Une extension d'un résultat d'indécidabilité pour les automates temporisés.

Catherine Dufourd [a]

[a]ENS de Cachan, Laboratoire Spécification et Vérification, CNRS URA 2236, 61 avenue du Président Wilson, 94235 Cachan Cedex, France. C.Dufourd@lsv.ens-cachan.fr

Résumé
Lorsqu'on ajoute la primitive d'addition dans les automates temporisés, il devient impossible de décider si le language reconnu est vide [2]. Dans cet article, nous étendons ce résultat au cas limite où seules deux horloges sont impliquées dans les contraintes avec addition.

Mots-clés : Automates temporisés; Indécidabilité; Vide du langage; Systèmes temps-réels.

1. Automates temporisés

Les automates temporisés, dont la classe a été définie en 1990 par Alur et Dill [1], constituent l'un des modèles les plus étudiés pour les systèmes temps réels. L'objectif est de vérifier si une modélisation répond à certaines spécifications ou possède certaines propriétés quantitatives liées au temps. L'idée est d'ajouter à un automate classique une notion explicite de temps afin de dater les actions du système. Pour ce faire, on utilise un ensemble d'horloges, ou plutôt de chronomètres, évoluant naturellement avec le temps mais testées et éventuellement remises à zéro lors d'une action du système. Les tests consistent en la comparaison d'une horloge avec une constante (comparaison notée: $x \# C$) ou de la différence de deux horloges avec une constante (notée: $x - y \# C$). La principale qualité du modèle est la décidabilité du problème de la vacuité du langage. Ce problème, résolu par le *graphe des régions*, est **PSpace**-complet.

Un automate temporisé est un tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F, X)$ où Q est un ensemble d'états, Σ un alphabet fini, δ une fonction de transition, q_0 un état initial, $F \subseteq Q$ un ensemble fini d'états finaux, X un ensemble fini d'horloges. Une *contrainte* est une combinaison booléenne de propositions atomiques de la forme: $(x \# C)$ ou $(x - y \# C)$, pour x et $y \in X$, $C \in \mathbb{N}$ et $\# \in \{<, =, >\}$. Une transition dans δ d'un état q vers un état q' est étiquetée par un triplet: $< a, T, R >$ où $a \in \Sigma$, T est une contrainte et $R \subseteq X$. Initialement, l'automate est dans l'état q_0 et toutes les valeurs des horloges sont nulles. Ces valeurs augmentent de manière synchrone avec le temps. Si l'automate est dans l'état q, la transition $(q, < a, T, R >, q')$ est franchissable à la date $t \in \mathbb{R}^+$ si les valeurs des horloges de X vérifient T; si la transition est franchie, les horloges de R sont remises à zéro, et l'automate passe dans l'état q'. Un mot temporisé est une suite $(a_1, t_1), \cdots, (a_n, t_n) \in (\Sigma \times \mathbb{R})^*$ où la suite des t_i est croissante. Un tel mot est *reconnu* par l'automate s'il étiquette une suite de transitions menant à l'état final.

L'article [2] considère l'inclusion de la primitive d'addition avec des tests de type $(x+y \# x'+y')$ et prouve qu'un modèle plus puissant est obtenu : le problème de vacuité du langage devient indécidable. Il est facile de constater que pour le modèle des automates temporisés avec au plus 2 horloges et des contraintes de la forme $(x \# C)$, $(x-y \# C)$ et $(x+y \# C)$, la vacuité du langage est décidable. Cette classe particulière contient cependant des langages non reconnaissables par la classe des automates temporisés classiques. Dans cet article, nous montrons que le test du vide du langage est indécidable pour les automates temporisés avec au plus 6 horloges, avec des contraintes de la forme $(x \# C)$ et $(x + y = 1)$ où l'unique contrainte additive implique toujours les mêmes deux horloges. Ainsi, d'une part nous étendons le résultat général d'indécidabilité et d'autre part nous donnons une borne supérieure sur le nombre d'horloges pour le résultat de décidabilité.

La preuve consiste à simuler une machine déterministe à deux compteurs. Une telle machine exécute un programme, dont chaque instruction est étiquetée, terminé par une instruction *Halte*. Toute autre instruction prend l'une des deux formes suivantes :

$$(1)\ e_i\ : a := a + 1; goto\ e_j; \qquad (2)\ e_i\ : Si\ a = 0\ goto\ e_j\ sinon\ a := a - 1;\ goto\ e_k;$$

où a désigne un des deux compteurs (c ou d). Sans perte de généralité, on suppose que les deux compteurs sont initialement à zéro. La terminaison d'un tel programme, c'est-à-dire l'accès à l'étiquette (ou l'état) *Halte*, est un problème indécidable [3].

2. Preuve d'indécidabilité

Théorème : Pour la classe des automates temporisés à au plus 6 horloges, augmentés de la primitive d'addition $(x + y = 1)$, où les horloges x et y sont fixées dès le départ, le problème de la vacuité du langage est indécidable.

Preuve : Nous réduisons le problème de la non-terminaison d'une machine déterministe à deux compteurs, c et d, à notre problème. Nous construisons un automate qui simule le comportement de la machine et atteint un état final si et seulement si la machine s'arrête. Supposons que le

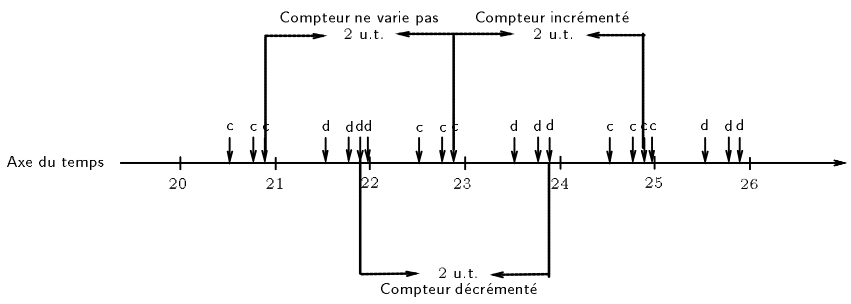

FIG. 1 –. Effet attendu sur la sous-séquence : (..., $< 3, 4 >$, $< 3, 3 >$, $< 4, 3 >$ ...).

comportement de la machine soit décrit par la séquence :

$$(< n_0, m_0 >, < n_1, m_1 >, \cdots, < n_i, m_i >, \cdots)$$

où n_i et m_i sont les valeurs des compteurs après la $i^{\text{ième}}$ instruction. Nous encodons chaque valeur dans un intervalle de temps unitaire et nous alternons sur l'axe du temps les valeurs de c et de d. Sur l'exemple de la figure 1, la configuration $< 3,4 >$ est codée par le facteur $(c,20+1/2)(c,20+3/4)(c,20+7/8)(d,21+1/2)(d,21+3/4)(d,21+7/8)(d,21+15/16)$. De façon générale, l'automate reconnaît n_0 symboles c durant l'intervalle de temps $[0..1]$, m_0 symboles d durant $[1..2]$, $\cdots$, n_i symboles c durant $[2i, 2i+1]$, m_i symboles d durant $[2i+1, 2(i+1)]$ etc. (outre les symboles, identifiés par $*$ sur la figure 2, correspondant à des transitions internes à la construction).

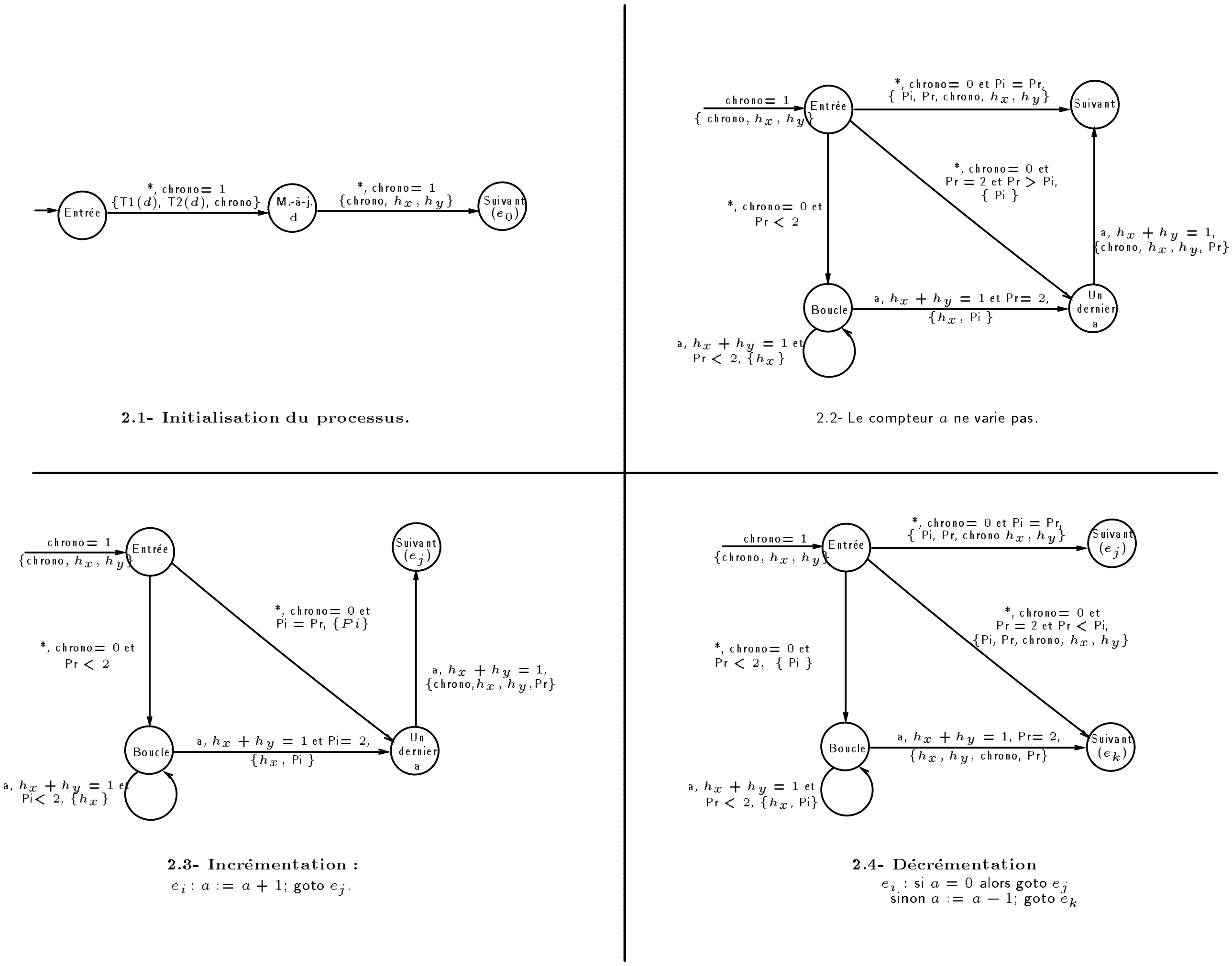

FIG. 2 –. Constructions génériques des sous-modules de l'automate.

Une horloge *chrono*, remise à zéro toutes les unités de temps entières, détermine les intervalles de longueur 1. À l'intérieur d'un tel intervalle, les dates des symboles sont précises. Elles sont calculées à l'aide de deux horloges, h_x et h_y, remises à zéro à chaque début d'intervalle. Les tests revêtent toujours la même forme : $h_x + h_y = 1$, $\{h_x\}$. Ainsi, pour k symboles, les dates calculées à partir de $h_x = h_y = 0$ sont : 0, 1/2, 3/4, 7/8, ... , $(2^k - 1)/2^k$. Chaque date marque la moitié du délai restant pour atteindre l'intervalle suivant. Sur un même intervalle, l'automate peut reconnaître un nombre illimité de symboles. Nous montrons à la figure 1 un exemple de l'effet attendu sur une sous-séquence arbitraire.

Hormis *chrono*, h_x et h_y, l'automate comprend 4 horloges. On associe à chaque compteur deux horloges de travail : *T1(c)* et *T2(c)* pour c; *T1(d)* et *T2(d)* pour d. Ces horloges assurent le bon compte des symboles sur chaque intervalle, en repérant d'une instruction à la suivante des délais constants de 2 unités. Les repères se font tantôt sur les derniers symboles d'un intervalle, tantôt sur les avant-derniers symboles (voir figure 1). L'horloge repérant la date du dernier symbole est dite *pilote*; l'autre est dite *prédécesseur*.

La figure 2 donne les constructions génériques des sous-modules de l'automate pour un compteur désigné par a et deux horloges de travail désignées par *Pi* (pour Pilote) et *Pr* (pour Prédécesseur). D'une instruction à la suivante, trois cas peuvent se présenter : le compteur n'est pas modifié (fig. 2.2), il est incrémenté (fig. 2.3) ou il est décrémenté (fig. 2.4). Les états de sortie des sous-modules, étiquetés par *Suivant*, correspondent soit au passage à l'instruction suivante, soit au passage à un module transitoire de type 2.2. À la fin d'un intervalle, les horloges *Pi* et *Pr* se suivent à un délai $1/2^k$ près, sauf lorsque la valeur du compteur est nulle. Cette situation apparaît soit en début de processus (fig. 2.1) puisque le compteur est nul au départ, soit lorsque le compteur est décrémenté alors que sa valeur est 1. Dans ce dernier cas, les deux horloges sont synchronisées sur la date du début d'intervalle (fig. 2.4 : arc de l'état *Entrée* vers l'état $Suivant(e_k)$). La synchronisation sert de repère lors du test à zéro (fig. 2.4 : arc de l'état *Entrée* vers l'état $Suivant(e_j)$).

Dans le cas d'une incrémentation, c'est le pilote qui guide la boucle de reconnaissance des symboles d'un intervalle (fig. 2.2). Dans le cas d'une décrémentation, c'est le prédécesseur qui guide la boucle (fig. 2.4); l'horloge pilote courante endosse alors le rôle du prédécesseur en prévision des instructions à venir. Les deux rôles sont alternativement tenus par les deux horloges. Dans l'automate il faut traiter ces deux cas de figure séparément et ceci entraîne la duplication de tous les modules (nous présentons les modules génériques). À des fins de simplification, nous échangeons les rôles systématiquement d'une instruction à la suivante (c'est pour cette raison que *Pr* guide la boucle du sous-module 2.2). Si la machine contient S états, l'automate associé contiendra en tout $4 * S$ sous-modules plus le sous-module de départ. La dernière étape consiste à relier ces modules pour simuler le comportement de la machine à deux compteurs et à ajouter un état final qui sera atteint losque l'instruction *Halte* sera rencontrée par la machine. En conséquence le langage reconnu par cet automate est vide si et seulement si la machine ne termine pas. Enfin, il est aisé de faire tenir à h_y le rôle de l'horloge *chrono*, ramenant ainsi à 6 le nombre total d'horloges □.

Bibliographie

1. R. Alur, D.L. Dill. Automata for modeling real-time systems. In *Proceedings of ICALP'90*, number 443 in Lecture Notes in Computer Science, pages 322-335. Springer-Verlag, 1990.
2. R. Alur, D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183-235, 1994.
3. M. Minski. Computation : Finite and Infinite Machines. Prentice Hall, 1967.