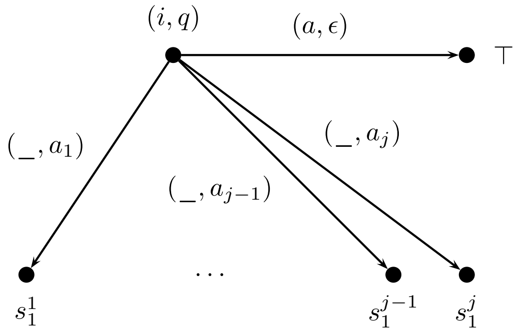
FIG. 7.2 – Codage d'une transition d'une ATM



Dans cette figure, le gadget $Test_i = a$ correspond au gadget de la figure 7.1 si $a = 1$, ou bien à sa légère modification permettant de tester si un bit est 0 si $a = 0$. C'est le contrôleur qui doit prouver la valeur du bit si $g(q) = \vee$, et l'environnement sinon. Si $a = b$, $Mod_i(a, b)$ ne fait rien, si $a = 1, b = 0$, c'est le soustracteur pour $k = 2^i$, et si $a = 0, b = 1$, c'est l'additionneur pour $k = 2^i$. Ce module produit donc bien l'effet attendu.
On code maintenant l'alternance. Pour cela, on utilise la remarque classique suivante : les états $\wedge$ doivent n'avoir que des successeurs qui permettent de gagner (puisque les configurations gagnantes correspondent aux configurations

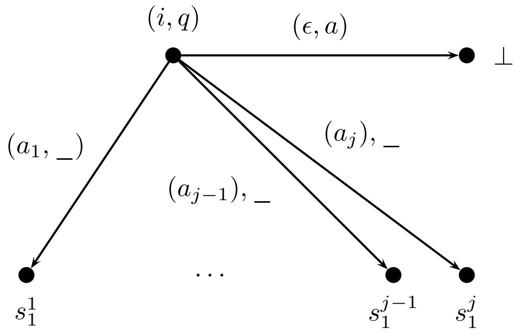
de M qui atteignent $\top$), donc il n'y a une stratégie gagnante que si depuis toutes les configurations successeurs il y a une stratégie gagnante. Donc, s'il n'existe pas de stratégie pour le contrôleur, il suffit à l'environnement de choisir un successeur pour lequel le contrôleur n'a pas de stratégie gagnante. Inversement, dans les états $\vee$, il suffit qu'il y ait un seul successeur qui atteignent $\top$, donc on laisse le contrôleur choisir l'état suivant.
Soit q une état tel que $g(q) = \wedge$, soient $i \in [\![1, n]\!]$, $t_1, \ldots, t_j$ les transitions sortantes de (i, q). Chacune de ces transitions est une transition correspondant à la transition $p \to s_1$ dans le gadget de la figure 7.1. On leur rajoute des étiquettes selon le schéma suivant, avec $a_1, \ldots a_j$ des lettres distinctes deux-à-deux de Σ_e :

FIG. 7.3 – Codage d'un état $\wedge$



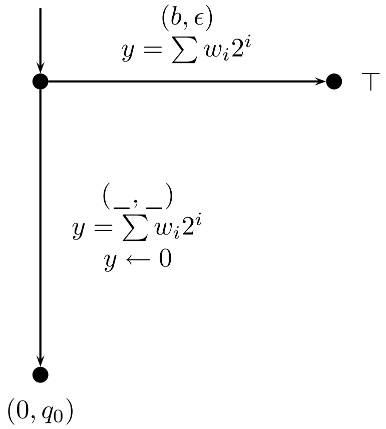
Si maintenant $g(q) = \vee$, on utilise plutôt le schéma suivant, avec $a_1, \ldots a_j$ des lettres distinctes deux-à-deux de Σ_c :

FIG. 7.4 – Codage d'un état $\vee$



Pour finir la construction, on rajoute la partie permettant d'initialiser l'automate en l'amenant à la configuration codant la configuration initiale de la machine de Turing alternante M. Pour cela, on ajoute un état initial de la façon suivante, avec $w = w_0 \dots w_{n-1}$:

FIG. 7.5 – Initialisation du calcul



Enfin, puisqu'on voudrait éviter les comportements Zeno, c'est-à-dire les exécutions infinies dont la somme des délais ne diverge pas, on ajoute dans le schémas de la figure 7.2 entre les deux boites un soustracteur qui soustrait 2^n, et qui prend un temps 2^n. Ainsi, tous les cycles de configurations ont un délai d'au moins 2^n, ce qui empêche les comportements Zeno.
La construction est maintenant achevée, on prouve que cette construction est correcte. Supposons qu'il existe une exécution ρ de M acceptante. On montre que la stratégie positionnelle suivante est gagnante : le contrôleur force les actions dans les états où il est censé le faire (états $\wedge$, s_1, s_2,s_3), et dans les états $\vee$, il choisit d'aller dans un successeur appartenant à ρ si possible. Enfin, si l'environnement décide de prendre une transition non-valide depuis un état $\wedge$ (la valeur du bit à tester n'est pas celle attendue), alors le contrôleur demande une preuve de valeur du bit, donc gagne. Cette stratégie est gagnante, puisque soit l'environnement « triche », et donc est pénalisé par construction, donc l'exécution atteint un état $\top$, soit l'environnement ne « triche » pas, donc la suite de configuration correspond à une suite de configuration de ρ, donc atteint aussi un état $\top$.
Supposons maintenant qu'il n'existe pas d'exécution acceptante dans M. On fait jouer l'environnement de la manière suivante : l'environnement empêche le contrôleur de « tricher », par forçage ou par demande de validation du test d'un bit s'il est erroné, et choisit dans les états $\wedge$ une configuration dont l'équivalent c dans M vérifie qu'il n'existe pas d'exécution acceptante depuis c si possible. Alors, depuis la configuration initiale, il n'est pas possible pour le contrôleur

d'atteindre une configuration depuis laquelle il existe une exécution acceptante. En effet, ce n'est pas le cas de la configuration initiale. Dans les états $\vee$, le contrôleur ne peut pas atteindre une telle configuration s'il n'est pas déjà dans une configuration de cette sorte, puisque sinon cela contredirait la non-existence d'une exécution acceptante depuis la configuration dans l'état $\vee$. Enfin, par un raisonnement semblable, depuis une configuration avec un état $\wedge$ atteint, il existe une configuration c qui vérifie qu'il n'existe pas d'exécution acceptante depuis c, et cet état est choisi par l'environnement pour évoluer. Donc toute exécution dans l'automate temporisé depuis la configuration initiale n'atteint jamais d'état $\top$. Puisque l'environnement a une stratégie gagnante, le contrôleur n'en a pas. $\blacksquare$

Chapitre 8

Conclusion

Malgré nos efforts, en particulier la résolution d'un cas particulier, le problème initial reste ouvert. Cependant, nous avons pu remarquer que si ce problème n'est peut-être pas PSPACE-complet, de très légères modifications le rendent PSPACE-complet. De plus, nous avons mis au point une technique, permettant de supprimer les transitions qui ne remettent pas d'horloges à zéro, dont on espère qu'elle sera la première étape d'un algorithme plus efficace que celui des régions. Notamment, grâce à cette technique, il est possible de transformer l'automate de telle manière que pour chaque état, il existe une horloge telle que toute transition rentrant dans cet état remette cette horloge à zéro. Nous espérons qu'il soit alors possible d'écrire de manière concise les dates d'entrée possible dans chaque état, ce qui permettrait de fait de calculer l'ensemble des configurations accessibles. Nous aboutirions alors éventuellement à un algorithme de complexité Σ_2.

Bibliographie

[1] L. ACETO, 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 (2002), pp. 7-51.

[2] R. ALUR et D. DILL, *A theory of timed automata*, Theoretical Computer Science, 126 (1994), pp. 183-235

[3] B.BÉRARD, M. BIDOIT, A. FINKEL, F. LAROUSSINIE, A. PETIT, L. PETRUCCI. P. SCHNOEBELEN, *Systems and Software Verification, Model-Checking Techniques and Tools*, Springer, (2001).

[4] S. CAMPOS, M. TEIXEIRA, M. MINEA, A. KUEHLMANN, E. M. CLARKE, *Model Checking Semi-Continuous Time Models Using BDDs*, Electr. Notes Theor. Comput. Sci. 23(2), (1999).

[5] C. CHOFFRUT, M. GOLDWURM, *Timed automata with periodic clock constraints*, Journal of Automata, Languages and Combinatorics, 5, (2000), pp. 371-404.

[6] E.M. CLARKE, O. GRUMBERG et D. PELED, *Model Checking*, MIT Press, (1999).

[7] C. COURCOUBETIS et M. YANNAKAKIS, *Minimum and maximum delay problems in real-time systems*, Formal Methods in System Design (1992), pp. 385-415.

[8] M.R. GAREY, D.S. JOHNSON, *Computers and intractability, a Guide to the Theory of NPcompleteness*, Freeman, (1979).

[9] T.A. HENZINGER, P.W. KOPKE, *Discrete-Time Control for Rectangular Hybrid Automata*, Theoretical Computer Science 221 (1999), pp. 369-392.

[10] F. LAROUSSINIE, N. MARKEY, Ph. SCHNOEBELEN, *Model Checking Timed Automata with One or Two Clocks*, Lecture Notes in Computer Science 3170 (2004), pp. 387-401.

[11] M. LOTHAIRE, *Combinatorics on Word*, Cambridge University Press, (1982).

[12] A. PNUELI, *The Temporal Logic of Programs*, Proceedings of the 18th IEEE Symposium Foundations of Computer Science (FOCS 1977), (1977), pp. 46-57.