Laboratoire
Spécification
et
Vérification

CACHAN

Automates temporisés et modularité

Patricia BOUYER

Mémoire de DEA, juin 1998

Résumé

Les automates temporisés sont un modèle très utilisé en vérification. Lorsqu'on étudie des systèmes réels, ceux-ci sont souvent donnés sous la forme de plusieurs sous-systèmes, pouvant être composés entre eux. Nous allons étudier ici de nouvelles opérations de composition sur la classe des automates temporisés et les utiliser pour présenter un algorithme modulaire pour tester le vide d'un langage temporisé reconnaissable.

Remerciements

Mon stage de DEA, effectué au L.S.V. (E.N.S. de Cachan), a été supervisé par Antoine PETIT. Je tiens à le remercier pour tout le temps qu'il a passé à revoir mon travail.

Je tiens également à remercier François LAROUSSINIE pour m'avoir fait découvrir le domaine des automates temporisés et de la vérification ainsi que d'avoir répondu à toutes mes questions depuis plus d'un an.

Table des matières

Introduction

Actuellement, des logiciels mis sur le marché contiennent des erreurs. Ceci provient du fait qu'ils ne sont pas vérifiés. Il serait donc utile d'avoir des méthodes de vérification, si possible entièrement automatisées, afin de pouvoir valider les programmes. C'est précisément l'objectif que se fixe l'informatique de la vérification. En effet, celle-ci peut par exemple consister à se donner un système (un logiciel par exemple), une propriété que l'on voudrait voir vérifiée par le système et à décider si le système vérifie ou non la propriété. C'est dans ce contexte que se situe le présent mémoire.

Pour modéliser les systèmes (réactifs), un modèle très couramment utilisé est celui des systèmes finis de transitions (comme par exemple les automates d'états finis). Ce type de modèles permet d'étudier l'ordre relatif des actions, comme « *déclencher l'alarme après toute réception d'un problème* ». Mais il ne contient pas d'information quantitative sur le délai séparant deux actions consécutives. Ainsi il n'est pas possible d'utiliser ce type de formalisation pour exprimer des propriétés comme « *déclencher l'alarme moins de 5 secondes après toute réception d'un problème* ». C'est ce qui incite à étudier non plus des systèmes finis de transitions, mais des systèmes temporisés : dans ces derniers, chaque action est datée.

Dans ce mémoire, nous nous intéresserons à un modèle de système temporisé de base, les *automates temporisés*. Depuis son introduction par Alur et Dill dans [AD90], celui-ci a été étudié sous de multiples aspects, tant au niveau théorie des langages qu'au niveau modèle temporisé pour la spécification et la vérification. En effet, d'une part, des problèmes tels que la déterminisation ([AFH94]), la minimisation ([ACH+92]), le pouvoir d'expression des horloges ([ACH94], [HKW95]) et la caractérisation logique des langages temporisés ([LLW95], [Wil94]) ont été étudiés et d'autre part, ce modèle a été utilisé avec succès dans la spécification et la vérification de systèmes temporisés ([HNSY94]).
Les automates temporisés sont une extension des automates finis classiques : à chaque automate est associé un ensemble fini d'horloges, ces horloges étant des variables qui évoluent toutes à la même vitesse (celle d'une horloge universelle externe, sur laquelle on ne peut pas agir) et pouvant être remises à zéro. On s'autorise à mettre des contraintes sur les transitions de l'automate, c'est à dire que sur une transition, on peut rajouter une contrainte du type : « pour pouvoir effectuer cette transition, il faut que l'horloge x ait sa valeur comprise entre 2 et 3 ». Comme les automates finis, les automates temporisés décrivent des comportements qui, cette fois, sont temporisés, c'est à dire qu'on associe à chaque action une date, qui correspond à l'instant auquel elle a lieu. Nous rappellerons tous les résultats qui nous seront utiles par la suite au chapitre I.

Les automates temporisés sont très utilisés dans la modélisation de systèmes temporisés, et servent dans des algorithmes de model-checking : le *model-checking* désigne une technique de vérification qui repose sur un algorithme vérifiant qu'une propriété énoncée en logique temporelle est satisfaite par un automate (ou un réseau d'automates) modélisant le système à valider. De nombreux résultats concernant le model-checking pour les automates temporisés ont déjà été démontrés ; citons par exemple l'un des premiers résultats en la matière : le model-checking pour TCTL est décidable ([ACD93]). Un algorithme de model-checking totalement différent est présenté dans [ABL97] pour une logique modale. Différents autres résultats sont cités dans [AH93].

En outre, certains algorithmes de model-checking pour les automates temporisés ont été implémentés dans des outils comme CMC, HYTECH, KRONOS, UPPAAL ... Par exemple, UPPAAL, outil développé à l'université d'Aalborg (Danemark) et à celle d'Uppsala (Suède) (voir [BLL+96] pour plus d'informations), permet de vérifier des propriétés d'accessibilité : étant donné un ensemble d'automates temporisés mis en parallèle, UPPAAL répond à la question : peut-on atteindre telle configuration ? Si la réponse est positive, il donne une exécution permettant d'atteindre cette configuration.

Ces outils de vérification ont permis de détecter des erreurs au sein d'applications industrielles. Un exemple d'une telle application est l'erreur trouvée par le model-checker UPPAAL dans un protocole audio/vidéo de Bang & Olufsen. Depuis plusieurs années, il était connu que ce protocole contenait une erreur, mais il avait été impossible de la cerner jusqu'à présent. UPPAAL a permis de trouver cette erreur et, une fois celle-ci corrigée, a validé le nouveau protocole (cf [HSGL97]). La vérification de logiciels n'est donc pas seulement une jolie théorie utilisant les automates temporisés, mais aussi un domaine ayant des applications concrètes et décisives dans l'industrie.

Lorsqu'on étudie des systèmes réels, les modèles qui correspondent sont souvent très gros. Il est alors judicieux de les modéliser non plus par un gros automate, mais par un réseau d'automates plus petits (ceux-ci étant synchronisés entre eux). On parle alors de décomposition des automates en *modules* de taille plus petite. Les propriétés que l'on cherche à étudier (au niveau du gros système) doivent alors être liées aux propriétés des petits systèmes. D'autre part, il est à constater que, souvent les systèmes sont déjà donnés sous la forme de plusieurs petits sous-systèmes.

Dans une première étape, nous allons définir de nouvelles opérations naturelles de composition des automates temporisés (voir le chapitre II). Nous voudrions composer séquentiellement deux automates temporisés (comme le fait l'opération de concaténation dans le cas des automates finis classiques). La question que l'on se pose alors est la suivante : que doit-on faire des horloges ? Deux réponses viennent immédiatement à l'esprit : on peut soit remettre toutes les horloges à zéro avant de passer dans le deuxième automate, soit ne pas le faire (la valeur des horloges au début du deuxième est alors celle à la fin du premier automate). Ces deux manières de composer séquentiellement des automates temporisés sont certes les plus naturelles, mais nous verrons que certains systèmes réels nécessitent une définition plus fine. Pour C ensemble d'horloges, nous définissons alors l'opération $\cdot_C$ qui compose de manière séquentielle deux automates de la manière suivante : on met « bout à bout » les deux automates temporisés en mettant les horloges de C à zéro entre temps. Ceci signifie que, lorsqu'on rentre dans le deuxième automate, la valeur des horloges de C est zéro alors que la valeur des autres horloges est celle qu'elles avaient à la sortie du premier automate. Il est à remarquer que les opérations de concaténation définies dans [ACM97] et [Asa98] pour les automates temporisés ayant une seule horloge sont un cas particulier de ce qui précède. En effet, elles correspondent l'une à ${}^{\cdot}_{\emptyset}$, l'autre à ${}^{\cdot}_{x}$ où x est l'unique horloge de l'automate.

Nous étudions ensuite le comportement composé de $\mathcal{A} {}^{\cdot}_{C} \mathcal{B}$ si $\mathcal{A}$ et $\mathcal{B}$ sont des automates temporisés. Il est usuel de considérer le comportement d'un automate comme étant un langage temporisé correspondant à une initialisation des horloges à zéro. Avec ce point de vue classique, il n'est pas possible d'exprimer le comportement de $\mathcal{A} {}^{\cdot}_{C} \mathcal{B}$ en fonction de ceux de $\mathcal{A}$ et de $\mathcal{B}$. Nous proposons alors de voir un automate temporisé non plus comme un langage temporisé, mais comme une fonction qui à toute valeur initiale associe le langage accepté par l'automate lorsqu'on initialise ses horloges à cette valeur. Nous voyons alors que l'opération ${}^{\cdot}_{C}$ agit de la manière suivante : un mot temporisé sera dans le langage accepté par $\mathcal{A} {}^{\cdot}_{C} \mathcal{B}$ s'il se décompose en deux mots, l'un étant dans le langage accepté par $\mathcal{A}$, l'autre étant dans le langage accepté par $\mathcal{B}$ lorsqu'on initialise ses horloges (celles qui ne sont pas dans C) à la valeur qu'elles avaient à la sortie de l'automate $\mathcal{A}$. Ainsi, bien plus que du langage accepté par un automate temporisé, nous aurons besoin d'une part de la fonction précédemment citée et d'autre part d'une fonction qui indique les valeurs des horloges à la sortie de l'automate. Afin de donner un cadre uniforme à notre travail, nous allons nous intéresser à la notion de *générateur contraint parfait*, pour lequel nous allons définir l'opération ${}^{\cdot}_{C}$. Si L est le générateur contraint parfait associé à $\mathcal{A}$ et si L' est celui associé à $\mathcal{A}$, on aura alors : $L {}^{\cdot}_{C} L'$ est le générateur contraint parfait associé à $\mathcal{A} {}^{\cdot}_{C} \mathcal{B}$.

Nous obtenons des résultats analogues pour des opérations ${}^{*}_{C}$ et ${}^{\omega}_{C}$ qui correspondent aux itérations finies et infinies de l'opération ${}^{\cdot}_{C}$.

Réciproquement, dans le chapitre III, nous allons tenter de décomposer tout automate temporisé de manière modulaire. Pour cela, nous allons définir des *automates de base* et montrer que tout automate temporisé peut s'exprimer grâce à ces automates de base en utilisant les opérations précédemment définies. L'automate de base que nous allons considérer est représenté à la FIG 1 : il permet de faire un a si chaque horloge x_j est dans I_j (sous-ensemble de dates).

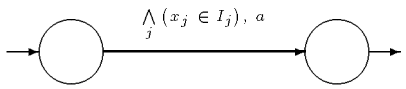


FIG. 1 – *Automate de base*

Pour réaliser notre décomposition, nous allons utiliser une méthode de résolution de systèmes d'équations linéaires temporisées. L'idée de cette méthode est la même que celle utilisée pour les automates finis lorsqu'on recherche une expression rationnelle définissant le langage accepté par un automate. Elle étend aussi la méthode présentée dans [Asa98]. Nous obtenons de cette manière un algorithme de décomposition d'un automate temporisé grâce aux automates de base en utilisant les opérations $+$, ${}^{\cdot}_{C}$, ${}^{*}_{C}$ et ${}^{\omega}_{C}$.

Nous avons rappelé dans ce qui précède que les automates temporisés constituaient une classe de modèles très largement utilisée dans le domaine de la vérification. Or, le test du vide est un problème important pour ce domaine : en effet, certaines propriétés peuvent être décidées grâce au test du vide. Nous allons alors voir dans le chapitre IV que cette décomposition modulaire va nous donner un algorithme pour tester le vide d'un langage temporisé accepté par automate. En effet, nous allons construire de manière modulaire un automate temporisé que l'on va qualifier de «fiable» pour les deux raisons suivantes :

1. il accepte le même langage temporisé que l'automate original (disons L)
2. en le considérant comme un automate non temporisé, c'est à dire lorsqu'on lui retire toutes les contraintes sur les horloges, il accepte exactement la projection du L sur sa composante non temporisée (ce que l'on appelle UNTIME(L))

L'algorithme donne, pour l'automate de base, l'automate de la FIG. 2.

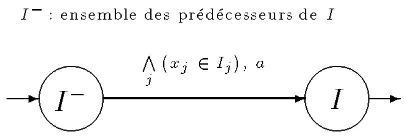


FIG. 2 – *Transformation de l'automate de base via l'algorithme*

La signification des invariants qui se trouvent dans les états est la suivante : lorsqu'on arrive dans l'état, la valeur des horloges doit se trouver dans l'invariant.
Nous obtenons ainsi, grâce à la décomposition modulaire présentée dans le chapitre III, un algorithme pour tester le vide. Rappelons que tester la vacuité d'un langage temporisé accepté par un automate est un problème PSPACE-complet. La complexité de notre algorithme est dans le cas pire la même que celui présenté dans [AD94] (c'est à dire, PSPACE : c'est le cas où tous les invariants des états sont des régions). Par contre, en pratique, la complexité est meilleure dans de nombreux cas puisque l'on construit en fait un automate des zones, c'est à dire que l'on regroupe certains états de l'automate des régions.

Chapitre I

Automates temporisés : un panorama général

Cette partie est consacrée à la description d'un modèle temporisé beaucoup étudié depuis son introduction par Alur et Dill dans [AD90], les *automates temporisés*. Ce modèle utilise celui des automates finis dans lequel des contraintes de temps sont ajoutées. Il sera donc associé à chaque automate temporisé un ensemble fini d'horloges, ces horloges pouvant être comparées à des constantes et pouvant être remises à zéro. En outre, toutes ces horloges avancent à la même vitesse, c'est à dire qu'elles suivent une horloge « universelle » qui avance régulièrement sans jamais être remise à zéro. Dans ce chapitre, nous allons définir ce modèle et étudier quelques propriétés importantes de cette classe d'automates qui nous seront utiles par la suite.

Dans tout ce qui suit, $\mathbb{T}$ est un ensemble de dates (par exemple $\mathbb{Q}^+$). Un intervalle de $\mathbb{T}$ sera un sous-ensemble de $\mathbb{T}$ de la forme (a, b) où a, $b \in \overline{\mathbb{T}}$, avec $a < b$ et $(,) \in \{], [\}$.
$\mathbb{N}_n$ désigne l'ensemble des entiers de 1 à n.

Nous allons commencer par définir les opérations qui nous seront utiles sur les horloges (les contraintes d'horloges, plutôt appelées gardes, ainsi que les valuations d'horloges).

Soit X un ensemble d'horloges.

- L'ensemble des *gardes* sur X est l'ensemble engendré par la grammaire suivante :

$$g ::= \top \mid x \sim c \mid g_1 \wedge g_2 \mid g_1 \vee g_2 \mid \neg g_1$$

  où g_1 et g_2 sont des gardes, $x \in X$, $c \in \mathbb{T}$ et $\sim \in \{=, <, >, \leq, \geq\}$. Une garde g est dite *vérifiée* pour les valeurs des horloges $(a_i)_{1 \leq i \leq n}$ si lorsque x_i est remplacé par a_i dans g, la proposition devient vraie.

- Une *valuation* des horloges de C est une fonction $v : C \longrightarrow \mathbb{T}$. Pour $D \subseteq C$, $[D \leftarrow 0]v$ désigne la valuation qui à $x \in D$ associe 0 et à $x \notin D$ associe $v(x)$.

- Pour v valuation d'horloges, la relation de satisfaction suivante est définie, g étant une garde :

$$v \models g \iff g(x_i/v(x_i)) \text{ est vraie}$$

où $g(x_i/v(x_i))$ désigne la valeur booléenne de g dans lequel x_i est remplacé par sa valeur : $v(x_i)$.

Nous allons maintenant passer à la description du modèle proprement dit.

I.1 Définition du modèle

Pour décrire un système non temporisé, le modèle le plus couramment adopté est celui des suites d'évènements : le mot *abaab* désigne la suite d'évènements (*abaab*) et signifie que l'action a est réalisée, puis l'action b, puis ... Pour décrire un système temporisé, ce modèle ne sera plus suffisant ; en effet, les suites d'évènements permettent juste de décrire l'ordre des actions effectuées par le système ; elles ne donnent aucun renseignement sur les écarts entre les différents évènements. Par exemple, il n'est pas possible d'exprimer le fait qu'une action a a lieu 5 unités de temps après l'action b. C'est pour cette raison que, dans le modèle temporisé, une date va être associée à chaque action.

- Une *séquence de dates* est une suite finie ou infinie de dates.
- Un *mot temporisé* sur l'alphabet A est un élément de $(A \times \mathbb{T})^\infty = (A \times \mathbb{T})^* \cup (A \times \mathbb{T})^\omega$. Il peut être vu comme un couple (σ, τ) avec τ séquence de dates et σ suite sur l'alphabet A de même longueur que τ. Le mot vide sera noté ε.
- Un *langage temporisé* est un ensemble de mots temporisés. Pour L langage temporisé, L_* désigne l'ensemble des mots temporisés finis de L et L_ω celui des mots temporisés infinis de L. Nous avons donc $L = L_* \cup L_\omega$.

NOTATION : Soit L un langage temporisé sur l'alphabet A. UNTIME(L) désigne l'ensemble $\{\sigma \in A^* \mid \exists \tau\, (\sigma, \tau) \in L\}$. C'est le langage non temporisé associé à L.

EXEMPLE : L'ensemble $L = \{(\sigma, \tau) \mid \sigma \in (ab)^\infty, |\sigma| = |\tau| \text{ et } \tau_i > \tau_{i-1} + 0,8\}$ est un langage temporisé. Le mot temporisé $(a; 0,9)(b; 2,1)(a; 3,4)(b; 4,3)$ appartient au langage L car la différence entre deux dates consécutives de ce mot est strictement supérieure à $0,8$.
UNTIME(L) est le langage non temporisé $(ab)^\infty = (ab)^* \cup (ab)^\omega$.

Il apparaît maintenant nécessaire de définir une machine capable de décrire ce type de comportement, en tenant compte, d'une part des actions effectuées, et d'autre part des dates auxquelles ces actions ont lieu. C'est ce qui nous amène à la définition suivante :

Définition 1 *Un* automate temporisé *est un 7-uplet $\langle A, Q, I, F, R, T, X\rangle$ où A est un alphabet fini, Q est un ensemble d'états, $I \subseteq Q$ est un ensemble d'états initiaux, $F \subseteq Q$ est un ensemble d'états*

finals, $R \subseteq Q$ est un ensemble d'états répétés, $T \subseteq Q \times A \times Q \times 2^C \times \Delta$ est un ensemble de transitions (avec Δ ensemble de gardes).
$\mathcal{A}ut(A)$ désigne l'ensemble des automates temporisés sur l'alphabet A.

Une transition de l'automate temporisé précédent est de la forme (q, a, q', r, g), mais elle peut aussi s'écrire $q \xrightarrow{g,a,r} q'$. Elle signifie que si le système se trouve dans l'état q et que la garde g est vérifiée, alors le système peut effectuer l'action a, mettre les horloges de r à zéro et aller dans l'état q'.
Lorsque nous représenterons un automate temporisé, les états seront schématisés par des cercles et les transitions par des flèches étiquetées entre les états. Un état initial sera représenté par un cercle et une petite flèche rentrante ; un état final par un cercle et une petite flèche sortante ; un état répété enfin sera représenté par deux cercles concentriques.

EXEMPLE : Un exemple d'automate temporisé est présenté à la FIG. 3. Cet automate possède

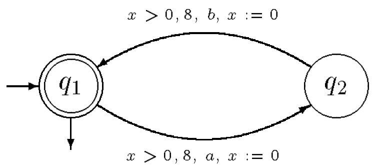


FIG. 3 – *Exemple d'automate temporisé*

une seule horloge, notée x, deux états : q_1 et q_2. L'état q_1 est tout à la fois initial, final et répété. Il possède deux transitions : une de l'état q_1 à l'état q_2 qui permet de réaliser l'action a si la valeur de l'horloge x est strictement supérieure à $0,8$ et remet cette même horloge à 0 ; l'autre transition mène de l'état q_2 à l'état q_1, permet d'effectuer l'action b si x vaut plus que $0,8$ et remet l'horloge x à zéro.
Il est important de remarquer que les comportements futurs de l'automate ne dépendent pas uniquement de l'état dans lequel le système se trouve, mais aussi de la valeur de l'horloge x. En effet, on ne pourra pas faire la même action selon la valeur de x : si $x = 0,5$, on ne peut pas changer d'état, alors que si $x = 0,9$, il est possible de passer dans l'état q_2.

C'est cette dernière constatation qui nous amène à définir ce qu'est une configuration d'un automate : une *configuration* d'un automate $\mathcal{A}$ est un couple $\langle q, v \rangle$ avec q état de $\mathcal{A}$ et v valuation des horloges de $\mathcal{A}$.

Il reste à faire le lien entre la machine que nous venons de définir et les modèles de comportements temporisés qui ont été décrits auparavant. Pour cela, nous allons définir formellement ce qu'est une exécution dans un automate temporisé et ce qu'est la condition d'acceptation pour une exécution donnée.

Définition 2 *Soit $\mathcal{A} = \langle A, Q, I, F, R, T, X \rangle$ un automate temporisé.*

- *Une* exécution *dans $\mathcal{A}$ sur le mot temporisé (σ, τ) est une suite (finie ou infinie) de configurations $(\langle q_i, v_i \rangle)_{i \in J}$ vérifiant :*

$$\exists\, e_i = (q_{i-1} \xrightarrow{g_i, \sigma_i, r_i} q_i) \in T \quad \begin{cases} v_{i-1} + \tau_i \models g_i \\ v_i = {}_{[r_i \leftarrow 0]}(v_{i-1} + \tau_i) \end{cases}$$

- *Une* exécution acceptante *dans $\mathcal{A}$ sur le mot temporisé u est une exécution $(\langle q_i, v_i \rangle)_{i \in J}$ sur u avec :*
  - *$q_0 \in I$ et $v_0 = 0^n$*
  - *si $|J| < \infty$ alors $q_{\max(J)} \in F$ sinon $|\{i \mid q_i \in R\}| = \infty$*

- *Un mot temporisé u est dit* accepté *par $\mathcal{A}$ s'il existe une exécution acceptante pour u dans $\mathcal{A}$.*

- *Le langage temporisé accepté par $\mathcal{A}$ est l'ensemble des mots temporisés acceptés par $\mathcal{A}$. Il est noté $\mathcal{T}(\mathcal{A})$. C'est un sous-ensemble de $(A \times \mathbb{T})^\infty$. Comme dans le cas d'un langage temporisé quelconque, il se décompose en deux sous-parties, $\mathcal{T}_*(\mathcal{A}) = \mathcal{T}(\mathcal{A}) \cap (A \times \mathbb{T})^*$, ensemble des comportements finis de $\mathcal{A}$ et $\mathcal{T}_\omega(\mathcal{A}) = \mathcal{T}(\mathcal{A}) \cap (A \times \mathbb{T})^\omega$, ensemble des comportements infinis de $\mathcal{A}$.*

EXEMPLE : Revenons à l'automate défini dans l'exemple précédent. Un exemple d'exécution de cet automate est le suivant :

$$\langle q_1, 0 \rangle \xrightarrow[0.9]{a} \langle q_2, 0 \rangle \xrightarrow[2.1]{b} \langle q_1, 0 \rangle \xrightarrow[3.4]{a} \langle q_2, 0 \rangle \xrightarrow[4.3]{b} \langle q_1, 0 \rangle$$

C'est une exécution acceptante sur le mot temporisé $(a, 0.9)(b, 2.1)(a, 3.4)(b, 4.3)$ car q_1 est initial et final.
Il est facile de voir que le langage accepté par cet automate temporisé est le langage défini dans le premier exemple, i.e. $L = \{(\sigma, \tau) \mid \sigma \in (ab)^\infty, |\sigma| = |\tau| \text{ et } \tau_i > \tau_{i-1} + 0,8\}$.

Un langage temporisé L est dit *reconnaissable* s'il existe un automate temporisé $\mathcal{A}$ tel que $L = \mathcal{T}(\mathcal{A})$. Il est facile de montrer que si L est un langage reconnaissable, L_* et L_ω sont aussi des langages reconnaissables.

Le modèle ayant été présenté, il est intéressant d'étudier les propriétés de la classe des automates temporisés, notamment en ce qui concerne sa fermeture pour les opérations usuelles sur les ensembles.

Proposition 1 ([AD94]) *La classe des langages temporisés reconnaissables est stable par union, par intersection, mais pas par passage au complémentaire.*

Preuve : – En ce qui concerne l'union et l'intersection, les constructions sont similaires à celles pour les automates de Büchi (voir [WVS83] ou [PP] pour les détails).

– Une idée de preuve est donnée dans [AD94], mais elle n'y est pas du tout développée. C'est pour cette raison qu'elle figure ici. Elle provient d'un cours de DEA d'Antoine Petit.
Pour montrer que la classe des langages temporisés reconnaissables n'est pas fermée par passage au complémentaire, nous allons présenter un contre-exemple : considérons le langage $L = \{(\sigma, \tau) \mid \exists i < j\, \tau_j = \tau_i + 1\}$ sur l'alphabet $\{a\}$. L est un langage temporisé reconnaissable. En effet, il est accepté par l'automate de la FIG. 4. Par contre, nous allons voir que le complémentaire de L n'est pas reconnaissable. Supposons que L^c soit accepté par l'automate

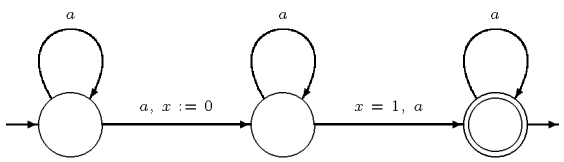
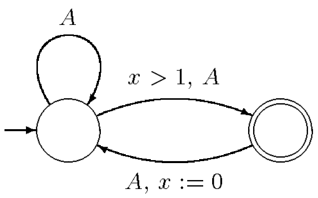


FIG. 4 – *Automate acceptant L*

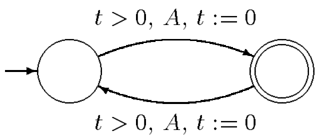
$\mathcal{A}$, automate mis sous forme normale disjonctive (i.e. les gardes ne font intervenir que des conditions d'horloges et l'opérateur $\wedge$). On peut supposer que $\mathcal{A}$ accepte non plus des mots temporisés, mais des séquences de dates. Soit c la plus petite constante non nulle apparaissant dans les gardes de $\mathcal{A}$. Prenons une séquence de dates $0 < t_1 < t_2 \cdots < t_n < c$ acceptée par $\mathcal{A}$. Alors, la séquence de dates $\left(t_1, \cdots t_n, 1 + \left(\frac{t_1+t_2}{2}\right), \cdots, 1 + \left(\frac{t_{n-1}+t_n}{2}\right)\right)$ est acceptée par $\mathcal{A}$. Lorsque l'on considère une exécution qui accepte cette séquence de dates, la garde qui correspond à $1 + \left(\frac{t_{i-1}+t_i}{2}\right)$ fait intervenir un certain nombre d'horloges, toutes ne participant pas à une contrainte « utile ». Celles qui font partie d'une contrainte utile sont des horloges qui ont été mises à zéro soit au tout début, soit lors d'une transition pour t_j car les horloges mises à zéro lors d'un $1 + \left(\frac{t_{j-1}+t_j}{2}\right)$ ne peuvent pas avoir d'influence : la garde qui leur correspond est obligatoirement triviale car $|1 + \left(\frac{t_{i-1}+t_i}{2}\right) - 1 + \left(\frac{t_{j-1}+t_j}{2}\right)| < c$ et c est la plus petite constante apparaissant dans l'automate. Ainsi, les horloges utiles à la transition $1 + \left(\frac{t_{i-1}+t_i}{2}\right)$ ont leur valeur fixée par leur valeur en c. Ainsi, ces horloges utiles définissent pour chaque transition intervenant dans l'exécution considérée un intervalle contenant chacun un $1 + \left(\frac{t_{j-1}+t_j}{2}\right)$. Notons A_j cet intervalle ; il doit être inclus dans l'intervalle $]1+t_{j-1}; 1+t_j[$ car sinon, l'automate accepterait soit le mot $\left(t_1, \cdots, t_n, 1 + \left(\frac{t_1+t_2}{2}\right), \cdots, q + t_{j-1}\right)$ soit le mot $\left(t_1, \cdots, t_n, 1 + \left(\frac{t_1+t_2}{2}\right), \cdots, q + t_j\right)$, ce qui n'est pas possible. Chaque A_j est contenu dans un intervalle I_j avec les I_k deux à deux disjoints. Or, pour toute exécution, le nombre d'intervalles A_j est bornée par $2^x * t$ où x est le nombre d'horloges de l'automate et t son nombre de transitions. Donc, il suffit de choisir n plus grand que $2^x * t$ pour obtenir une contradiction (il y aura plus d'intervalles A_j que la borne supérieure autorisée, donc deux A_j seront égaux, ce qui n'est pas possible car les I_k sont deux à deux disjoints).

$\square$

REMARQUE : *Dans l'article original d'Alur et Dill ([AD94]), les conditions d'acceptation d'un mot temporisé étaient plus restrictives : la séquence de dates devait être infinie, strictement croissante et divergente (lorsque la séquence de dates est divergente, le mot est dit* non-Zeno*). Mais ces conditions d'acceptation peuvent être retrouvées grâce au modèle défini ci-avant. Soit $\mathcal{A}$ un automate temporisé. Alors le langage accepté par $\mathcal{A}$ pour Alur et Dill est en fait $\mathcal{T}(\mathcal{A}) \cap \mathcal{T}(\mathcal{B}) \cap \mathcal{T}(\mathcal{C})$ où $\mathcal{B}$ et $\mathcal{C}$ sont définis à la* FIG. 5.



(a) Automate $\mathcal{B}$ reconnaissant les mots temporisés non-Zeno



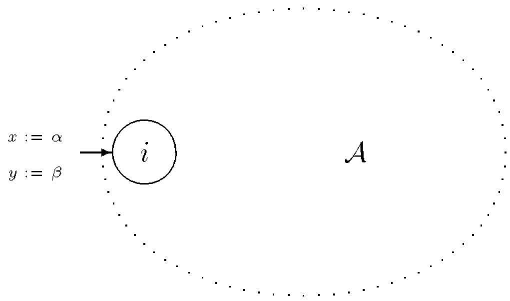
(b) Automate $\mathcal{C}$ reconnaissant les mots temporisés strictement croissants

FIG. 5 – *Automates $\mathcal{B}$ et $\mathcal{C}$*

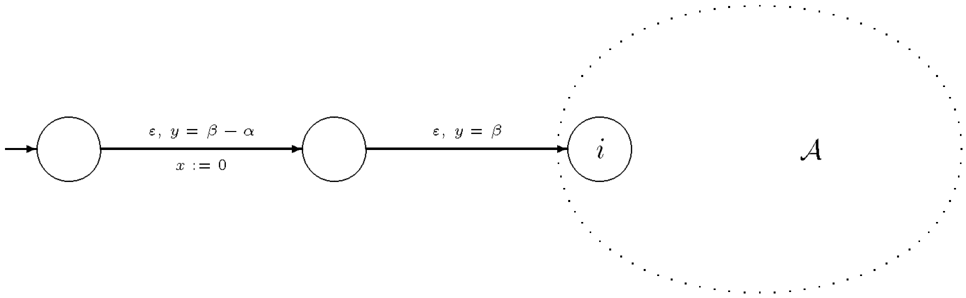
Dans la définition d'un automate temporisé, toutes les horloges sont initialisées à zéro à l'entrée de l'automate. Cependant, il serait légitime de se poser la question de savoir si le fait d'initialiser les horloges à une autre valeur que zéro donnerait un modèle plus puissant que celui défini ou pas. Le lemme 2 établit que, en fait, ce n'est pas le cas.

Lemme 2 *Soit $\mathcal{A}$ un automate temporisé pour lequel les n horloges sont initialisées à $\alpha \in \mathbb{T}^n$. Alors il existe un automate temporisé $\mathcal{B}$ dans lequel les horloges sont initialisées à zéro acceptant le même langage que $\mathcal{A}$.*

Idée de preuve : Le cas général réclame de faire une figure, certes facile à construire, mais un peu illisible et surchargée. C'est pour cette raison que nous allons uniquement donner une idée de preuve dans le cas d'un automate $\mathcal{A}$ qui possède deux horloges, x et y, que nous voulons initialiser non plus à zéro, mais à α et β respectivement. Par exemple avec $\alpha < \beta$, l'automate $\mathcal{A}$ peut être représenté comme à la FIG. 6(a). Les automates de la FIG. 6 reconnaissent le même langage temporisé. En effet, dans le deuxième automate, lorsque l'on arrive dans l'ex-état initial de l'automate de départ, x vaut α et y vaut β. Ainsi, le langage accepté par cet automate est la concaténation de $(\varepsilon, \beta - \alpha)(\varepsilon, \beta)$ et du langage accepté par $\mathcal{A}$ lorsque ses horloges sont initialisées non plus à zéro, mais à α et β ; c'est donc le même langage que le premier automate. Or dans [DGP97], il a été montré que toutes les ε-transitions présentes dans l'automate de la FIG. 6(b) peuvent être enlevées (car aucune ε-transition n'est dans un cycle), i.e. il existe un automate temporisé $\mathcal{B}$ sans ε-transitions qui reconnait le même langage que $\mathcal{A}$. $\square$



(a) Automate avec initialisation des horloges



(b) Automate avec ε-transitions

FIG. 6 – *Automates reconnaissant le même langage ($\alpha < \beta$)*

Ainsi, dans la suite, les automates pour lesquels les horloges sont initialisées à une valeur quelconque seront autorisés, comme dans la FIG. 6(a). Si $\mathcal{A}$ est un automate temporisé (dans lequel les horloges sont initialisées à zéro), $\mathcal{A}^\alpha$ désignera le même automate pour lequel les horloges sont initialisées à α.

I.2 Propriétés de décidabilité

Le problème qui nous intéresse ici est celui de la vérification. Quelques problèmes basiques de décidabilité sont liés à celui de la vérification, notamment ceux du test du vide et du test de l'inclusion. En effet, si L est un langage temporisé que l'on veut vérifier et si L' est le langage des modèles de la spécification, décider que L vérifie la spécification revient à décider si $L \subseteq L'$. En outre, si q' est un état qu'il ne faut pas atteindre dans l'automate, vérifier que le système est correct pour cette propriété revient à tester si le langage accepté par l'automate pour lequel l'état final est q' est vide ou non. Il est donc important de connaître la complexité de ces différents problèmes pour la classe d'automates que l'on considère ici.

I.2.1 Résultats de décidabilité ([AD94])

En ce qui concerne les langages non temporisés classiques, le vide et l'inclusion sont décidables. Ces propriétés donnent alors des algorithmes de model-checking pour certaines logiques temporelles non temporisées telles que CTL. En effet, pour chaque formule de CTL, il est possible de construire un automate reconnaissant exactement les modèles de cette formule. Ainsi, tester si un langage reconnaissable L vérifie une propriété de CTL revient à tester si L est inclus dans le langage accepté par l'automate. C'est pour cette raison qu'il serait intéressant de savoir quelles sont les propriétés de décidabilité qui restent vraies pour les automates temporisés. Ainsi, cela pourrait donner directement des algorithmes de model-checking pour les automates temporisés.

Théorème 3 ([AD94]) *Le problème de décider du vide d'un langage temporisé reconnaissable est* PSPACE*-complet.*

Cependant, comme la classe des automates temporisés n'est pas fermée par passage au complémentaire, décider de l'inclusion de deux langages reconnaissables ne se réduit pas à décider le vide d'un certain langage reconnaissable comme dans le cas non temporisé. Et malheureusement, nous obtenons même le résultat d'indécidabilité suivant :

Théorème 4 ([AD94]) *Décider de l'inclusion d'un langage temporisé reconnaissable dans un autre langage temporisé reconnaissable est un problème indécidable.*

Ce premier résultat négatif va inciter à se restreindre à une sous-classe des automates temporisés, pour tenter d'obtenir un ensemble dans lequel l'inclusion serait décidable.

Soit $\mathcal{A} = \langle A, Q, I, F, R, T, X \rangle$ un automate temporisé. Il sera dit *déterministe* si pour toutes les transitions (q, a, q', r, g) et (q, a, q'', r', g') on a $g \wedge g'$ toujours faux.

Théorème 5 ([AD94]) *Décider de l'inclusion d'un langage reconnaissable dans un langage reconnaissable accepté par un automate temporisé déterministe est un problème* PSPACE*-complet.*

La preuve de tous ces théorèmes repose sur la construction de l'automate des régions qui va être présentée dans la section I.2.2 qui suit.

I.2.2 Construction de l'automate des régions

Un automate temporisé possède un nombre infini de configurations possibles. Ce que nous cherchons à faire, c'est d'en regrouper certaines pour obtenir un système possédant un nombre fini de configurations possibles et qui, cependant, pourrait « reproduire » les comportements de l'automate initial (nous verrons un petit peu plus tard ce que « reproduire » signifie exactement).
Pour réaliser cela, donnons-nous un automate temporisé $\mathcal{A} = \langle A, Q, I, F, R, T, X \rangle$ pour lequel toutes les contraintes sont dans $\mathbb{Q}^+$ (si $\mathbb{T} = \mathbb{R}^+$, on peut se ramener à ce cas).

Lemme 6 *Soit $\mathcal{A}$ un automate temporisé. Soit t le plus grand multiple commun des numérateurs des constantes apparaissant dans les contraintes de $\mathcal{A}$. Soit $\mathcal{B}$ l'automate obtenu à partir de $\mathcal{A}$ en multipliant toutes ces constantes par t. Alors*

$$(\sigma, \tau) \in \mathcal{T}(\mathcal{A}) \iff (\sigma, t.\tau) \in \mathcal{T}(\mathcal{B})$$

D'après le lemme 6, quitte à multiplier par une constante, nous pouvons supposer que toutes les constantes apparaissant dans les gardes de l'automate $\mathcal{A}$ sont entières. Par la suite, tous les automates considérés seront supposés vérifier cette propriété.

Une *région* est un sous-ensemble de $\mathbb{T}^n$ défini par les équations suivantes : pour toute horloge $x \in X$, la contrainte sur x est soit $x = k$ avec $k \in [\![0..c_x]\!]$, soit $x > c_x$, soit $k - 1 < x < k$ avec $k \in \mathbb{N}_{c_x}$; pour tout couple d'horloges $(x, y) \in X^2$, si $x \leq c_x$ et $y \leq c_y$, alors il y a une contrainte du type $\mathsf{Frac}(x) \sim \mathsf{Frac}(y)$ avec $\sim \in \{<, >, =\}$, $\mathsf{Frac}(x)$ désignant la partie fractionnaire de x. Un exemple de régions est présenté à la Fig. 7. Ces régions sont destinées à découper l'ensemble des comportements possibles de l'automate de départ en un nombre plus «restreint» de comportements. Elles vont avoir la propriété suivante : « être dans l'état q avec comme valuation $v \in R$ » va être équivalent à « être dans l'état q avec comme valuation $v' \in R$ », dans le sens que les comportements possibles futurs vont être les mêmes.

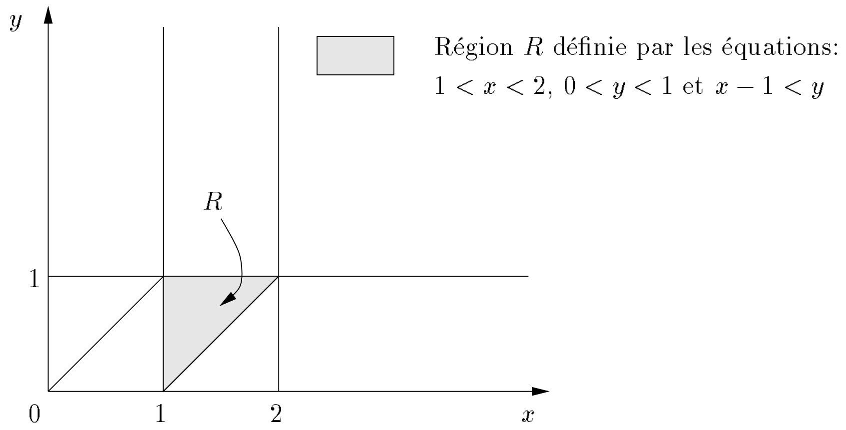


Fig. 7 – *Les 28 régions dans le cas de 2 horloges x et y avec $c_x = 2$ et $c_y = 1$*

Notation : Soit R une région. Soit $C \subseteq X$. On note $\mathsf{Succ}_C(R)$ l'ensemble des régions R' telles que pour toute valuation $v' \in R'$, il existe une valuation $v \in R$, telle qu'il existe $t \in \mathbb{T}$ avec pour tout $x \in X \backslash C$, $v'(x) = v(x) + t$ et si $x \in C$, $v'(x) = 0$. En fait, $\mathsf{Succ}_C(R)$ est l'ensemble des régions accessibles à partir de R en laissant passer le temps et en remettant les horloges de C à zéro.
Exemple : Sur la Fig. 7, $\mathsf{Succ}_y(R)$ est l'ensemble des régions suivantes :

- le segment $\{1 < x < 2; y = 0\}$
- le point $(x = 2, y = 0)$
- le segment $\{x > 2; y = 0\}$

Nous allons construire un *graphe des successeurs des régions* qui va modéliser la relation «successeur» entre les régions. Il est défini de la manière suivante : les états de ce graphe sont les régions ; il y a une transition étiquetée par $C := 0$ entre les régions R et R' si $R' \subseteq \mathsf{Succ}_C(R)$.

Définition 3 *Soit $\mathcal{A} = \langle A, Q, I, F, R, T, X\rangle$ un automate temporisé. Soit $\mathcal{R}$ l'ensemble des régions associées à $\mathcal{A}$. L'automate des régions associé à $\mathcal{A}$ est le produit cartésien de $\mathcal{A}$ avec le graphe des successeurs des régions précédemment défini, c'est à dire c'est l'automate fini $\langle A, Q', I', F', R', T'\rangle$ défini par :*

$$\begin{cases} Q' = Q \times \mathcal{R} \\ I' = I \times 0^n \\ F' = F \times \mathcal{R} \\ R' = R \times \mathcal{R} \\ T' = \{((q,R), a, (q',R')) \mid \exists (q,a,q',r,g) \quad g \in R' \text{ et } R' \in \mathsf{Succ}_r(R)\} \end{cases}$$

Nous noterons $\mathcal{R}(\mathcal{A})$ l'automate des régions ainsi obtenu et $\mathcal{L}(\mathcal{R}(\mathcal{A}))$ le langage non temporisé accepté par l'automates des régions. Comme nous allons le voir, il va «reproduire» les comportements de $\mathcal{A}$ (dans le sens du théorème suivant).

Théorème 7 ([AD94]) *Soit $\mathcal{A}$ un automate temporisé et soit $\mathcal{R}(\mathcal{A})$ l'automate des régions qui lui est associé. Alors :*

- *pour tout chemin $(\langle q_i, v_i\rangle)_{i\in J}$ sur (σ, τ) dans $\mathcal{A}$, il existe un chemin $((q_i, R_i))_{i\in J}$ sur σ dans $\mathcal{R}(\mathcal{A})$ avec pour tout $i \in J$, $v_i \in R_i$.*
- *la réciproque est vraie : pour tout chemin $((q_i, R_i))_{i\in J}$ sur σ dans $\mathcal{R}(\mathcal{A})$, il existe un chemin $(\langle q_i, v_i\rangle)_{i\in J}$ sur (σ, τ) dans $\mathcal{A}$ tel que pour tout $i \in J$, $v_i \in R_i$.*

La preuve de ce théorème ne sera pas rédigée ici. Elle se trouve dans l'article [AD94].

Corollaire 8 *Soit $\mathcal{A}$ un automate temporisé. Soit $\mathcal{R}(\mathcal{A})$ l'automate des régions qui lui est associé. Alors $\mathcal{L}(\mathcal{R}(\mathcal{A})) = \textsc{Untime}(\mathcal{T}(\mathcal{A}))$.*

I.2.3 Comment cet automate permet-il de démontrer les théorèmes annoncés ?

Nous avons vu dans la section précédente que si $\mathcal{A}$ est un automate temporisé, il est possible de construire un automate fini $\mathcal{R}(\mathcal{A})$ vérifiant $\mathcal{L}(\mathcal{R}(\mathcal{A})) = \textsc{Untime}(\mathcal{T}(\mathcal{A}))$. En conséquence,

$$\mathcal{T}(\mathcal{A}) = \varnothing \iff \mathcal{L}(\mathcal{R}(\mathcal{A})) = \varnothing$$

Or, tester le vide d'un langage reconnu par un automate fini est linéaire en la taille de l'automate fini. Ici, la taille de l'automate des régions est linéaire en la taille de l'automate initial, mais exponentielle en le nombre d'horloges. Un algorithme dans PSPACE consiste à ne pas construire directement l'automate des régions, mais de le construire au fur et à mesure que l'on a besoin des états. Donc, tester le vide est dans PSPACE. En outre, il est possible de coder le problème « appartenance d'une chaîne de caractères au langage accepté par un automate linéaire borné » sur le test du vide d'un automate temporisé. Ceci signifie que « tester le vide » est un problème PSPACE-dur.

Ceci conclut la preuve de PSPACE-complétude du problème « tester le vide d'un langage reconnaissable ».

En ce qui concerne l'inclusion de deux langages temporisés reconnaissables, ce n'est pas aussi simple que pour la classe des automates finis. En effet, la classe des automates temporisés n'étant pas fermée par passage au complémentaire, il n'est pas très utile de constater que $L \subseteq L' \iff L \cap L'^c = \varnothing$. En fait, le problème universel pour les automates temporisés se trouve être indécidable (réduction du problème de l'exécution récurrente d'une machine à deux compteurs). Alors, le problème de l'inclusion s'en trouve aussi indécidable.

Le cas des automates temporisés déterministes est beaucoup plus simple : le complémentaire d'un langage accepté par un automate temporisé déterministe est reconnaissable (voir lemme 9). Donc, nous pouvons utiliser la propriété $L \subseteq L' \iff L \cap L'^c = \varnothing$ si L' est un langage reconnu par un automate temporisé déterministe. La PSPACE-complétude du problème provient alors de la PSPACE-complétude du test du vide.

Lemme 9 *Soit $\mathcal{A}$ un automate temporisé déterministe. Alors, $\mathcal{T}(\mathcal{A})^c$ est reconnaissable.*

Preuve : Contrairement à tout ce qui vient d'être fait, la preuve de ce lemme est très simple. En effet, supposons que $\mathcal{A} = \langle A, Q, I, F, R, T, X \rangle$. Soit $\mathcal{B}$ l'automate temporisé complet à la Müller reconnaissant le même langage que $\mathcal{A}$ (R se transforme en $\mathcal{R} = \{P \subseteq Q \mid P \cap R \neq \varnothing\}$). Seule la condition d'acceptation pour les mots temporisés infinis change ; elle devient : $(\langle q_i, v_i \rangle)_{i \in \mathbb{N}}$ est une exécution acceptante pour u si et seulement c'est une exécution sur u vérifiant $q_0 \in I$ et $\{q \in Q \mid |\{i \mid q_i = q\}| = \infty\} \in \mathcal{R}$. Cela signifie que l'ensemble des états par lesquels l'exécution passe une infinfité de fois doit appartenir à $\mathcal{R}$. Alors $\mathcal{B}$ est déterministe et l'automate $\mathcal{C}$ identique à $\mathcal{B}$ sauf pour la condition d'acceptation accepte exactement $\mathcal{T}(\mathcal{A})$. Il suffit alors de prendre le complémentaire de F dans Q et le complémentaire de $\mathcal{R}$ dans $\mathcal{P}(Q)$ pour obtenir un automate acceptant le complémentaire de $\mathcal{T}(\mathcal{A})$. □

Ceci conclut les preuves des théorèmes de décidabilité.

Maintenant que nous avons présenté quelques résultats de base sur le modèle que nous étudions, nous pouvons vraiment commencer notre étude.

Chapitre II Composition d'automates temporisés

Lorsqu'on cherche à modéliser les systèmes réels, on se heurte souvent au problème de la taille du modèle que l'on construit pour ce système (voir par exemple [HSGL97]). Une solution couramment utilisée est de modéliser le gros système par une famille de petits systèmes (comme par exemple un réseau d'automates temporisés pour lesquels on définit des opérarions de composition parallèle, comme la synchronisation). Dans ce chapitre, nous allons chercher à définir une famille d'opérations de composition sur les automates temporisés. Nous verrons ensuite quelques propriétés liées à ces opérations.

II.1 Nouvelles opérations sur les automates temporisés

Une idée simple pour définir une opération de composition d'automates temporisés consisterait à composer séquentiellement des automates temporisés. Pour réaliser ceci, deux possibilités naturelles s'offrent à nous : soit toutes les horloges du premier automate sont remises à zéro, soit aucune n'est remise à zéro avant de continuer dans l'automate $\mathcal{B}$. En effet, le premier cas représenterait l'exécution séquentielle de deux programmes indépendants, alors que le second cas représenterait l'exécution séquentielle de deux programmes qui ont leurs horloges en commun.
Dans leur papier [ACM97], Asarin, Caspi et Maler ont aussi choisi ce genre de modèle : ayant restreint dans un premier temps leur étude aux automates temporisés ayant une seule horloge, ils ont défini deux opérations sur les automates temporisés qui correspondent à la concaténation de deux automates avec ou sans remise à zéro de l'horloge.
Cependant, ces deux types de concaténation ne semblent pas suffisants pour décrire certains mécanismes. En effet, considérons un poste de travail dans une chaîne (cf FIG. 8). Un «travail élémentaire» est la succession de trois actions : prendre la pièce, travailler sur cette pièce puis la reposer. Une fois qu'un travail élémentaire est effectué (dans un temps limité à $0,2$ heures), la personne qui travaille à ce poste recommence la même séquence de travail. Tout ceci est vrai pendant la journée de travail, pas entre 17 heures et 9 heures le lendemain matin. En conséquence, lorsque l'on recommence un cycle de travail, l'heure doit être comprise entre 9 heures et 17 heures. Une horloge, h sur le dessin, modélise l'heure « absolue » et ne doit par conséquent pas être remise à zéro au début d'un travail élémentaire, alors que l'autre horloge, x, qui mesure le temps mis à effectuer le travail élémentaire

doit être remise à zéro à chaque cycle. Ceci nous amène à vouloir définir un opérateur qui compose séquentiellement deux automates temporisés en mettant uniquement certaines horloges à zéro entre les deux automates.

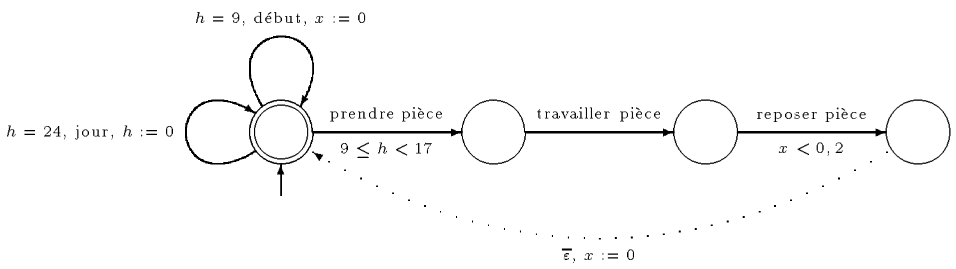


FIG. 8 – *Modélisation d'un travail à la chaîne*

Ainsi, nous allons définir pour chaque sous-ensemble d'horloges $C \subseteq X$ une opération de composition entre les automates temporisés, notée $;_C$ et décrite à la FIG. 9(a).
Formellement, cette opération se définit de la manière suivante : étant donnés des automates temporisés $\mathcal{A} = \langle A, Q_1, I_1, F_1, R_1, T_1, X_1 \rangle$ et $\mathcal{B} = \langle A, Q_2, I_2, F_2, R_2, T_2, X_2 \rangle$, soit y une horloge externe, i.e. une horloge qui n'appartient ni à X_1, ni à X_2 et soit $X = X_1 \cup X_2$, alors nous définissons

$$\mathcal{A} ;_C \mathcal{B} = \langle A, Q_1 \cup Q_2, I_1, F_2, R_1 \cup R_2, T, X \cup \{y\} \rangle$$

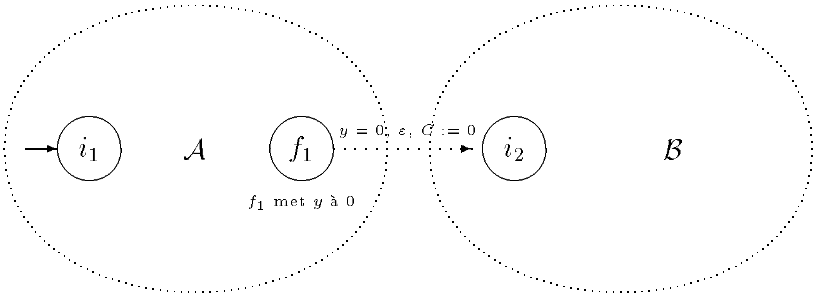
avec

$$\begin{aligned} T &= \{(q, a, q', r, g) \in T_1 \mid q' \notin F_1\} \\ &\cup \{(q, a, q', r \cup \{y\}, g) \mid (q, a, q', r, g) \in T_1 \text{ et } q' \in F_1\} \\ &\cup T_2 \cup \{(f, \varepsilon, i, C, y = 0) \mid f \in F_1 \text{ et } i \in I_2\} \end{aligned}$$

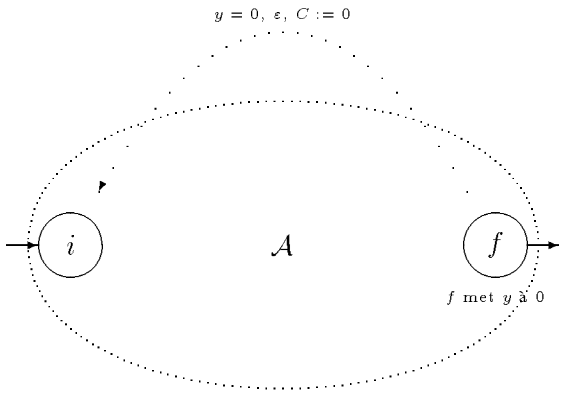
Nous n'avons plus qu'à réaliser la clôture de ces opérations par itérations finies et infinies. Ces nouvelles opérations, notées respectivement $\overset{*}{C}$ et $\overset{\omega}{C}$ sont décrites sur les FIG 9(b) et 9(c).
Formellement, ces opérations sont définies de la manière suivante : étant donné un automate temporisé $\mathcal{A}$, soit y une horloge externe, i.e. une horloge qui n'appartient ni à X_1, ni à X_2 et soit $X = X_1 \cup X_2$, alors nous définissons

- $\mathcal{A}^{\overset{*}{C}} = \langle A, Q_1, I_1, F_1, R_1, T, X \cup \{y\} \rangle$ avec

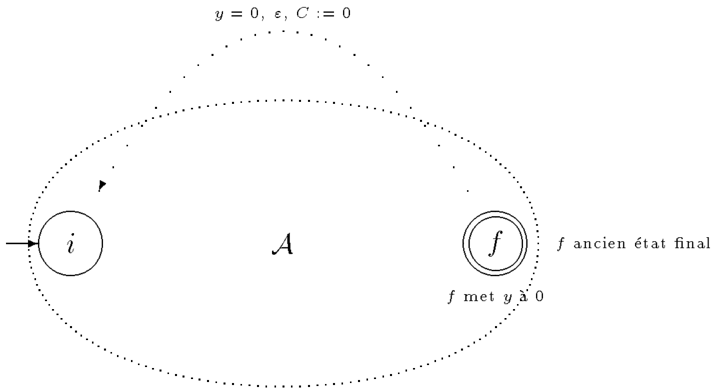
$$\begin{aligned} T &= \{(q, a, q', r, g) \in T_1 \mid q' \notin F_1\} \\ &\cup \{(q, a, q', r \cup \{y\}, g) \mid (q, a, q', r, g) \in T_1 \text{ et } q' \in F_1\} \\ &\cup \{(f, \varepsilon, i, C, y = 0) \mid, f \in F_1 \text{ et } i \in I_1\} \end{aligned}$$



(a) Automate $\mathcal{A} \cdot_C \mathcal{B}$



(b) Automate $\mathcal{A}^{\overset{*}{C}}$



(c) Automate $\mathcal{A}^{\overset{\omega}{C}}$

FIG. 9 – *Nouvelles opérations sur les automates*

- $\mathcal{A}^{\overset{\omega}{C}} = \langle A, Q_1, I_1, \varnothing, R_1 \cup F_1, T, X \cup \{y\}\rangle$ avec

$$\begin{array}{rcl} T & = & \{(q,a,q',r,g) \in T_1 \mid q' \notin F_1\} \\ & & \cup\{(q,a,q',r\cup\{y\},g) \mid (q,a,q',r,g) \in T_1 \text{ et } q' \in F_1\} \\ & & \cup\{(f,\varepsilon,i,C,y=0) \mid, f \in F_1 \text{ et } i \in I_1\} \end{array}$$

Par la suite, au lieu de rajouter une horloge y, on notera les ε-transitions qui sont rajoutées $\overline{\varepsilon}$. Si $(q,\overline{\varepsilon},q',r,g)$ est une transition, on pourra passer de la configuration $\langle q,v\rangle$ à la configuration $\langle q',v'\rangle$ si et seulement si $v \models g$ et si $v' = {}_{[r\leftarrow 0]}v$.

II.2 Action sur les langages temporisés reconnaissables

Une fois définies ces opérations de composition, il est important d'étudier les comportements d'un automate composé en fonction de ses sous-automates de base. Dans ce but, nous allons essayer de translater sur les langages temporisés reconnaissables ces trois opérations, qui paraissent naturelles sur les automates temporisés. Une fois ceci fait, il faudra alors étendre notre étude aux ensembles de comportements quelconques, c'est à dire aux langages temporisés quelconques (qu'ils soient reconnaissables ou non).

Comme dit plus haut, nous allons commencer par étudier comment ces nouvelles opérations agissent sur les langages temporisés reconnaissables. Dans ce but, donnons-nous deux automates temporisés $\mathcal{A}$ et $\mathcal{B}$ et étudions le langage accepté par $\mathcal{A}\,\dot{}_C\,\mathcal{B}$.
NOTATION : Si $u = (\sigma,\tau)$ et $u' = (\sigma',\tau')$ sont des mots temporisés, u' étant un mot fini, nous noterons $u\backslash_{u'}$ le mot temporisé $(\sigma, \tau - \mathsf{max}(\tau'))$.
Soit $u \in \mathcal{T}_*(\mathcal{A}\,\dot{}_C\,\mathcal{B})$. Un chemin acceptant pour u dans $\mathcal{A}\,\dot{}_C\,\mathcal{B}$ peut se décomposer en deux parties : une dans $\mathcal{A}$ et une autre dans $\mathcal{B}$ (voir FIG. 10). On peut donc écrire $u = v.w$ avec v mot accepté par l'automate $\mathcal{A}$ et $w\backslash_v$ mot temporisé accepté par l'automate $\mathcal{B}$ lorsque l'on initialise ses horloges à zéro pour les horloges qui appartiennent à C et à la valeur de sortie de l'automate $\mathcal{A}$ après lecture de v pour les horloges n'appartenant pas à C.
Soit $u \in \mathcal{T}_\omega(\mathcal{A}\,\dot{}_C\,\mathcal{B})$. Deux cas sont possibles : soit lorsqu'on lit le mot dans l'automate, on reste dans $\mathcal{A}$, soit, à un moment donné, on passe dans l'automate $\mathcal{B}$. Dans le premier cas, cela signifie que $u \in \mathcal{T}_\omega(\mathcal{A})$. Dans le deuxième cas, on peut, comme pour les mots finis, découper u en deux parties v et w vérifiant : v appartient au langage fini accepté par $\mathcal{A}$ et $w\backslash_v$ appartient au langage infini accepté par $\mathcal{B}$ lorsque les horloges sont initialisées à zéro pour les horloges qui appartiennent à C et à la valeur de sortie de l'automate $\mathcal{A}$ après lecture de v pour les horloges n'appartenant pas à C.

Nous remarquons qu'il devient nécessaire, étant donné un mot temporisé fini u, de connaître les différentes valeurs possibles des horloges après acceptation de u par un automate temporisé donné. Il apparaît donc nécessaire de se donner une fonction qui donnerait les valeurs possibles des horloges « à la sortie » de l'automate.
Par contre, il n'est pas facile d'exprimer le langage accepté par $\mathcal{A}\,\dot{}_C\,\mathcal{B}$ en fonction des langages acceptés par $\mathcal{A}$ et $\mathcal{B}$. L'exemple suivant le montre bien.
EXEMPLE : Nous allons voir que, en général, $\mathcal{T}(\mathcal{A}\,\dot{}_y\,\mathcal{B})$ ne s'obtient en général pas à partir de

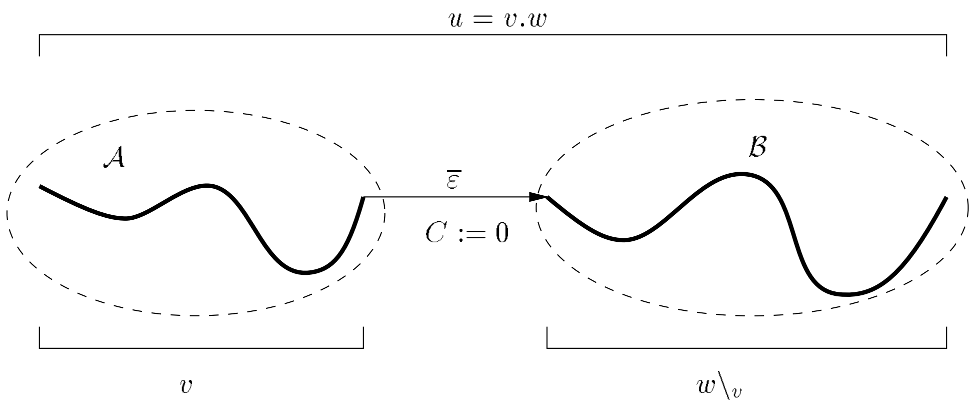


FIG. 10 – *Décomposition des mots acceptés par* $\mathcal{A} \mathbin{;_C} \mathcal{B}$

$\mathcal{T}(\mathcal{A})$ et $\mathcal{T}(\mathcal{B})$. En effet, considérons les automates de la FIG. 11. L'automate $\mathcal{A}$ accepte le langage temporisé

$$\{(a,t)(b,t') \mid t' = 5 \text{ et } t < 2\}$$

et l'automate $\mathcal{B}$ accepte le langage temporisé

$$\{(d,t) \mid t < 4\}$$

alors que l'automate $\mathcal{A} \mathbin{;_y} \mathcal{B}$ accepte le langage temporisé

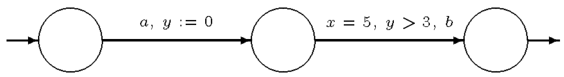
$$\{(a,t)(b,t')(c,t'') \mid t < 2,\, t' = 5 \text{ et } 6 < t'' < 7\}$$

Ceci montre bien que le langage $\mathcal{T}(\mathcal{A} \mathbin{;_y} \mathcal{B})$ ne s'obtient pas à partir les langages $\mathcal{T}(\mathcal{A})$ et $\mathcal{T}(\mathcal{B})$. C'est à ce moment qu'il faut remettre en question notre point de vue sur ce qu'est exactement un automate temporisé. Jusqu'à présent, à tout automate temporisé définissait un langage temporisé. En fait, *un automate temporisé définit une famille de langages* : à chaque valeur initiale des horloges est associé le langage accepté par l'automate lorsqu'on initialise ses horloges à cette valeur. C'est ce changement de point de vue sur les automates temporisés qui va permettre de comprendre exactement quels sont les comportements de $\mathcal{A} \mathbin{;_C} \mathcal{B}$ en fonction de ceux de $\mathcal{A}$ et de $\mathcal{B}$.

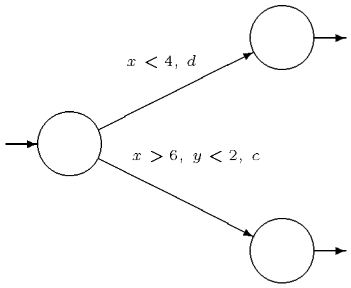
Le même genre d'étude peut être fait pour les deux autres opérations $\overset{*}{;}_C$ et $\overset{\omega}{;}_C$. Nous ne les détaillerons pas ici.

Nous allons maintenant formaliser ces notions.

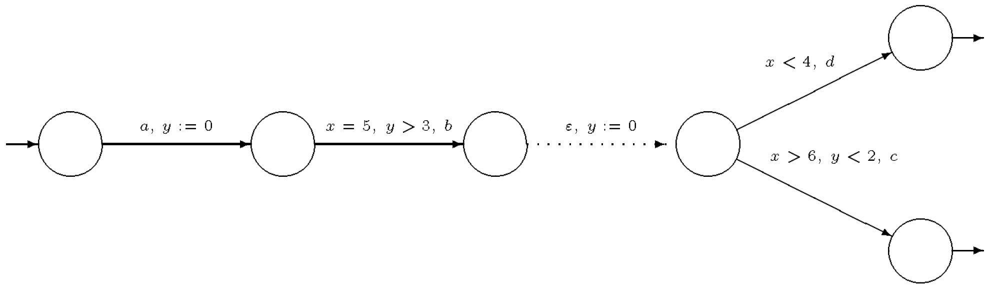
Soit $\mathcal{C}$ un automate temporisé. Pour $u \in \mathcal{T}_*(\mathcal{C})$, une *valeur finale* de u est un élément $\beta \in \mathbb{T}^n$ vérifiant : il existe une exécution acceptante $(\langle q_i, v_i\rangle)_{0 \leq i \leq m}$ pour u dans $\mathcal{C}$ telle que $v_m = \beta$. Nous notons $\lambda_{\mathcal{C}}$ la fonction qui à un mot temporisé $u \in \mathcal{T}_*(\mathcal{C})$ associe l'ensemble des valeurs finales de u et à tout autre mot temporisé associe $\emptyset$.



(a) Automate $\mathcal{A}$



(b) Automate $\mathcal{B}$



(c) Automate $\mathcal{A}\,\dot{}_{y}\,\mathcal{B}$

FIG. 11 – *Exemple de « concaténation »*

En étudiant de plus près ce qui précède, il est facile de prouver les propriétés suivantes :

$$\begin{cases} \mathcal{T}_*(\mathcal{A}\,\dot{}_{C}\,\mathcal{B}) & = & \{u.v \mid u \in \mathcal{T}_*(\mathcal{A}) \text{ et } \exists \alpha \in \lambda_{\mathcal{A}}(u)\ v\backslash_u \in \mathcal{T}_*(\mathcal{B}^\alpha)\} \\ \mathcal{T}_\omega(\mathcal{A}\,\dot{}_{C}\,\mathcal{B}) & = & \mathcal{T}_\omega(\mathcal{A}) \cup \{u.v \mid u \in \mathcal{T}_*(\mathcal{A}) \text{ et } \exists \alpha \in \lambda_{\mathcal{A}}(u)\ v\backslash_u \in \mathcal{T}_\omega(\mathcal{B}^\alpha)\} \end{cases}$$

$$\begin{cases} \mathcal{T}_*(\mathcal{A}^{\overset{*}{C}}) & = & \displaystyle\bigcup_{i \in \mathbb{N}^*} \mathcal{T}_*(\mathcal{A}^{(\dot{C})^i}) \\ \mathcal{T}_\omega(\mathcal{A}^{\overset{*}{C}}) & = & \mathcal{T}_\omega(\mathcal{A}) \cup \{u.v \mid u \in \mathcal{T}_*(\mathcal{A}^{\overset{*}{C}}) \text{ et } \exists \alpha \in \lambda_{\mathcal{A}^{\overset{*}{C}}}(u)\ v\backslash_u \in \mathcal{T}_\omega(\mathcal{A})\} \end{cases}$$

$$\begin{cases} \mathcal{T}_*(\mathcal{A}^{\overset{\omega}{C}}) & = & \varnothing \\ \mathcal{T}_\omega(\mathcal{A}^{\overset{\omega}{C}}) & = & \mathcal{T}_\omega(\mathcal{A}^{\overset{*}{C}}) \cup \{u_0.u_1 \cdots \mid u_0 \in \mathcal{T}_*(\mathcal{A}) \text{ et } \forall i \in \mathbb{N}^*, \\ & & \qquad \exists \alpha_i \in \lambda_{\mathcal{A}^{\alpha_{i-1}}}(u_{i-1}\backslash_{(u_0 \cdots u_{i-2})}) u_i\backslash_{(u_0 \cdots ui-1)} \in \mathcal{T}_*(\mathcal{A}^{\alpha_i})\} \end{cases}$$

où $\mathcal{A}^{(\dot{C})^{i+1}} = \mathcal{A} \dot{_C} \mathcal{A}^{(\dot{C})^{i}}$ et $\mathcal{A}^{(\dot{C})^{1}} = \mathcal{A}$.

Maintenant que nous voyons mieux comment les trois opérations agissent sur les langages temporisés reconnaissables, nous allons nous plonger sur une définition formelle de trois opérations sur les langages temporisés quelconques visant à correspondre avec celles sur les automates temporisés. En effet, plus tard, nous allons avoir besoin d'avoir cette opération non plus pour un langage temporisé reconnaissable, mais pour un langage temporisé quelconque (voir section III.1).

II.3 Extension aux langages temporisés

Comme nous l'avons vu dans la partie précédente, chaque automate temporisé définit en fait une famille de langages temporisés, chacun étant donné par la valeur initiale des horloges de l'automate. Cette remarque est très importante et est à la base de tout ce qui suit. En effet, ceci nous amène à définir la notion de générateur temporisé :

Définition 4 *Soit n un entier. Un* générateur temporisé *sur n horloges est une application de $\mathbb{T}^n$ dans $\mathcal{P}((A \times \mathbb{T})^\infty)$.*

Nous avons aussi vu qu'il était utile (voire nécessaire ?) d'associer à chaque automate temporisé une fonction servant à « retenir » les valeurs des horloges à la sortie de l'automate après lecture d'un mot temporisé fini. Mais évidemment, un langage temporisé reconnaissable n'étant pas accepté par un unique automate, il est impossible d'associer à chaque langage temporisé une fonction donnant les valeurs des horloges à la fin de la lecture des mots. Par contre, il est possible de lui associer des fonctions, ces fonctions devant toujours vérifier qu'un mot est dans le langage si et seulement si la valeur de la fonction appliquée au mot n'est pas vide. En effet, ne pas être dans le langage s'exprimerait alors par le fait que la valeur de la fonction est l'ensemble vide. Formellement :

Définition 5
- *Soit L un générateur temporisé sur n horloges et soit $\alpha \in \mathbb{T}^n$. On dit qu'une fonction $\lambda : (A \times \mathbb{T})^* \longrightarrow \mathcal{P}(\mathbb{T}^n)$ est* compatible *avec (L, α) si pour tout $(\sigma, \tau) \in (A \times \mathbb{T})^*$, $(\sigma, \tau) \in L(\alpha) \iff \lambda(\sigma, \tau) \neq \varnothing$.*
- *Une* contrainte *avec n horloges est une application de $\mathbb{T}^n$ dans l'ensemble des fonctions de $(A \times \mathbb{T})^*$ dans $\mathcal{P}(\mathbb{T}^n)$.*
- *Un* générateur contraint parfait *sur n horloges est un couple (L, Λ) avec L générateur temporisé et Λ contrainte et pour tout $\alpha \in \mathbb{T}^n$, $\Lambda(\alpha)$ est compatible avec $L(\alpha)$. On note $\mathcal{G}cp_n$ l'ensemble des générateurs contraints parfaits sur n horloges.*

On considérera que pour tout $n \leq k$, $\mathcal{G}cp_n \subseteq \mathcal{G}cp_k$, les horloges supplémentaires ne jouant aucun rôle.

REMARQUE : *Quel que soit L, générateur temporisé sur n horloges, il existe Λ contrainte telle que (L, Λ) soit un générateur contraint parfait. En effet, il suffit de prendre Λ tel que pour tout $\alpha \in \mathbb{T}^n$, $\Lambda(\alpha)$ soit la fonction qui à tout $(\sigma, \tau) \in (A \times \mathbb{T})^*$ associe $\mathbb{T}^n$ si $(\sigma, \tau) \in L(\alpha)$, $\varnothing$ sinon.*

Définition 6 *On définit sur l'ensemble $\mathcal{G}cp_n$ les opérations $+$, $\dot{}_C$, $\overset{*}{}^C$ et $\overset{\omega}{}^C$ de la manière décrite ci-après. Soient (L, Λ) et (L', Λ') des générateurs contraints parfaits :*

- $(L,\Lambda) + (L',\Lambda') = (K,\Gamma)$ *avec pour tout* $\alpha \in \mathbb{T}^n$, $\begin{cases} K(\alpha) = L(\alpha) \cup L'(\alpha) \\ \Gamma(\alpha) = \Lambda(\alpha) \cup \Lambda'(\alpha) \end{cases}$
  où $\Lambda(\alpha) \cup \Lambda'(\alpha)$ *désigne la fonction qui à un mot temporisé* u *associe* $\Lambda(\alpha)(u) \cup \Lambda'(\alpha)(u)$.
- $(L,\Lambda) \dot{}_C (L',\Lambda') = (K,\Gamma)$ *avec pour tout* $\alpha \in \mathbb{T}^n$,

$$\begin{cases} K(\alpha) = L_\omega(\alpha) \cup \{u.v \mid u \in L_*(\alpha) \text{ et } \exists \beta \in \Lambda(\alpha)(u)\ v\backslash_u \in L'(\beta)\} \\ \\ \Gamma(\alpha) = \bigcup\limits_{u=v.w} \quad \bigcup\limits_{\beta \in (\Lambda(\alpha))(v)} (\Lambda'([C\leftarrow 0]\beta))(w\backslash_v) \end{cases}$$

- $(L,\Lambda)^{\overset{*}{C}} = (K,\Gamma)$ *avec pour tout* $\alpha \in \mathbb{T}^n$,

$$\begin{cases} K(\alpha) = \bigcup\limits_{i\in\mathbb{N}^*} L^i(\alpha) \\ \\ \Gamma(\alpha) = \bigcup\limits_{i\in\mathbb{N}^*} \Lambda^i(\alpha) \end{cases}$$

  avec $(L^i, \Lambda^i) = (L,\Lambda) \dot{}_C (L^{i-1}, \Lambda^{i-1})$ *si* $i > 1$ *et* $(L^1, \Lambda^1) = (L, \Lambda)$.
- $(L,\Lambda)^{\overset{\omega}{C}} = (K,\Gamma)$ *avec pour tout* $\alpha \in \mathbb{T}^n$,

$$\begin{cases} K(\alpha) = L^{\overset{*}{C}}{}_\omega(\alpha) \cup \{u_0.u_1 \cdots \mid u_0 \in L_*(\alpha) \text{ et } \forall i \in \mathbb{N}^*, \\ \qquad\qquad \exists \alpha_i \in \Lambda(\alpha_{i-1})(u_{i-1}\backslash_{u_0\cdots u_{i-2}})\ u_i \in L_*(\alpha_i)\} \\ \\ \Lambda(\alpha) = \Lambda^{\overset{*}{C}}(\alpha) \end{cases}$$

  avec $(L^{\overset{*}{C}}, \Lambda^{\overset{*}{C}}) = (L,\Lambda)^{\overset{*}{C}}$.

Nous allons voir que ces définitions correspondent bien à ce qui a été dit sur les langages reconnaissables. Pour ce faire, nous allons revoir à quoi correspondent toutes ces définitions dans le cas des automates temporisés pour pouvoir insérer le tout dans le cadre théorique de l'étude en cours.

Soit $\mathcal{A}$ un automate temporisé ayant n horloges. Soit $\lambda^\alpha_\mathcal{A}$ la fonction suivante :

$$\lambda^\alpha_\mathcal{A} : \quad u \in (A \times \mathbb{T})^* \quad \longmapsto \quad \{\beta \in \mathbb{T}^n \mid \text{il existe une exécution acceptante } u \text{ dans } \mathcal{A}^\alpha \text{ telle que la valuation des horloges à la fin de l'exécution soit } \beta\}$$

Cette fonction vérifie la propriété que pour tout mot temporisé u,

$$u \in \mathcal{T}_*(\mathcal{A}^\alpha) \iff \lambda^\alpha_\mathcal{A}(u) \neq \varnothing$$

Soit $\mathcal{T}_\mathcal{A}$ la fonction qui à $\alpha \in \mathbb{T}^n$ associe $\mathcal{T}(\mathcal{A}^\alpha)$. D'après ce que nous venons de voir, $\lambda_\mathcal{A}^\alpha$ est bien une fonction compatible avec $(\mathcal{T}_\mathcal{A}, \alpha)$.
Soit $\Lambda_\mathcal{A}$ la fonction qui à $\alpha \in \mathbb{T}^n$ associe $\lambda_\mathcal{A}^\alpha$. Alors, d'après ce qui précède, $(\mathcal{T}_\mathcal{A}, \Lambda_\mathcal{A})$ est un générateur contraint parfait . Par la suite, nous confondrons $\mathcal{T}_\mathcal{A}$ avec le couple $(\mathcal{T}_\mathcal{A}, \Lambda_\mathcal{A})$ ($\Lambda_\mathcal{A}$ est la fonction *canonique* associée avec $\mathcal{T}_\mathcal{A}$ ou avec $\mathcal{A}$).
La fonction canonique associée à ε, notée Λ_ε, est l'application qui à $\alpha \in \mathbb{T}^n$ associe la fonction qui à ε associe $\{\alpha\}$ et à tout autre mot temporisé associe $\emptyset$.

La proposition qui suit est l'aboutissement de toutes les définitions qui viennent d'être faites. Elle exprime exactement ce que nous avons annoncé.

Proposition 10 *Soient $\mathcal{A}$ et $\mathcal{B}$ des automates temporisés ayant pour ensemble d'horloges X. Soit $C \subseteq X$. Alors on a :*

$$\begin{aligned} \mathcal{T}_\mathcal{A} \underset{C}{\dot{}} \mathcal{T}_\mathcal{B} &= \mathcal{T}_{\mathcal{A} \underset{C}{\dot{}} \mathcal{B}} \\ \mathcal{T}_\mathcal{A}^{\overset{*}{C}} &= \mathcal{T}_{\mathcal{A}^{\overset{*}{C}}} \\ \mathcal{T}_\mathcal{A}^{\overset{\omega}{C}} &= \mathcal{T}_{\mathcal{A}^{\overset{\omega}{C}}} \end{aligned}$$

Preuve : Toutes les constructions ont été faites dans ce but. □

Plus tard, ces opérations vont servir à résoudre des systèmes d'équations. Pour simplifier les écritures et résoudre ce problème, nous aurons besoin de quelques propriétés qui vont être montrées à la section suivante.

II.4 Quelques propriétés

L'opération $\underset{C}{\dot{}}$ possède certaines propriétés intéressantes d'associativité et de distributivité. Ces propriétés seront surtout utiles lorque nous allons chercher à résoudre les systèmes d'équations linéaires temporisées à la section III.

Proposition 11 *Soient (L_1, Λ_1), (L_2, Λ_2) et (L_3, Λ_3) des générateurs contraints parfaits . Soit X leur ensemble d'horloges commun et soit $C \subseteq X$ et $D \subseteq X$. Alors, les trois propriétés suivantes sont vérifiées (Λ_ε désigne la fonction canonique associée à ε) :*

1. $(L_1, \Lambda_1) \underset{C}{\dot{}} ((L_2, \Lambda_2) + (L_3, \Lambda_3)) = ((L_1, \Lambda_1) \underset{C}{\dot{}} (L_2, \Lambda_2)) + ((L_1, \Lambda_1) \underset{C}{\dot{}} (L_3, \Lambda_3))$
2. $((L_1, \Lambda_1) + (L_2, \Lambda_2)) \underset{C}{\dot{}} (L_3, \Lambda_3) = ((L_1, \Lambda_1) \underset{C}{\dot{}} (L_3, \Lambda_3)) + ((L_2, \Lambda_2) \underset{C}{\dot{}} (L_3, \Lambda_3))$
3. $(L_1, \Lambda_1) \underset{C}{\dot{}} ((L_2, \Lambda_2) \underset{D}{\dot{}} (L_3, \Lambda_3)) = ((L_1, \Lambda_1) \underset{C}{\dot{}} (L_2, \Lambda_2)) \underset{D}{\dot{}} (L_3, \Lambda_3)$
4. $(L_1, \Lambda_1) \underset{C}{\dot{}} (\varepsilon, \Lambda_\varepsilon) \underset{D}{\dot{}} (L_2, \Lambda_2) = (L_1, \Lambda_1) \underset{C \cup D}{\dot{}} (L_2, \Lambda_2)$

REMARQUE : *Dans la preuve qui va suivre, pour éviter des lourdeurs dues à des notations, si* $(L, \Lambda) = (L_1, \Lambda_1) \dot{_C} (L_2, \Lambda_2)$, *on notera* $L = L_1 \dot{_C} L_2$ *et* $\Lambda = \Lambda_1 \dot{_C} \Lambda_2$.

Preuve : La construction initiale de l'opération $\dot{_C}$ donne immédiatement ces propriétés si les L_i sont des langages reconnaissables et si Λ_i est leur fonction canonique.
Dans le cas général :

1. Ce que nous allons démontrer ici est une propriété de distributivité à droite de $+$ par rapport à $\dot{_C}$. Soient (L, Λ) le membre de gauche de l'égalité que nous voulons montrer et (L', Λ') le membre de droite.

$$\begin{aligned} L(\alpha) &= \{u.v \mid u \in L_1(\alpha) \text{ et } \exists\beta \in \Lambda_1(\alpha)(u)\ v\backslash_u \in L_2([C\leftarrow 0]\beta) \cup L_3([C\leftarrow 0]\beta)\} \\ &= L'(\alpha) \\ \Lambda(\alpha) &= \Lambda'(\alpha) \quad \text{(facile à montrer)} \end{aligned}$$

2. Ceci est une autre propriété de distributivité : la distributivité à droite de $+$ par rapport à $\dot{_C}$. Soient (L, Λ) le membre de gauche de l'égalité que nous voulons montrer et (L', Λ') le membre de droite.

$$\begin{aligned} L(\alpha) &= \{u.v \mid u \in L_1(\alpha) \cup L_2(\alpha) \text{ et } \exists\beta \in (\Lambda_1 \cup \Lambda_2)(\alpha)(u)\ v\backslash_u \in L_3([C\leftarrow 0]\beta)\} \\ &= L'(\alpha) \\ \Lambda(\alpha) &= \Lambda'(\alpha) \quad \text{(facile à montrer)} \end{aligned}$$

3. Ce que nous allons montrer ici est une propriété d'associativité des opérations $\dot{_C}$ qui est indispensable pour les résolutions d'équations qui vont être faites à la partie III.1. Soient (L, Λ) le membre droit de l'égalité que nous voulons montrer et (L', Λ') le membre gauche.

$$\begin{aligned} L(\alpha) &= \{u.v \mid u \in L_1(\alpha) \text{ et } \exists\beta \in \Lambda_1(\alpha)(u)\ v\backslash_u \in (L_2 \dot{_D} L_3)([C\leftarrow 0]\beta)\} \\ &= \{u.v.w \mid u \in L_1(\alpha),\ \exists\beta \in \Lambda_1(\alpha)(u)\ v\backslash_u \in L_2([C\leftarrow 0]\beta), \\ &\qquad \exists\gamma \in \Lambda_2([C\leftarrow 0]\beta)(v\backslash_u)\ w\backslash_{u.v} \in L_3([D\leftarrow 0]\gamma)\} \end{aligned}$$

$$\begin{aligned} L'(\alpha) &= \{u.w \mid u \in (L_1 \dot{_C} L_2)(\alpha) \text{ et } \exists\gamma \in (\Lambda_1 \dot{_C} \Lambda_2)(\alpha)(u)\ w\backslash_u \in L_3([D\leftarrow 0]\gamma)\} \\ &= \{u.v.w \mid u \in L_1(\alpha),\ \exists\beta \in \Lambda_1(\alpha)(u)\ v\backslash_u \in L_2([C\leftarrow 0]\beta), \\ &\qquad \exists\gamma \in (\Lambda_1 \dot{_C} \Lambda_2)(\alpha)(u.v)\ w\backslash_{uv} \in L_3([D\leftarrow 0]\gamma)\} \end{aligned}$$

Nous obtenons tout de suite que $L(\alpha) \subseteq L'(\alpha)$.

Réciproquement, supposons que $z \in L'(\alpha)$.
Nous pouvons écrire $z = u.v.w$ avec :

$$\left\{ \begin{array}{l} u \in L_1(\alpha) \\ \exists\beta \in \Lambda_1(\alpha)(u)\ v\backslash_u \in L_2([C\leftarrow 0]\beta) \\ \exists\gamma \in (\Lambda_1 \dot{_C} \Lambda_2)(\alpha)(u.v)\ w\backslash_{u.v} \in L_3([D\leftarrow 0]\gamma) \end{array} \right.$$

Il s'en déduit : il existe $u_1.v_1 = u.v$ tel que $\gamma \in \Lambda_2([C{\leftarrow}0]\beta')(v_1\backslash_{u_1})$ avec $\beta' \in \Lambda_1(\alpha)(u_1)$.
Ainsi, $z = u_1.v_1.w$ avec :

$$\left\{\begin{array}{l} u_1 \in L_1(\alpha) \text{ car } \Lambda_1(\alpha)(u_1) \neq \varnothing \\ \exists\beta' \in \Lambda_1(\alpha)(u_1)\; v_1\backslash_{u_1} \in L_2([C{\leftarrow}0]\beta') \text{ car } \Lambda_2([C{\leftarrow}0]\beta')(v_1\backslash_{u_1}) \neq \varnothing \\ \exists\gamma \in \Lambda_2([C{\leftarrow}0]\beta')(v_1\backslash_{u_1})\; w\backslash_{u_1.v_1} \in L_3([D{\leftarrow}0]\gamma) \end{array}\right.$$

Donc, $z \in L(\alpha)$.
Il est facile de montrer de la même manière que $\Lambda(\alpha) = \Lambda'(\alpha)$.

4. L'écriture de l'énoncé peut être utilisée car l'associativité des opérateurs $\cdot_C$ a été prouvée juste avant. Soient (L, Λ) le membre gauche de l'égalité que l'on veut prouver et (L', Λ') le membre droit.

$$\begin{array}{rcl} L(\alpha) & = & \{u.v.w \mid u \in L_1(\alpha),\, \exists\beta \in \Lambda_1(\alpha)(u)\; v\backslash_u \in \varepsilon([C{\leftarrow}0]\beta) \\ & & \qquad \text{et } \exists\gamma \in \Lambda_\varepsilon([C{\leftarrow}0]\beta)(v\backslash_u)\; w\backslash_{u.v} \in L_2([C{\leftarrow}0]\gamma)\} \\ & = & \{u.w \mid u \in L_1(\alpha) \text{ et } \exists\beta \in \Lambda_1(\alpha)(u)\; w\backslash_u \in L_2([(C\cup D){\leftarrow}0]\beta)\} \\ & = & L'(\alpha) \end{array}$$

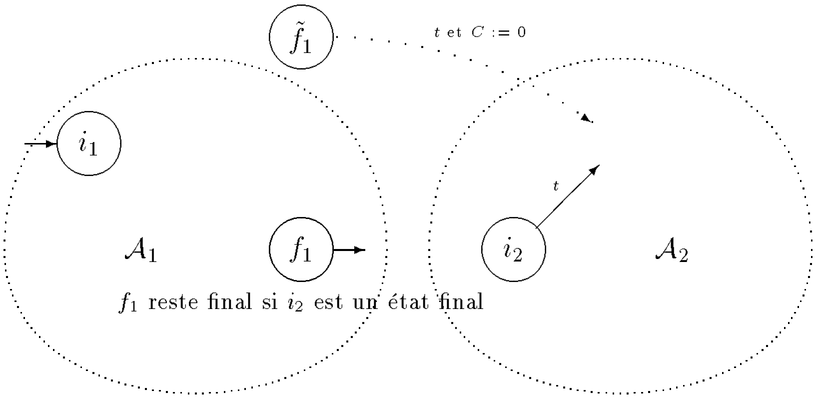
De la même façon, il est facile de montrer que $\Lambda(\alpha) = \Lambda'(\alpha)$.

Ces propriétés sont des propriétés élémentaires sur ces opérations, qui vont être utiles dans la section III.1 lorsque l'on va chercher à résoudre les systèmes d'équations linéaires temporisées. □

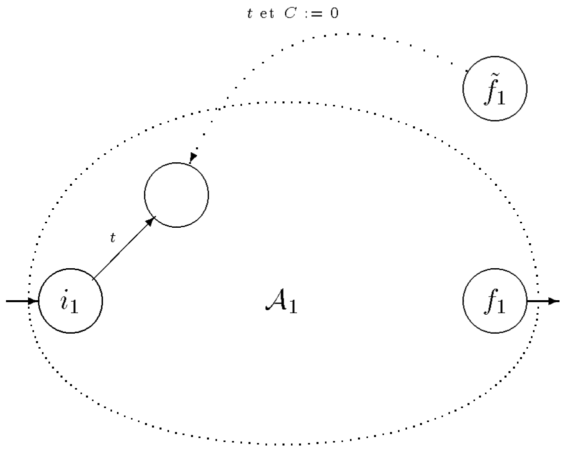
II.5 Propriétés de clôture

Ces opérations sur les automates semblent faire sortir de la classe des automates temporisés. En effet, nous rajoutons quelques ε-transitions. Mais, en fait, ces transitions ne sont pas nécessaires et elles peuvent toutes être retirées (dans le cas général, les ε-transitions augmentent le pouvoir d'expression des automates temporisés, mais, ici, bien que les résultats de [DGP97] ne s'appliquent pas, nous sommes dans un cas particulier où elles peuvent être retirées). En effet, les ε-transitions qui ont été rajoutées sont un petit peu spéciales : elles sont «urgentes», c'est à dire qu'on ne peut pas attendre puis prendre ces ε-transitions. C'est pour cette raison qu'elles vont pouvoir être retirées.

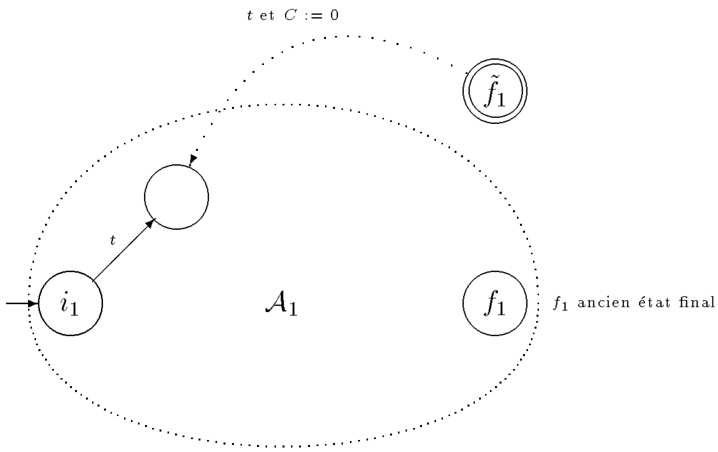
Proposition 12 *La classe des automates temporisés est close par les opérations précédemment définies ($\cdot_C$, $^{\overset{*}{C}}$ et $^{\overset{\omega}{C}}$).*



(a) $\mathcal{A}_{\dot{C}}$



(b) $\mathcal{A}_{\overset{*}{C}}$



(c) $\mathcal{A}_{\overset{\omega}{C}}$

FIG. 12 – *Automates reconnaissant les opérations de composition* $\dot{C}$, $\overset{*}{C}$ *et* $\overset{\omega}{C}$

Preuve : Soient $\mathcal{A}_1 = \langle A, Q_1, I_1, F_1, R_1, T_1, C_1\rangle$ et $\mathcal{A}_2 = \langle A, Q_2, I_2, F_2, R_2, T_2, C_2\rangle$ des automates temporisés.

- $\mathcal{A}_{\dot{C}} = \langle A, Q, I, F, R, T, X\rangle$ où

$$\left\{\begin{array}{lcl} Q & = & Q_1 \cup Q_2 \cup \tilde{F}_1 \\ I & = & \left\{\begin{array}{ll} I_1 & \text{si } I_1 \cap F_1 = \varnothing \\ I_1 \cup I_2 & \text{sinon} \end{array}\right. \\ F & = & \left\{\begin{array}{ll} F_2 & \text{si } I_2 \cap F_2 = \varnothing \\ F_1 \cup F_2 & \text{sinon} \end{array}\right. \\ R & = & R_1 \cup R_2 \\ T & = & T_1 \cup T_2 \cup \{\langle q, a, \tilde{f}, r \cup C, g\rangle \mid \langle q, a, f, r, g\rangle \in T_1\} \\ & & \qquad \cup\{\langle \tilde{f}, a, q, r, g\rangle \mid \exists i \in I_2 \, \langle i, a, q, r, g\rangle \in T_2\} \end{array}\right.$$

- $\mathcal{A}_{\overset{*}{C}} = \langle A, Q, I, F, R, T, X\rangle$ où

$$\left\{\begin{array}{lcl} Q & = & Q_1 \cup \tilde{F}_1 \\ I & = & I_1 \\ F & = & F_1 \\ R & = & R_1 \\ T & = & T_1 \cup \{\langle q, a, \tilde{F}, r \cup C, g\rangle \mid \langle q, a, f, r, g\rangle \in T_1\} \\ & & \qquad \cup\{\langle \tilde{f}, a, q, r, g\rangle \mid \exists i \in I_1 \, \langle i, a, q, r, g\rangle \in T_1\} \end{array}\right.$$

- $\mathcal{A}_{\overset{\omega}{C}} = \langle A, Q, I, F, R, T, X\rangle$ où

$$\left\{\begin{array}{lcl} Q & = & Q_1 \cup \tilde{F}_1 \\ I & = & I_1 \\ F & = & \varnothing \\ R & = & R_1 \cup \tilde{F}_1 \\ T & = & T_1 \cup \{\langle q, a, \tilde{f}, r \cup C, g\rangle \mid \langle q, a, f, r, g\rangle \in T_1\} \\ & & \qquad \cup\{\langle \tilde{f}, a, q, r, g\rangle \mid \exists i \in I_1 \, \langle i, a, q, r, g\rangle \in T_1\} \end{array}\right.$$

Lemme 13 *Les trois égalités suivantes sont vraies :*

$$\begin{array}{lcl} \mathcal{T}_{\mathcal{A}_{\dot{C}}} & = & \mathcal{T}_{\mathcal{A}_1} \, {}_{\dot{C}} \, \mathcal{T}_{\mathcal{A}_2} \\ \mathcal{T}_{\mathcal{A}_{\overset{*}{C}}} & = & \mathcal{T}_{\mathcal{A}_1}^{\overset{*}{C}} \\ \mathcal{T}_{\mathcal{A}_{\overset{\omega}{C}}} & = & \mathcal{T}_{\mathcal{A}_1}^{\overset{\omega}{C}} \end{array}$$

Preuve : La preuve va être rédigée pour l'opérateur ${}_{\dot{C}}$. Les autres preuves sont similaires et ne sont donc pas détaillées.
Les deux implications vont être montrées séparément.

- Soit $u \in (\mathcal{T}_{\mathcal{A}_1} \, {}_{\dot{C}} \, \mathcal{T}_{\mathcal{A}_2})(\alpha)$.

Par définition de $\cdot_C$, il existe $v \in \mathcal{T}_{\mathcal{A}_1}(\alpha)$ et $w\backslash_v \in \mathcal{T}_{\mathcal{A}_2}([C\leftarrow 0]\beta)$ avec $\beta \in \lambda_{\mathcal{A}_1}(\alpha)(v)$ tels que $u = v.w$. Il existe donc une exécution $(\langle q_i, v_i\rangle)_{0\leq i\leq m}$ pour v dans $\mathcal{A}_1^\alpha$ tel que $(v_m(x_i))_{0\leq i\leq n} = \beta$. Il existe aussi $(\langle q'_j, v'_j\rangle)_{0\leq j\leq m'}$, exécution acceptante $w\backslash_v$ dans $\mathcal{A}_2^{[C\leftarrow 0]\beta}$.

Si $w = \varepsilon$, alors $q_m \in F_1 \subseteq F$, et donc $(\langle q_i, v_i\rangle)_{0\leq i\leq m}$ est une exécution acceptante pour u. Si $v = \varepsilon$, alors $\beta = \alpha$ et $q'_0 \in I$, donc $(\langle q'_j, v'_j\rangle)_{0\leq j\leq m'}$ est une exécution acceptante pour u. Sinon, $(\langle q_i, v_i\rangle)_{0\leq i\leq m-1}(\langle \tilde{q}_m, [C\leftarrow 0]v_m\rangle)(\langle q'_j, v'_j\rangle)_{1\leq j\leq m'}$ est une exécution acceptante pour u. Dans tous les cas, $u \in \mathcal{T}_{\mathcal{A}_{\dot{C}}}(\alpha)$.

- Réciproquement, soit $u \in \mathcal{T}_{\mathcal{A}_{\dot{C}}}(\alpha)$.Soit $(\langle q_i, v_i\rangle)_{0\leq i}$ une exécution acceptante pour u. Si tous les états q_i viennent de l'automate $\mathcal{A}_1$, cela signifie que $\varepsilon \in \mathcal{T}(\mathcal{A}_2)$ et que $u \in \mathcal{T}_{\mathcal{A}_1}(\alpha)$. Donc $u \in (\mathcal{T}_{\mathcal{A}_1} \cdot_C \mathcal{T}_{\mathcal{A}_2})(\alpha)$. Sinon, soit j le plus petit indice tel que q_j ne soit pas un élément des états de $\mathcal{A}_1$. Alors $q_j \in \tilde{F}_1$, disons que $q_j = \tilde{f}$. Alors si u_k est le préfixe de u de longueur k et si u^k vérifie $u = u_k.u^k$, $u_j \in \mathcal{T}_{\mathcal{A}_1}(\alpha)$ et $u^j\backslash_{u_j} \in \mathcal{T}_{\mathcal{A}_2}(\beta)$ avec $\beta = (v_j(x_i))_{0\leq i\leq n}$. Donc $u \in (\mathcal{T}_{\mathcal{A}_1} \cdot_C \mathcal{T}_{\mathcal{A}_2})(\alpha)$.

□

Ce lemme montre donc que la classe des automates temporisés sans ε-transition est close par les trois opérations $\cdot_C$, $^{\overset{*}{C}}$ et $^{\overset{\omega}{C}}$ car pour tous les automates $\mathcal{A}_1$ et $\mathcal{A}_2$, il existe des automates $\mathcal{B}$, $\mathcal{C}$ et $\mathcal{D}$ tels que pour tout $\alpha \in \mathbb{T}^n$, $\mathcal{T}_\mathcal{B}(\alpha) = (\mathcal{T}_{\mathcal{A}_1} \cdot_C \mathcal{T}_{\mathcal{A}_2})(\alpha)$, $\mathcal{T}_\mathcal{C}(\alpha) = \mathcal{T}_{\mathcal{A}_1}^{\overset{*}{C}}(\alpha)$ et $\mathcal{T}_\mathcal{D}(\alpha) = \mathcal{T}_{\mathcal{A}_1}^{\overset{\omega}{C}}(\alpha)$. □

II.6 Composition d'automates temporisés

Il est maintenant temps de faire une synthèse de ce qui vient d'être étudié et de conclure cette partie sur la composition des automates temporisés.

Définition 7 *Soit $n \in \mathbb{N}^*$. On note $\langle\delta a\rangle_I$, avec I une partie de $\mathbb{T}^n$, le générateur temporisé qui à $\alpha \in \mathbb{T}^n$ associe $\{(a,t) \mid \alpha + t \in I\}$. On note ε le générateur temporisé qui à toute valeur associe le mot vide ε, qui est en fait la suite vide de $(A \times \mathbb{T})^*$.*

REMARQUE : *Le générateur temporisé $\langle\delta a\rangle_I$ est un générateur temporisé reconnu par automate temporisé, i.e. il existe un automate $\mathcal{A}$ tel que pour tout $\alpha \in \mathbb{T}^n$, $(\langle\delta a\rangle_I)(\alpha) = \mathcal{T}_\mathcal{A}(\alpha)$ (cf* FIG. *13). Cet automate sera noté par la suite $\mathcal{A}_{\langle\delta a\rangle_I}$. La contrainte canonique qui lui est associée est la fonction qui à α associe $\{\alpha + t \mid t \in \mathbb{T}\} \cap I$. Elle est notée $\Lambda_{\langle\delta a\rangle_I}$.*

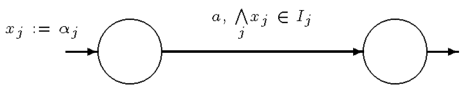


FIG. 13 – *Automate $\mathcal{A}_{\langle\delta a\rangle_I}$*

REMARQUE : *Le langage temporisé ε est reconnu par l'automate de la* FIG. *14. Cet automate sera noté $\mathcal{A}_\varepsilon$. La contrainte canonique qui lui est associée est la fonction qui à tout α associe la fonction $\varepsilon \mapsto \{\alpha\}$ et $u \neq \varepsilon \mapsto \emptyset$.*

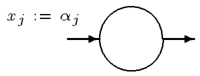


FIG. 14 – *Automate* $\mathcal{A}_\varepsilon$

Définition 8 *Soit* $\mathcal{M}od(A)$ *le plus petit ensemble engendré par la grammaire suivante :*

$$\mathcal{A} ::= \mathcal{A}_\varepsilon \mid \mathcal{A}_{\langle \delta a \rangle_I} \mid \mathcal{B}_1 + \mathcal{B}_2 \mid \mathcal{B}_1 \underset{C}{\cdot} \mathcal{B}_2 \mid \mathcal{B}^{\overset{*}{C}} \mid \mathcal{B}^{\overset{\omega}{C}}$$

avec $a \in A$, $I \subseteq \mathbb{T}^n$ *et* $C \subseteq X$ *est un ensemble d'horloges.*

Corollaire 14 *Soit* A *un alphabet. Alors quelque soit* $\mathcal{A} \in \mathcal{M}od(A)$*, il existe un automate temporisé* $\mathcal{B}$ *vérifiant :* $\forall \alpha \in \mathbb{T}^n\ \mathcal{T}(\mathcal{A}^\alpha) = \mathcal{T}(\mathcal{B}^\alpha)$.

Preuve : C'est un corollaire immédiat de ce qui précède. □

Nous avons donc défini des opérations de composition sur la classe des automates temporisés. Celles-ci permettent de relier entre eux de petits automates et de représenter de cette manière certains gros systèmes.

REMARQUE : *Nous allons finir ce chapitre en comparant de ce qui vient d'être fait avec ce qu'ont défini Asarin, Caspi et Maler dans [ACM97] et [Asa98]. Le modèle de comportement temporisé choisi par ces derniers est le signal alors que ce modèle est strictement moins expressif que celui des mots temporisés choisi ici. En outre, les opérations de concaténation qu'ils ont définies sur les signaux correspondent en fait à* $\underset{\emptyset}{\cdot}$ *et* $\underset{x}{\cdot}$ *pour les automates ayant une seule horloge. Les opérations de concaténation que nous avons étudiées ici apparaissent donc comme une extension des opérations de [ACM97] (grâce à nos opérations, il est possible de composer des automates ayant un nombre quelconque d'horloges).*

Chapitre III

Décomposition des automates temporisés

Dans tout ce qui suit $X = \{x_1, \cdots, x_n\}$ est un ensemble d'horloges et α est un élément de $\mathbb{T}^n$. Le but de ce chapitre est de faire la réciproque de ce qui a été fait dans le chapitre précédent. Nous allons donc essayer de décomposer tout automate temporisé en automates de base, ces automates de base étant composés les uns aux autres par les opérations définies dans le chapitre précédent : $+$, $\underset{C}{\cdot}$, $\overset{*}{C}$ et $\overset{\omega}{C}$.

III.1 Résolution de systèmes d'équations linéaires temporisées

Lorsque nous étudions les langages rationnels et les automates finis classiques, une solution pour trouver une expression rationnelle exprimant le langage accepté par un automate fini est de résoudre un système d'équations «linéaires». Voici un exemple illustrant cette méthode de résolution.
EXEMPLE : L'automate de la FIG. 15 définit le système d'équations suivant :

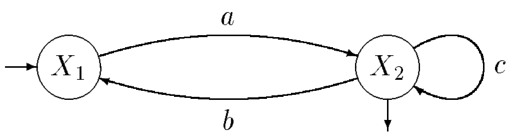


FIG. 15 – *Automate fini*

$$\left\{\begin{array}{lcl} X_1 & = & a.X_2 \\ X_2 & = & \varepsilon + b.X_1 + c.X_2 \end{array}\right.$$

La solution de ce système est :

$$\left\{\begin{array}{lcl} X_1 & = & a.(ba+c)^* \\ X_2 & = & (ba+c)^* \end{array}\right.$$

Une expression rationnelle décrivant le langage accepté par l'automate initial est alors $a.(ba+c)^*$.

Une telle méthode a aussi été utilisée dans [Asa98] pour trouver une expression rationnelle temporisée exprimant le langage temporisé accepté par un automate temporisé ayant une horloge. Nous allons transformer quelque peu cette méthode pour trouver une manière de décomposer les automates temporisés grâce aux opérations $\dot{}_C$, $^{\overset{*}{C}}$ et $^{\overset{\omega}{C}}$. Ici, les systèmes que nous allons être amenés à résoudre sont un petit peu plus compliqués que pour les automates finis car ils font intervenir plus d'opérateurs. Nous allons tout de suite passer à la définition exacte de ces systèmes puis nous passerons à la méthode de résolution.

Définition 9 *Un* système d'équations linéaires temporisées à n *horloges est un système de la forme :*

$$Z \;=\; \sum_{C\subseteq X} B_C \,\dot{}_C\, Z$$

où $Z \in \mathcal{M}_{1,m}(\mathcal{G}cp_n)$, $B_C \in \mathcal{M}_{m,n}(\mathcal{G}cp_n)$ [1].
Une solution de ce système est un vecteur de générateurs contraints parfaits de mots finis $Z = (Z_i)_{i\in\mathbb{N}_m}$ *vérifiant :*

$$\forall i \in \mathbb{N}_m \quad Z_i(\alpha) = \left(\sum_{C\subseteq X}\sum_{j=1}^{n} B_C^{i,j} \,\dot{}_C\, Z_j\right)(\alpha)$$

si $B_C = \left(B_C^{i,j}\right)_{i\in\mathbb{N}_m\, j\in\mathbb{N}_n}$.

Il faut remarquer que c'est ici qu'intervient le fait qu'il était nécessaire de définir, dans le dernier chapitre, les opérations, pour un générateur contraint parfait quelconque et pas seulement pour des générateurs contraints parfaits associés à des automates temporisés. En effet, nous ne pouvons présumer à l'avance de la nature de la solution : nous ne pouvons pas supposer avant de résoudre les équations que la solution sera reconnaissable. Donc ici, la solution est a priori quelconque (c'est à dire reconnaissable ou pas).

Le lemme suivant est une sorte de lemme d'interversion de limites (comme pour les séries entières).

Lemme 15 *Soit* (L_1, Λ_1) *et* (L_2, Λ_2) *des générateurs contraints parfaits de mots finis. Nous avons pour tout* $\alpha \in \mathbb{T}^n$ *:*

$$\left((L_1,\Lambda_1)^{\overset{*}{C}} \,\dot{}_C\, (L_2,\Lambda_2)\right)(\alpha) = \left(\sum_{i=1}^{\infty}\left((L_1,\Lambda_1)^{(\dot{}_C)^i} \,\dot{}_C\, (L_2,\Lambda_2)\right)\right)(\alpha)$$

Preuve : Soit $u \in \left(L_1^{\overset{*}{C}} \,\dot{}_C\, L_2\right)(\alpha)$. Nous pouvons écrire $u = v.w$ avec $v \in L_1^{\overset{*}{C}}(\alpha)$ et $w\backslash_v \in L_2(\alpha')$ où $\alpha' \in \Lambda_1^{\overset{*}{C}}(\alpha)(v)$. Il existe $i \in \mathbb{N}$ tel que $\alpha' \in \Lambda_1^{(\dot{}_C)^i}(\alpha)(v)$.

1. $\mathcal{M}_{k,l}(P)$ désigne l'ensemble des matrices $k \times l$ à valeurs dans P

Alors $v \in L_1^{(\cdot_C)^i}(\alpha)$, donc $u \in (L_1^{(\cdot_C)^i} \cdot_C L_2)(\alpha)$.
De même, pour tout $\alpha \in \mathbb{T}^n$ et pour tout mot temporisé u,

$$\left(\Lambda_1^{\overset{*}{C}} \cdot_C \Lambda_2\right)(\alpha)(u) \subseteq \left(\sum_{i=1}^{\infty} \left(\Lambda_1{}^{(\cdot_C)^i} \cdot_C \Lambda_2\right)\right)(\alpha)(u)$$

La réciproque est évidente. □

Lemme 16 *Soient β et γ des générateurs contraints parfaits .*
La solution minimale de l'équation

$$Z = \beta + \gamma \cdot_C Z \tag{1}$$

est $Z = \gamma^{\overset{*}{C}} \cdot_C \beta + \beta$.

Preuve : Tout d'abord, il est facile de voir que $Z = \gamma^{\overset{*}{C}} \cdot_C \beta + \beta$ est solution de l'équation 1. Notons $Z = (L, \Lambda)$.
Soit $Y = (L', \Lambda')$ une autre solution de cette équation. Pour prouver que Z est la solution minimale, nous allons montrer que $\forall \alpha \in \mathbb{T}^n$, $L(\alpha) \subseteq L'(\alpha)$. Soit $u \in L(\alpha)$. Nous avons $\gamma^{\overset{*}{C}} \cdot_C \beta = \sum_{i=1}^{\infty} \gamma^{(\cdot_C)^i} \cdot_C \beta$ d'après le lemme précédent. Nous allons montrer le résultat par induction sur la longueur des mots temporisés. Si $u \in \beta(\alpha)$, alors immédiatement, $u \in L'(\alpha)$ car L' vérifie l'équation (1). Sinon, on peut écrire $u = v.w$ avec $v \in \gamma(\alpha)$ et $w\backslash_v \in (\gamma^{(\cdot_C)^i} \cdot_C \beta)(\alpha')$ où $\alpha' \in \lambda_\gamma^\alpha(v)$.
Par hypothèse d'induction, $w \in L'(\alpha')$, donc $u \in (\gamma \cdot_C L')(\alpha) \subseteq L'(\alpha)$. La base de l'induction est donnée par les éléments de β, qui sont à la fois dans L et dans L'.
De la même façon, nous montrons que pour tout $\alpha \in \mathbb{T}^n$ et pour tout mot temporisé u,

$$\Lambda(\alpha)(u) \subseteq \Lambda'(\alpha)(u)$$

□

Corollaire 17 *La solution minimale d'un système linéaire temporisé est une expression rationnelle temporisée et nous savons la calculer.*

REMARQUE : *L'associativité des opérateurs $\cdot_C$ est ici essentielle. En effet, si nous voulons résoudre le système suivant :*

$$\left\{\begin{array}{lcl} Z & = & \alpha_1 + \alpha_2 \cdot_C Z + \alpha_3 \cdot_{C'} Y \\ Y & = & \beta_1 + \beta_2 \cdot_C Z + \beta_3 \cdot_{C'} Y \end{array}\right.$$

nous obtenons successivement :

$$\left\{\begin{array}{lcl} Z & = & (\alpha_2)^{\overset{*}{C}} \cdot_C (\alpha_1 + \alpha_3 \cdot_{C'} Y) + \alpha_1 + \alpha_3 \cdot_{C'} Y \\ Y & = & \beta_1 + \beta_2 \cdot_C Z + \beta_3 \cdot_{C'} Y \end{array}\right.$$

$$\left\{\begin{array}{lcl} Z & = & \alpha_2^{\overset{*}{C}} \underset{C}{\cdot} (\alpha_1 + \alpha_3 \underset{C'}{\cdot} Y) + \alpha_1 + \alpha_3 \underset{C'}{\cdot} Y \\ Y & = & \beta_1 + \beta_2 \underset{C}{\cdot} (\alpha_2^{\overset{*}{C}} \underset{C}{\cdot} (\alpha_1 + \alpha_3 \underset{C'}{\cdot} Y) + \alpha_1 + \alpha_3 \underset{C'}{\cdot} Y) + \beta_3 \underset{C'}{\cdot} Y \end{array}\right.$$

Les opérateurs $\underset{C}{\cdot}$ étant associatifs, ceci peut se réécrire :

$$\left\{\begin{array}{lcl} Z & = & \alpha_2^{\overset{*}{C}} \underset{C}{\cdot} (\alpha_1 + \alpha_3 \underset{C'}{\cdot} Y) \\ Y & = & \beta_1 + \beta_2 \underset{C}{\cdot} \alpha_2^{\overset{*}{C}} \underset{C}{\cdot} \alpha_1 + (\beta_2 \underset{C}{\cdot} \alpha_2^{\overset{*}{C}} \underset{C}{\cdot} \alpha_3) \underset{C'}{\cdot} Y \\ & & + \beta_3 \underset{C'}{\cdot} Y + \beta_2 \underset{C}{\cdot} \alpha_1 + \beta_2 \underset{C}{\cdot} \alpha_3 \underset{C'}{\cdot} Y \end{array}\right.$$

Remarque : *La propriété 11 permet de simplifier les écritures dans les équations précédentes.*

III.2 Application aux automates temporisés

Nous arrivons enfin véritablement à la décomposition des automates temporisés. Elle repose sur la résolution d'un système d'équations linéaires temporisées particulier.

Soit $\mathcal{A} = \langle A, Q, I, F, R, T, C\rangle$ un automate temporisé ayant n horloges. Posons le système d'équations linéaires temporisées suivant

$$\left\{\begin{array}{lcll} Z_q & = & \varepsilon + \sum\limits_{(q,a,q',r,g)\in T} \langle \delta a\rangle_g \underset{r}{;} Z_{q'} & \text{si } q \in F \\ Z_q & = & \sum\limits_{(q,a,q',r,g)\in T} \langle \delta a\rangle_g \underset{r}{;} Z_{q'} & \text{si } q \in Q\backslash F \end{array}\right. \tag{2}$$

Soit pour $q \in Q$, $\mathcal{A}_q = \langle A, Q, q, F, R, T, C\rangle$. Notons L_q la fonction qui à $\alpha \in \mathbb{T}^n$ associe $\mathcal{T}_*({}_\alpha\mathcal{A}_q)$, c'est à dire l'ensemble des mots finis acceptés par $\mathcal{A}_q$ en initialisant les horloges à α et Λ_q la fonction canonique associée à $\mathcal{A}_q$.

Proposition 18 *$((L_q, \Lambda_q))_{q\in Q}$ est la solution minimale de (2).*

Preuve : Soit u un mot temporisé.

$$\begin{array}{rcl} u \in L_q(\alpha) & \Longleftrightarrow & \text{si } u = (a,t).v \text{ alors } \exists (q,a,q',r,g) \in T \\ & & \alpha + t \models g,\ v\backslash_{(a,t)} \in L_{q'}([r\leftarrow 0](\alpha + t)) \\ & \Longleftrightarrow & u = (a,t).v \text{ avec } (a,t) \in (\langle \delta a\rangle_g)(\alpha) \\ & & \text{ et } v\backslash_{(a,t)} \in L_{q'}([r\leftarrow 0]\beta) \text{ où } \beta \in \Lambda_{\langle\delta a\rangle_g}(\alpha)((a,t)) \\ & \Longleftrightarrow & u \in (\langle \delta a\rangle_g \underset{r}{;} L_{q'})(\alpha) \end{array}$$

La preuve utilise les mêmes arguments pour Λ_q. Ainsi, $((L_q, \Lambda_q)y)_{q\in Q}$ est solution du système (2). Il reste à montrer que c'est la solution minimale de (2). Soit $((L_q, \Lambda'_q))_{q\in Q}$ une autre solution de ce système. Soit $u \in L_q(\alpha)$. Nous voulons montrer que $u \in L'_q(\alpha)$. Ceci va être fait par récurrence sur la longueur de u. Si $|u| = 0$, cela signifie que $q \in F$, et donc $\varepsilon = u \in L'_q(\alpha)$. Si $|u| = i+1$, $u = (a,t).v$ avec $(a,t) \in (\langle\delta a\rangle_g)(\alpha)$, et $v\backslash_{(a,t)} \in L_{q'}([C\leftarrow 0]\beta)$ où $\beta \in \Lambda_{\langle\delta a\rangle_g}(\alpha)(a,t)$. Ainsi, $v\backslash_{(a,t)} \in L'_{q'}([C\leftarrow 0]\beta)$ par hypothèse d'induction, et donc $u \in L_q(\alpha)$. Le raisonnement est le même pour Λ_q et Λ'_q. $\square$

Corollaire 19 *Le langage fini accepté par $\mathcal{A}^\alpha$ est donné par $\bigcup_{q\in I} L_q(\alpha) = (\sum_{q\in I} L_q)(\alpha)$.*

Proposition 20 *Soit $\mathcal{A} = \langle A, Q, I, F, R, T, C\rangle$ un automate temporisé. Le langage de mots infinis accepté par $\mathcal{A}$ est donné par la formule suivante :*

$$
\begin{aligned}
\mathcal{T}_\omega(\mathcal{A}^\alpha) &= \bigcup_{r\in R} \mathcal{T}_\omega(\mathcal{A}_r^\alpha) \\
&= \bigcup_{r\in R} \left((L_r, \Lambda_r) \underset{C_r}{\cdot} (L_{r,r}, \Lambda_{r,r})^{\overset{\omega}{C_r}} \right)(\alpha) \\
&= \mathcal{T}\left(\left(\sum_{r\in R} \mathcal{A}_r \underset{C_r}{\cdot} \mathcal{A}_{r,r}^{\overset{\omega}{C_r}} \right)^\alpha \right)
\end{aligned}
$$

où $\mathcal{A}_r = \langle A, Q, I, \varnothing, r, T, X\rangle$, $\mathcal{A}_{r,r} = \langle A, Q, r, \varnothing, r, T, X\rangle$, (L_r, Λ_r) est le générateur contraint parfait associé à $\mathcal{A}_r$ et $(L_{r,r}, \Lambda_{r,r})$ est le générateur contraint parfait associé à $\mathcal{A}_{r,r}$.

Théorème 21 *Soit $\mathcal{A} \in \mathcal{A}ut(A)$. Il existe un automate $\mathcal{B} \in \mathcal{M}od(A)$ tel que pour tout $\alpha \in \mathbb{T}^n$, $L(\mathcal{A}^\alpha) = L(\mathcal{B}^\alpha)$.*

Preuve : Ce théorème est juste une interprétation en termes d'automates de ce qui précède. □

EXEMPLE : Un exemple d'une telle décomposition se trouve à l'annexe A.

REMARQUE : *Pour trouver une décomposition modulaire des automates temporisés, il aurait été possible d'utiliser un algorithme à la Mac Naughton et Yamada en supposant que toutes les transitions arrivant dans un même état mettent à zéro le même ensemble d'horloges.*

Nous sommes donc arrivés au but que nous nous étions fixés : nous avons donné un algorithme de décomposition des automates en automates de base grâce aux opérations $+$, $\underset{C}{\cdot}$, $\overset{*}{C}$ et $\overset{\omega}{C}$. Dans le chapitre suivant, nous verrons que cette décomposition des automates temporisés va nous permettre d'avoir un algorithme modulaire pour tester le vide.

REMARQUE : *Ici, le nombre maximal d'opérations servant dans la décomposition des automates est exponentiel en le nombre d'horloges de l'automate. Il apparaît cependant normal que, plus le nombre d'horloges est important, plus il est complexe de décomposer les automates. En effet, dans [HKW95], il a été montré qu'augmenter le nombre d'horloges accroît strictement l'expressivité du modèle. Cependant, dans [ACM97] et [Asa98], une seule opération de composition est utile. Mais la décomposition proprement dite n'est faite que pour les automates ayant une seule horloge ; l'extension aux automates ayant un nombre quelconque d'horloges est alors obtenue par intersection et par renommage. Ainsi, la complexité due au nombre d'horloges est contenue non pas dans les opérations de composition, mais dans l'intersection (il a été montré dans [ACM97] que l'intersection est nécessaire à leur décomposition). Le dernier point important de cette comparaison est le fait que dans notre étude, nous n'avons paseu besoin du renommage, alors que, dans [ACM97], celui-ci est* *utilisé, même s'il n'est pas prouvé qu'il soit nécessaire (cependant, une conjecture consiste à penser que leur décomposition ne peut se faire sans renommage).*

Chapitre IV

Comment tester le vide?

Dans ce chapitre, nous allons utiliser ce qui a été fait dans ce qui précède pour tester le vide des langages reconnaissables. En effet, peut-être est-il nécessaire de rappeler que ce problème est fondamental lorsqu'on s'intéresse à la vérification. Pour ce faire, il nous faut donner certaines définitions et indiquer certaines notations utilisées ultérieurement.

Soit $\mathcal{A}$ un automate temporisé. Une *zone* pour $\mathcal{A}$ est une union finie de régions. L'ensemble des zones de $\mathcal{A}$ sera noté $\mathcal{Z}_{\mathcal{A}}$.
NOTATION : Soit $\mathcal{A} \in \mathcal{A}\mathrm{ut}(A)$ un automate temporisé ayant n horloges.
Si Z est une zone pour $\mathcal{A}$, on note $(\mathcal{T}_{\mathcal{A}})_Z$ le générateur temporisé qui à $\alpha \in \mathbb{T}^n$ associe $\{u \in \mathcal{T}_{\mathcal{A}}(\alpha) \mid \lambda^{\alpha}_{\mathcal{A}}(u) \cap Z \neq \varnothing\}$.
Si Z est une zone pour $\mathcal{A}$, on note ${}_Z(\mathcal{T}_{\mathcal{A}})$ le générateur temporisé qui à $\alpha \in \mathbb{T}^n$ associe $\mathcal{T}_{\mathcal{A}}(\alpha)$ si $\alpha \in Z$, l'ensemble vide sinon.

L'algorithme présenté dans ce chapitre construit modulairement un automate vérifiant les propriétés suivantes :

1. il accepte le même langage temporisé que l'automate original (disons L)
2. en le considérant comme un automate non temporisé, c'est à dire lorsqu'on lui retire toutes les contraintes sur les horloges, il accepte exactement UNTIME(L)

C'est alors grâce à ce dernier que l'on pourra tester le vide de l'automate initial.

IV.1 Automates temporisés avec invariants

Nous allons débuter en définissant une extension de la classe des automates temporisés, appelée classe des automates temporisés avec invariants. La sémantique des invariants n'est actuellement pas définie de manière universelle, il faudra donc faire attention à celle utilisée ici.

Définition 10 *Soit A un alphabet fini. Un* automate temporisé avec invariants *sur A est un couple $\langle \mathcal{A}, \varphi \rangle$ tel que*

- *$\mathcal{A}$ est un automate temporisé*
- *φ est une application de l'ensemble des états de $\mathcal{A}$ dans $\mathcal{Z}_{\mathcal{A}}$*

*φ est l'*invariant *associé à l'automate $\mathcal{A}$. On notera $\mathcal{A}ut\mathcal{I}(A)$ l'ensemble des automates temporisés avec invariants sur l'alphabet A.*

Ici, les invariants des états sont des conditions d'arrivée dans l'état, pas une condition nécessaire pour rester dans l'état, contrairement à certains modèles d'automates temporisés avec invariants, comme celui défini par Alur dans [Alu98].

EXEMPLE : Un exemple d'automates temporisés avec invariants est présenté à la FIG. 16. Seul

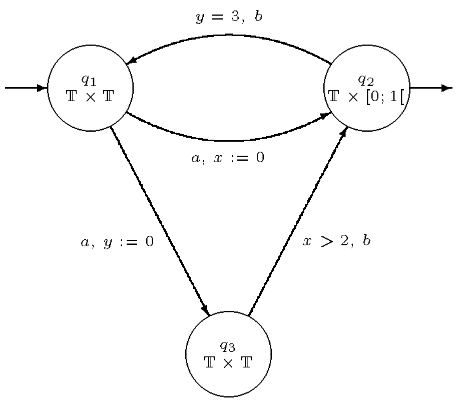


FIG. 16 –

l'invariant de q_2, $\mathbb{T} \times [0;1[$, n'est pas trivial. Il va imposer que, lorsque l'on arrive dans q_3, y doit avoir sa valeur dans $[0;1[$.

Nous allons étendre les définitions d'exécution et de condition d'acceptation à ces automates.

Définition 11 *Soit $\langle \mathcal{A}, \varphi \rangle \in \mathcal{A}ut\mathcal{I}(A)$.*

- *Une* exécution *sur le mot temporisé u dans $\langle \mathcal{A}, \varphi \rangle$ est une exécution $(\langle q_j, v_j \rangle)_{j \in J}$ sur u dans $\mathcal{A}$ vérifiant : $\forall j \in J$, $v_j \in \varphi(q_j)$.*
- *Un mot temporisé u est dit* accepté *par $\langle \mathcal{A}, \varphi \rangle$ en commençant en $\alpha \in \mathbb{T}^{|X|}$ s'il existe une exécution $(\langle q_j, v_j \rangle)_{j \in J}$ sur u telle que $\alpha = v_0$ et si u est un mot fini, alors $q_m \in F$ sinon, $\{k \mid q_k \in R\}$ est un ensemble infini.*

On note $\mathcal{T}_{\langle \mathcal{A},\varphi\rangle}$ la fonction qui à $\alpha \in \mathbb{T}^{|X|}$ associe l'ensemble des mots temporisés acceptés par $\langle \mathcal{A}, \varphi\rangle$ en initialisant les horloges à α.

Comme dans le cas des automates temporisés, nous allons associer à chaque automate temporisé avec invariants une contrainte canonique.

Définition 12 *Soit $\langle \mathcal{A}, \varphi\rangle \in \mathcal{A}ut\mathcal{I}(A)$. Soit pour tout α une fonction* compatible *$\lambda^{\alpha}_{\langle \mathcal{A},\varphi\rangle}$ qui à tout mot temporisé u associe l'ensemble des valuations des horloges après avoir lu u dans $\langle \mathcal{A}, \varphi\rangle$ en initialisant les horloges à α.*

Bien que le modèle ait été étendu en rajoutant des invariants, le pouvoir de cette nouvelle classe d'automates n'est pas plus important que celui de la classe des automates temporisés classiques.

Proposition 22 *Ces automates ont exactement le même pouvoir d'expression que les automates temporisés classiques, i.e. :*

- $\forall \mathcal{A} \in \mathcal{A}ut(A)$, $\exists \langle \mathcal{B}, \varphi\rangle \in \mathcal{A}ut\mathcal{I}(A)$ *tel que* $\mathcal{T}_{\mathcal{A}} = \mathcal{T}_{\langle \mathcal{B},\varphi\rangle}$.
- $\forall \langle \mathcal{B}, \varphi\rangle \in \mathcal{A}ut\mathcal{I}(A)$, $\exists \mathcal{A} \in \mathcal{A}ut(A)$ *tel que* $\mathcal{T}_{\mathcal{A}} = \mathcal{T}_{\langle \mathcal{B},\varphi\rangle}$.

Preuve :

- Soit $\mathcal{A} \in \mathcal{A}\mathrm{ut}(A)$. Soit φ l'application qui à tout état de $\mathcal{A}$ associe $\mathbb{T}^n$. Alors, il est facile d'obtenir que $\mathcal{T}_{\mathcal{A}} = \mathcal{T}_{\langle \mathcal{A},\varphi\rangle}$.
- Soit $\langle \mathcal{B}, \varphi\rangle \in \mathcal{A}\mathrm{ut}\mathcal{I}(A)$. On suppose que $\mathcal{B} = \langle A, Q, I, F, R, T, X\rangle$. Soit I_0 une copie des états initiaux de $\mathcal{B}$. Soit $\mathcal{A} = \langle A, Q', I', F', R', T', X\rangle$ avec, (voir FIG. 17)

$$\left\{\begin{array}{lcl} Q' & = & Q \cup I_0 \\ I' & = & I_0 \\ F' & = & F \\ R' & = & R \\ T' & = & \{\langle i_0, \varepsilon, i, \varphi(i), \varnothing\rangle \mid i \text{ état initial de } \mathcal{B}\} \\ & & \cup\{\langle q, a, q', g \cup \varphi(q'), r\rangle \mid \langle q, a, q', g, r\rangle \in T\} \end{array}\right.$$

Les ε-transitions qui sont rajoutées peuvent être enlevées par l'algorithme présenté dans [DGP97]. En outre, il est facile de voir que $\mathcal{T}_{\mathcal{A}} = \mathcal{T}_{\langle \mathcal{B},\varphi\rangle}$.

□

Il serait normal de se poser la question suivante : pourquoi avoir défini cette nouvelle classe d'automates temporisés, alors qu'elle n'exprime pas plus de choses que la classe originale?

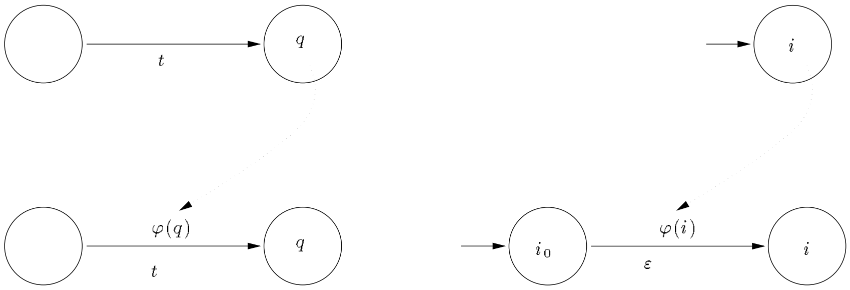


FIG. 17 – *Elimination des invariants*

IV.2 « Détemporisation » des automates temporisés avec invariants

Les automates temporisés avec invariants définis précédemment, qui n'ont pas un pouvoir d'expression plus grand que les automates temporisés classiques, vont être utiles pour enlever les contraintes de temps sur les transitions tout en préservant les langages temporisés acceptés.

Soit $\Pi : \mathcal{A}\mathrm{ut}\mathcal{I}(A) \longrightarrow \mathcal{A}\mathrm{ut}\mathcal{I}(A)$ qui à $\langle \mathcal{A}, \varphi \rangle \in \mathcal{A}\mathrm{ut}\mathcal{I}(A)$ associe l'automate $\langle \mathcal{A}', \varphi \rangle$ tel que : si $q \xrightarrow{g,a,r} q'$ est une transition de $\mathcal{A}'$, alors $q \xrightarrow{a} q'$ sera une transition de $\mathcal{A}'$.
Dans la suite, si $\langle \mathcal{A}, \varphi \rangle \in \mathcal{A}\mathrm{ut}\mathcal{I}(A)$, nous noterons $\langle \mathcal{A}', \varphi \rangle = \Pi(\langle \mathcal{A}, \varphi \rangle)$.

Pour $\langle \mathcal{A}, \varphi \rangle \in \mathcal{A}\mathrm{ut}\mathcal{I}(A)$, $\langle \mathcal{A}', \varphi \rangle$ peut être considéré comme un automate fini (en oubliant la fonction φ et les contraintes sur les horloges de $\mathcal{A}$). Nous notons $\mathcal{L}(\langle \mathcal{A}', \varphi \rangle)$ le langage non temporisé accepté par $\langle \mathcal{A}', \varphi \rangle$. Pour $\alpha \in \mathbb{T}^n$, nous noterons $\langle \mathcal{A}', \varphi \rangle^\alpha$ l'automate tel que l'on restreint les états initiaux à ceux dont l'invariant contient α.
Nous avons tout de suite les propriétés suivantes :

Proposition 23 *Soit $\langle \mathcal{A}, \varphi \rangle \in \mathcal{A}ut\mathcal{I}(A)$. Alors, pour tout $\alpha \in \mathbb{T}^n$:*

$$\begin{cases} \mathcal{T}(\langle \mathcal{A}, \varphi \rangle^\alpha) \subseteq \mathcal{T}(\langle \mathcal{A}', \varphi \rangle^\alpha) \\ \\ \lambda^\alpha_{\langle \mathcal{A}, \varphi \rangle} \subseteq \lambda^\alpha_{\langle \mathcal{A}', \varphi \rangle} \\ \\ \textsc{Untime}\,(\mathcal{T}\,(\langle \mathcal{A}', \varphi \rangle^\alpha)) \subseteq \mathcal{L}(\langle \mathcal{A}', \varphi \rangle^\alpha) \end{cases}$$

Les automates auxquels nous allons surtout nous intéresser sont ceux pour lesquels les propriétés réciproques sont vérifiées.

Définition 13 *Soit $\langle \mathcal{A}, \varphi \rangle \in \mathcal{A}ut\mathcal{I}(A)$.*

- *$\mathcal{A}$ est dit* quasi-fiable *si pour tout $\alpha \in \mathbb{T}^n$,*

$$\begin{cases} \mathcal{T}(\langle \mathcal{A}', \varphi \rangle^\alpha) \subseteq \mathcal{T}(\langle \mathcal{A}, \varphi \rangle^\alpha) \\ \lambda^\alpha_{\langle \mathcal{A}', \varphi \rangle} \subseteq \lambda^\alpha_{\langle \mathcal{A}, \varphi \rangle} \end{cases}$$

- $\mathcal{A}$ *est dit* fiable *si* $\mathcal{A}$ *est quasi-fiable et si pour tout* $\alpha \in \mathbb{T}^n$,

$$\mathcal{L}(\langle \mathcal{A}', \varphi \rangle^\alpha) \subseteq \textsc{Untime}(\mathcal{T}(\langle \mathcal{A}', \varphi \rangle^\alpha)$$

IV.3 Propriétés des automates temporisés avec invariants quasi-fiables et fiables

Dans cette partie, nous allons détailler un algorithme permettant de construire, à partir d'un automate donné sous forme modulaire, un automate temporisé avec invariants qui reconnaîtra le même langage que l'automate original et qui, en plus, sera fiable. Pour tester la vacuité du langage initial, il sera donc suffisant de tester le vide du langage non temporisé accepté par l'automate fiable.

Proposition 24 *Soit* $\mathcal{A} \in \mathcal{A}ut(A)$. *Il existe* $\langle \mathcal{B}, \varphi \rangle \in \mathcal{A}ut\mathcal{I}(A)$ *tel que* $\mathcal{T}_\mathcal{A} = \mathcal{T}_{\langle \mathcal{B}, \varphi \rangle}$ *et* $\mathcal{B}$ *fiable.*

Preuve : Il suffit pour cela de faire le produit cartésien de l'automate temporisé $\mathcal{A}$ par l'automate des successeurs des régions défini dans la partie I.2.2.

L'automate qui résulte de ce produit cartésien est un automate des régions que l'on peut qualifier d'étendu. □

Nous allons essayer de construire ici un automate qui sera en pratique plus petit que celui des régions (certaines régions étant regroupées en zones).

Proposition 25 *Soit* $\langle \mathcal{A}, \varphi \rangle \in \mathcal{A}ut\mathcal{I}(A)$ *quasi-fiable. Il existe un algorithme effectif qui transforme* $\langle \mathcal{A}, \varphi \rangle$ *en un automate* $\langle \mathcal{B}, \psi \rangle \in \mathcal{A}ut\mathcal{I}(A)$ *fiable tel que* $\mathcal{T}_{\langle \mathcal{A}, \varphi \rangle} = \mathcal{T}_{\langle \mathcal{B}, \psi \rangle}$.

Preuve : Soit $\langle \mathcal{A}, \varphi \rangle \in \mathcal{A}\mathrm{ut}\mathcal{I}(A)$. Soit $e = \langle q, a, q', r, g \rangle$ une transition de $\langle \mathcal{A}, \varphi \rangle$. Soit $Z = \varphi(q)$ et $Z' = \varphi(q')$. Soit Γ_e la transformation suivante décrite à la FIG. 18 :

- Si $a \neq \overline{\varepsilon}$, nous remplaçons l'état q par deux états qui jouent le même rôle que lui sauf pour la transition e : $(q, Z \cap \mathsf{Pred}_r(Z'))$ et $(q, Z \backslash \mathsf{Pred}_r(Z'))$. Les invariants de ces nouveaux états sont : $\psi(q, Z \cap \mathsf{Pred}_r(Z')) = Z \cap \mathsf{Pred}_r(Z')$ et $\psi(q, Z \backslash \mathsf{Pred}_r(Z')) = Z \backslash \mathsf{Pred}_r(Z')$. La transition $\langle (q, Z \cap \mathsf{Pred}_r(Z')), a, q', r, g \rangle$ est conservée alors que l'on ne rajoute pas la transition $\langle (q, Z \backslash \mathsf{Pred}_r(Z')), a, q', r, g \rangle$. Si un des états voit son invariant devenir vide, l'état n'est pas conservé.

- Si $a = \overline{\varepsilon}$, nous remplaçons l'état q par deux états qui jouent le même rôle que lui sauf pour e : $(q, Z \cap Z'_{(r)})$ et $(q, Z \backslash Z'_{(r)})$. Les invariants de ces nouveaux états deviennent : $\psi(q, Z \cap Z'_{(r)}) = Z \cap Z'_{(r)}$ et $\psi(q, Z \backslash Z'_{(r)}) = Z \backslash Z'_{(r)}$. La transition $\langle (q, Z \cap Z'_{(r)}), a, q', r, g \rangle$ est conservée alors que l'on ne rajoute pas la transition $\langle (q, Z \backslash Z'_{(r)}), a, q', r, g \rangle$. Si un des états voit son invariant devenir vide, l'état n'est pas conservé.

où $\mathsf{Pred}_r(Z')) = \mathsf{Pred}(Z')_{|X \backslash r} \times \mathbb{T}^{|r|}$ et $Z'_{(r)} = Z'_{|X \backslash r} \times \mathbb{T}^{|r|}$.

REMARQUE : *Z' étant une zone, $\mathsf{Pred}_r(Z'))$ et $Z'_{(r)}$ sont des zones. L'intersection et le complémentaire d'une zone étant toujours soit vide, soit une zone, les invariants de tous les états restent toujours des zones.*

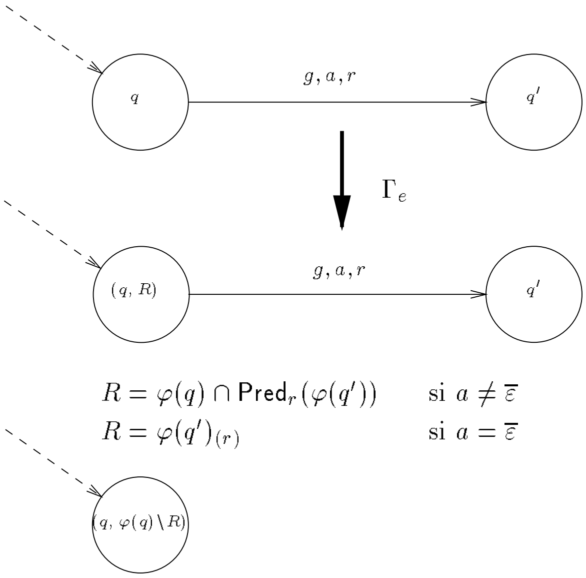


FIG. 18 – *Transformation* Γ_e

Pour toute valuation v dans R, il existe une valuation v' dans Z' telle qu'il existe $t \in \mathbb{T}$ avec $v+t \models g$ et $v' = {}_{[r\leftarrow 0]}(v+t)$ (si $a = \overline{\varepsilon}$, nous avons en plus $t = 0$). Nous dirons que v' vérifie la propriété $(*)_{(v,t)}$. Tout ceci signifie que quelle que soit la valuation des horloges lorsque l'on arrive dans (q, Z), on ne reste pas bloqué dans cet état. Ceci revient juste à enlever des états les valuations pour lesquelles il n'est pas possible de prendre la transition e ; mais ceci ne retire pas de possibilités quant au langage accepté.
Il est alors facile de prouver que pour tout $\alpha \in \mathbb{T}^n$

$$\left\{\begin{array}{l} \mathcal{T}_{\Gamma_e(\langle\mathcal{A},\varphi\rangle)}(\alpha) = \mathcal{T}_{\langle\mathcal{A},\varphi\rangle}(\alpha) = \mathcal{T}_{\Pi(\Gamma_e(\langle\mathcal{A},\varphi\rangle))}(\alpha) \\ \lambda^{\alpha}_{\Gamma_e(\langle\mathcal{A},\varphi\rangle)} = \lambda^{\alpha}_{\langle\mathcal{A},\varphi\rangle} = \lambda^{\alpha}_{\Pi(\Gamma_e(\langle\mathcal{A},\varphi\rangle))} \end{array}\right.$$

car Γ_e n'enlève pas à un mot temporisé de façon d'être accepté.

Nous réalisons cette transformation pour toutes les transitions qui existaient déjà dans $\langle\mathcal{A}, \varphi\rangle$ et pour toutes celles qui se sont formées jusqu'à ce que cela s'arrête.
REMARQUE : *La suite de transformations s'arrête forcément car, comme nous l'avons dit précédemment, les invariants sont toujours des zones, donc ils ne deviennent jamais plus petits qu'une région. Au bout d'un certain temps, il va y avoir stabilisation des invariants et donc l'algorithme de transformation va s'arrêter.*

Soit $\Gamma(\langle\mathcal{A}, \varphi\rangle)$ le résultat de cette succession de transformations. Alors $\Gamma(\langle\mathcal{A}, \varphi\rangle)$ est fiable et pour tout $\alpha \in \mathbb{T}^n$, $\mathcal{T}(\langle\mathcal{A}, \varphi\rangle^{\alpha}) = \mathcal{T}(\Gamma(\langle\mathcal{A}, \varphi\rangle)^{\alpha})$.
En effet, $\Gamma(\langle\mathcal{A}, \varphi\rangle)$ vérifie déjà la propriété de quasi-fiabilité d'après ce qui précède. Il reste à montrer que pour tout α

$$\mathcal{L}(\Pi(\Gamma(\langle\mathcal{A}, \varphi\rangle))^{\alpha}) \subseteq \textsc{Untime}\,(\mathcal{T}\,(\Pi\,(\Gamma\,(\langle\mathcal{A}, \varphi\rangle))^{\alpha}))$$

Pour prouver ceci, donnons-nous $u = (\sigma, \tau) \in \mathcal{L}(\Pi(\Gamma(\langle \mathcal{A}, \varphi \rangle))^\alpha)$ et $(q_i)_{i \in J}$ un chemin acceptant u. D'après la définition de $\Gamma(\langle \mathcal{A}, \varphi \rangle)$, $\alpha \in \varphi(q_0)$. Soient t_1 et $v_1 \in \varphi(q_1)$ vérifiant la propriété $(*)_{(\alpha, t_1)}$. Nous construisons ainsi inductivement des dates t_i et des valuations v_i tels que, si $\tau_i = \sum_{j=1}^{i} t_j$,

$$\langle q_0, \alpha \rangle \xrightarrow[\tau_1]{\sigma_1} \langle q_1, v_1 \rangle \cdots$$

soit un chemin acceptant pour u dans $\Pi(\Gamma(\langle \mathcal{A}, \varphi \rangle))$.
Ainsi $u \in \mathcal{T}(\Pi(\Gamma(\langle \mathcal{A}, \varphi \rangle))^\alpha)$ et donc $\sigma \in \textsc{Untime}(\mathcal{T}(\Pi(\Gamma(\langle \mathcal{A}, \varphi \rangle))^\alpha))$.

En conclusion, $\Gamma(\langle \mathcal{A}, \varphi \rangle)$ est fiable, et c'est ce que nous voulions montrer. □

Proposition 26 *Soit $\langle \mathcal{A}, \varphi \rangle \in \mathcal{A}ut\mathcal{I}(A)$ fiable. Soit Z une zone. Il existe $\langle \mathcal{B}, \psi \rangle \in \mathcal{A}ut\mathcal{I}(A)$ fiable tel que $(\mathcal{T}(\langle \mathcal{A}, \varphi \rangle)_Z = \mathcal{T}(\langle \mathcal{B}, \psi \rangle)$.*

Preuve : Soit $\langle \mathcal{A}, \varphi \rangle \in \mathcal{A}\mathrm{ut}\mathcal{I}(A)$ et Z une zone. Soit κ_Z la transformation suivante :

- Pour tout état final f de $\langle \mathcal{A}, \varphi \rangle$, nous le remplaçons par deux états qui jouent le même rôle que lui $(Z = \varphi(f))$: $(f, R \cap Z)$ et $(f, R \backslash Z)$. Les invariants de ces nouveaux états deviennent $\psi(f, R \cap Z) = R \cap Z$ et $\psi(f, R \backslash Z) = R \backslash Z$.
- Seul $(f, R \cap Z)$ reste final.

Alors, il est facile de voir que $\kappa_Z(\langle \mathcal{A}, \varphi \rangle)$ est quasi-fiable (car $\langle \mathcal{A}, \varphi \rangle$ l'était) et que

$$\mathcal{T}(\kappa_Z(\langle \mathcal{A}, \varphi \rangle)) = (\mathcal{T}(\langle \mathcal{A}, \varphi \rangle))_Z$$

Alors, d'après la proposition 25, nous obtenons le résultat annoncé. □

Proposition 27 *Soit $\langle \mathcal{A}, \varphi \rangle \in \mathcal{A}ut\mathcal{I}(A)$ fiable. Soit Z une zone. Il existe $\langle \mathcal{B}, \psi \rangle \in \mathcal{A}ut\mathcal{I}(A)$ fiable tel que $_Z(\mathcal{T}(\langle \mathcal{A}, \varphi \rangle)) = \mathcal{T}(\langle \mathcal{B}, \psi \rangle)$.*

Preuve : Soit $\langle \mathcal{A}, \varphi \rangle \in \mathcal{A}\mathrm{ut}\mathcal{I}(A)$ et Z une zone. Soit κ_Z la transformation suivante :

- Pour tout état initial i de $\mathcal{A}$, nous le remplaçons par deux états qui jouent le même rôle que lui $(Z = \varphi(i))$: $(i, R \cap Z)$ et $(i, R \backslash Z)$. Les invariants de ces nouveaux états deviennent $\psi(i, R \cap Z) = R \cap Z$ et $\psi(i, R \backslash Z) = R \backslash Z$.
- Seul $(i, R \cap Z)$ reste initial.

Alors, il est facile de voir que $\kappa_Z(\langle \mathcal{A}, \varphi \rangle)$ est quasi-fiable (car $\langle \mathcal{A}, \varphi \rangle$ l'était) et que

$$\mathcal{T}(\kappa_Z(\langle \mathcal{A}, \varphi \rangle)) = (\mathcal{T}(\langle \mathcal{A}, \varphi \rangle))_Z$$

Alors, d'après la proposition 25, nous obtenons le résultat annoncé. □

IV.4 Test du vide

Nous arrivons maintenant à ce pour quoi tout ce qui vient d'être défini l'a été : tester le vide de manière modulaire.

Tout repose sur la propriété suivante :

Proposition 28 *Soit $\mathcal{A} \in \mathcal{M}od(A)$. Le calcul de $\langle \mathcal{B}, \psi \rangle \in \mathcal{A}ut\mathcal{I}(A)$ fiable tel que pour tout $\alpha \in \mathbb{T}^n$, $\mathcal{T}_\mathcal{A}(\alpha) = \mathcal{T}_{\langle \mathcal{B}, \psi \rangle}(\alpha)$ peut se faire de manière modulaire.*

Preuve : Cette preuve va être faite par induction sur la forme de $\mathcal{A}$.

- BASE. Un automate fiable pour $\mathcal{A}_{\langle \delta a \rangle_I}$ est celui représenté à la FIG. 19 où I^- représente les prédécesseurs de I, i.e. $\{(a_1 \cdots a_n) \mid \exists t \in \mathbb{T}\, a_i + t \in I_i\}$.

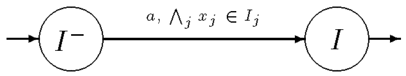


FIG. 19 – *Automate fiable pour $\mathcal{A}_{\langle \delta a \rangle_I}$*

  Un automate fiable pour $\mathcal{A}_\varepsilon$ est l'automate à un état à la fois final et initial dont l'étiquette est $\mathbb{T}^n$.

- OPÉRATEUR $+$. C'est l'union classique des deux automates fiables correspondants aux automates dont nous voulons faire l'union.

- OPÉRATEUR $\dot{_C}$. Supposons que pour $\mathcal{A}$ et pour $\mathcal{B}$, des automates fiables $\langle \mathcal{A}_0, \varphi \rangle$ et $\langle \mathcal{B}_0, \psi \rangle$ sont construits.

  Soit $Z = (\bigcup\{\psi(i) \mid i \text{ état initial de } \langle \mathcal{B}_0, \psi \rangle\})_{|X \setminus C} \times \mathbb{T}^{|C|}$. D'après la proposition 26, il existe $\langle \mathcal{C}, \omega \rangle \in \mathcal{A}ut\mathcal{I}(A)$ fiable tel que $\mathcal{T}(\langle \mathcal{C}, \rangle) = (\mathcal{T}(\langle \mathcal{A}_0, \varphi \rangle))_Z$.
  Soit $Z' = (\bigcup\{\psi(i) \mid i \text{ état initial de } \langle \mathcal{B}_0, \psi \rangle\})_{|X \setminus C} \times [C \leftarrow 0]$. D'après la proposition 27, il existe $\langle \mathcal{D}, \rho \rangle \in \mathcal{A}ut\mathcal{I}(A)$ fiable tel que $\mathcal{T}(\langle \mathcal{D}, \rho \rangle) = {}_Z(\mathcal{T}(\langle \mathcal{A}_0, \varphi \rangle))$.
  Nous plaçns une $\overline{\varepsilon}$-transition entre tous les états finals f de $\langle \mathcal{C}, \omega \rangle$ et tous les états initiaux i de $\langle \mathcal{D}, \rho \rangle$ tels que $(\omega(f) \cap \rho(i))_{|X \setminus C} \neq \varnothing$. Soit $\langle \mathcal{E}, . \rangle$ l'automate résultant. Soit $u \in \mathcal{T}_{\langle \mathcal{E}, . \rangle}(\alpha)$. Ecrivons $u = u_1.u_2$ avec $u_1 \in \mathcal{T}_{\langle \mathcal{C}, \omega \rangle}(\alpha)$ et $u_2 \backslash_{u_1} \in \mathcal{T}_{\langle \mathcal{D}, \rho \rangle}([C \leftarrow 0]\beta)$ avec $\beta \in \lambda^{\alpha}_{\langle \mathcal{C}, \omega \rangle}(u_1)$. Alors, par hypothèse sur $\langle \mathcal{C}, \omega \rangle$ et $\langle \mathcal{D}, \rho \rangle$ (qui sont fiables), $u_1 \in \mathcal{T}_{\langle \mathcal{C}', \omega \rangle}(\alpha)$ et $u_2 \backslash_{u_1} \in \mathcal{T}_{\langle \mathcal{D}', \rho \rangle}([C \leftarrow 0]\beta)$, donc $u \in \mathcal{T}_{\langle \mathcal{E}', . \rangle}(\alpha)$.
  De même, $\lambda^{\alpha}_{\langle \mathcal{E}', . \rangle} \subseteq \lambda^{\alpha}_{\langle \mathcal{E}, . \rangle}$. Donc $\langle \mathcal{E}, . \rangle$ est quasi-fiable.

  En appliquant la proposition 25, nous savons construire un automate $\mathcal{F}$ fiable vérifiant $\mathcal{T}_\mathcal{F} = \mathcal{T}_{\mathcal{A} \dot{_C} \mathcal{B}}$.

- OPÉRATEUR $\overset{*}{C}$. Supposons que pour $\mathcal{A}$ un automate fiable $\langle \mathcal{A}_0, \varphi_0 \rangle$ soit construit.

  Soit $Z_i = (\bigcup\{\varphi_{i-1}(i) \mid \exists i$ état initial de $\langle \mathcal{A}_{i-1}, \varphi_{i-1}\rangle\})_{|X\setminus C} \times \mathbb{T}^{|C|}$.
  Soit $Z'_i = (\bigcup\{\varphi_{i-1}(i) \mid \exists i$ état initial de $\langle \mathcal{A}_{i-1}, \varphi_{i-1}\rangle\})_{|X\setminus C} \times [C\leftarrow 0]$.
  Soit $\langle \mathcal{A}_i, \varphi_i \rangle$ automate fiable tel que $\mathcal{T}_{\langle \mathcal{A}_i, \varphi_i\rangle} = {}_{Z'_i}(\mathcal{T}_{\langle \mathcal{A}_{i-1}, \varphi_{i-1}\rangle})_{Z_i}$. La suite des automates $(\langle \mathcal{A}_i, \varphi_i \rangle)$ est stationnaire. Soit $\langle \mathcal{B}, \psi \rangle$ la «limite» de cette suite. Si i était initial dans $\langle \mathcal{A}_0, \varphi_0 \rangle$, alors pour tout état i' « provenant » de i, i' est initial pour $\langle \mathcal{B}, \psi \rangle$. Si f était final dans $\langle \mathcal{A}_0, \varphi_0 \rangle$, pour tout état f' provenant de f, f' sera final pour $\langle \mathcal{B}, \psi \rangle$. Nous plaçons une $\overline{\varepsilon}$-transition entre tous les états finals f de $\langle \mathcal{B}, \psi \rangle$ et tous les états initiaux i de $\langle \mathcal{B}, \psi \rangle$ tels que $(\psi(f) \cap \psi(i))_{|X\setminus C} \neq \varnothing$. Soit $\langle \mathcal{C}, \omega \rangle$ l'automate résultant. Alors, $\langle \mathcal{C}, \omega \rangle$ est quasi-fiable et vérifie $\mathcal{T}_{\langle \mathcal{C}, \omega\rangle} = \mathcal{T}_{\mathcal{A}^{\overset{*}{C}}}$.

- OPÉRATEUR $\overset{\omega}{C}$. La construction est similaire à celle pour l'opérateur $\overset{*}{C}$.

  Et nous obtenons donc ce que nous voulions montrer.

□

EXEMPLE : Un exemple pour cet algorithme se trouve à l'annexe B.

Corollaire 29 *Pour tout automate temporisé $\mathcal{A}$, il existe $\langle \mathcal{B}, \varphi \rangle \in \mathcal{A}ut\mathcal{I}(A)$ fiable tel que $\mathcal{T}_{\mathcal{A}} = \mathcal{T}_{\langle \mathcal{B}, \varphi\rangle}$.*

Preuve : C'est une conséquence immédiate du théorème de décomposition modulaire des automates temporisés (théorème 21) et de la proposition précédente. □

Corollaire 30 *Pour tout automate temporisé $\mathcal{A}$, il est possible de construire un automate temporisé $\mathcal{F} \in \mathcal{A}ut\mathcal{I}(A)$ tel que pour tout α, $\mathcal{T}_{\mathcal{A}}(\alpha) = \varnothing$ si et seulement $\mathcal{L}(\mathcal{F}'^{\alpha}) = \varnothing$.*

Nous allons faire une remarque sur la complexité de cet algorithme. L'automate fiable associé à un automate donné est en fait un «automate des zones». Dans le pire des cas, tous les invariants sont des régions et nous retrouvons alors l'automate des régions « étendu » vu à la propriété 24. Donc, la complexité la pire est PSPACE. Cependant, il est souvent vrai qu'il n'y a pas besoin de considérer les régions une par une : plusieurs régions peuvent être « équivalentes » ; c'est alors qu'elles sont regroupées dans un même invariant. La complexité de cet automate est en pratique meilleure dans de nombreux cas, car il arrive souvent que l'on puisse regrouper des régions en des zones.

Conclusion

L'objet de ce travail était de définir de «bonnes» opérations de composition entre les automates : «bonnes» signifie ici que si l'on se donne un système de taille importante, certaines propriétés que l'on cherche à étudier sont obtenues en s'intéressant aux propriétés des sous-modules qui le composent. Ceci s'inscrit dans le cadre de la modularité, idée fondamentale en informatique de la vérification car les systèmes réels que l'on cherche à étudier sont souvent de taille importante et donnés sous forme de plusieurs petits systèmes.

Nous avons commencé par étudier le travail d'Asarin, Caspi et Maler ([ACM97] puis [Asa98]) qui consiste à définir la notion d'expression rationnelle temporisée. Une opération de composition séquentielle avait alors été définie pour la classe des automates temporisés ayant une seule horloge, l'extension aux autres automates temporisés étant réalisée par intersection et par renommage. Cependant, le renommage n'étant pas toujours aisé à calculer, il nous a semblé judicieux de tenter d'arriver au même but sans utiliser cette opération. Nous avons alors essayé d'étendre directement l'opération de composition précédemment citée aux automates temporisés ayant un nombre quelconque d'horloges.

Nous avons alors défini une famille d'opérations de composition séquentielle sur la classe des automates temporisés. Nous avons ensuite décrit en termes de comportements ce que ces opérations signifient. Ensuite, ayant vérifié certaines propriétés de ces opérations (comme l'associativité) nous avons pu présenter un algorithme de décomposition modulaire de tout automate temporisé en automates «élémentaires» en utilisant les opérations de composition précédemment définies. En outre, nous avons obtenu un algorithme modulaire pour tester le vide d'un système composé.

Les perspectives de ce travail sont de natures diverses :

1. il serait intéressant d'étendre les résultats théoriques à des classes d'automates plus grandes telles que la classe d'automates temporisés avec ε-transitions ou celle des automates temporisés avec invariants (du type de ceux définis par Alur dans [Alu98]),

2. les algorithmes présentés ici pourraient être testés sur des cas réels,

3. ils pourraient aussi être implantés dans un outil de model-checking existant,

4. d'autres techniques de model-checking pourraient utiliser cette décomposition modulaire des automates temporisés, telles que le model-checking compositionnel.

Annexe A
Exemple de résolution d'un système

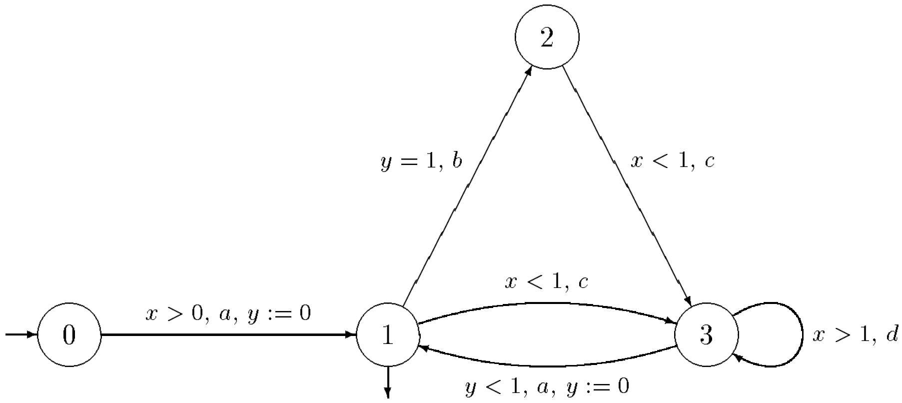


FIG. 20 – *Automate $\mathcal{A}$*

L'automate $\mathcal{A}$ de la FIG. 20 définit le système d'équations linéaires temporisées suivant :

$$\begin{cases} X_0 &= \langle \delta a \rangle_{\mathbb{Q}_*^+ \times \mathbb{Q}^+} \,\dot{}_{\{y\}}\, X_1 \\ X_1 &= \varepsilon + \langle \delta b \rangle_{\mathbb{Q}^+ \times \{1\}} \,\dot{}_{\varnothing}\, X_2 + \langle \delta c \rangle_{[0;1[\times \mathbb{Q}^+} \,\dot{}_{\varnothing}\, X_3 \\ X_2 &= \langle \delta c \rangle_{[0;1[\times \mathbb{Q}^+} \,\dot{}_{\varnothing}\, X_3 \\ X_3 &= \langle \delta d \rangle_{]1;+\infty[\times \mathbb{Q}^+} \,\dot{}_{\varnothing}\, X_3 + \langle \delta a \rangle_{\mathbb{Q}^+ \times [0;1[} \,\dot{}_{\{y\}}\, X_1 \end{cases}$$

En résolvant ce système d'équations, on peut obtenir comme solution :

$$X_0 = \langle \delta a \rangle_{\mathbb{Q}_*^+ \times \mathbb{Q}^+} \,\dot{}_{\{y\}} \left(\varepsilon + \left(\langle \delta b \rangle_{\mathbb{Q}^+ \times \{1\}} \,\dot{}_{\varnothing}\, \langle \delta c \rangle_{[0;1[\times \mathbb{Q}^+} \,\dot{}_{\varnothing}\, \alpha + \langle \delta c \rangle_{[0;1[\times \mathbb{Q}^+} \,\dot{}_{\varnothing}\, \alpha \right)^{\overset{*}{\{y\}}} \right)$$

où

$$\alpha = \left(\varepsilon + \left(\langle \delta d \rangle_{]1;+\infty[\times \mathbb{Q}^+} \right)^{\overset{*}{\varnothing}} \right) \,\dot{}_{\varnothing}\, \langle \delta a \rangle_{\mathbb{Q}^+ \times [0;1[}$$

Ceci définit donc une décomposition modulaire de $\mathcal{A}$ en automates temporiés simples comme $\mathcal{A}_{\langle \delta a \rangle_{\mathbb{Q}_*^+ \times \mathbb{Q}^+}}$, $\mathcal{A}_{\langle \delta c \rangle_{[0;1[\times \mathbb{Q}^+}}$, ...

Annexe B

Exemple de test du vide via l'algorithme présenté

Nous allons traiter un exemple pour voir comment l'algorithme fonctionne exactement : nous voudrions tester la vacuité de

$$L = \left(\langle\delta a\rangle_{[0;1]\times\mathbb{Q}^+} + \langle\delta b\rangle_{\mathbb{Q}^+\times\mathbb{Q}^+}\right) \dot{\varnothing} \langle\delta c\rangle_{\mathbb{Q}^+\times[2;3]}$$

La première étape consiste à construire les automates fiables correspondant aux automates de base intervenant dans l'expression ci-avant. Elle est représentée à la FIG. 21.
La seconde étape consiste à relier les différents automates avec les $\overline{\varepsilon}$-transitions en faisant attention aux invariants. Cette étape est représentée à la FIG. 22.
La dernière étape consiste à raffiner toutes les transitions, c'est à dire à appliquer Γ défini dans l'algorithme, à l'automate obtenu. Le résultat est représenté à la FIG. 23.
Le dernier automate obtenu donne toutes les informations nécessaires sur le langage accepté par l'automate initial. Par exemple, on a tout de suite que le langage que l'on considérait, L, n'est pas vide car le langage reconnu par le dernier automate n'est pas vide. En outre, on obtient aussi que $\textsc{Untime}(L) = ac + bc$. Les autres résultats que l'on peut obtenir sont du type :

$$[0;1]\times[0;3] \xrightarrow{b}]1;+\infty[\times[0;3] \xrightarrow{\overline{\varepsilon}}]1;+\infty[\times[0;3] \xrightarrow{c} \mathbb{Q}^+\times[2;3]$$

étant une exécution acceptante de l'automate fini, cela signifie qu'il existe une exécution acceptante dans l'automate initial pour le mot $(b;2)(c;3)$ et que celui-ci est donc accepté par l'automate initial lorsqu'on initialise ses horloges à zéro. On peut effectivement voir que c'était le cas.

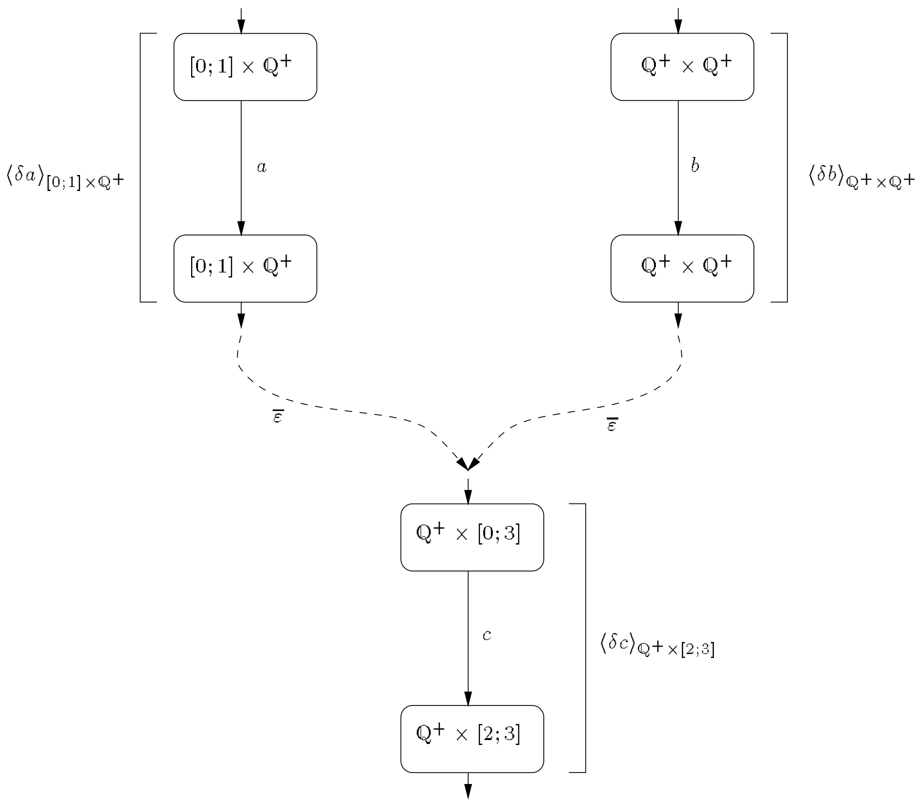


Fig. 21 – *Automates de base*

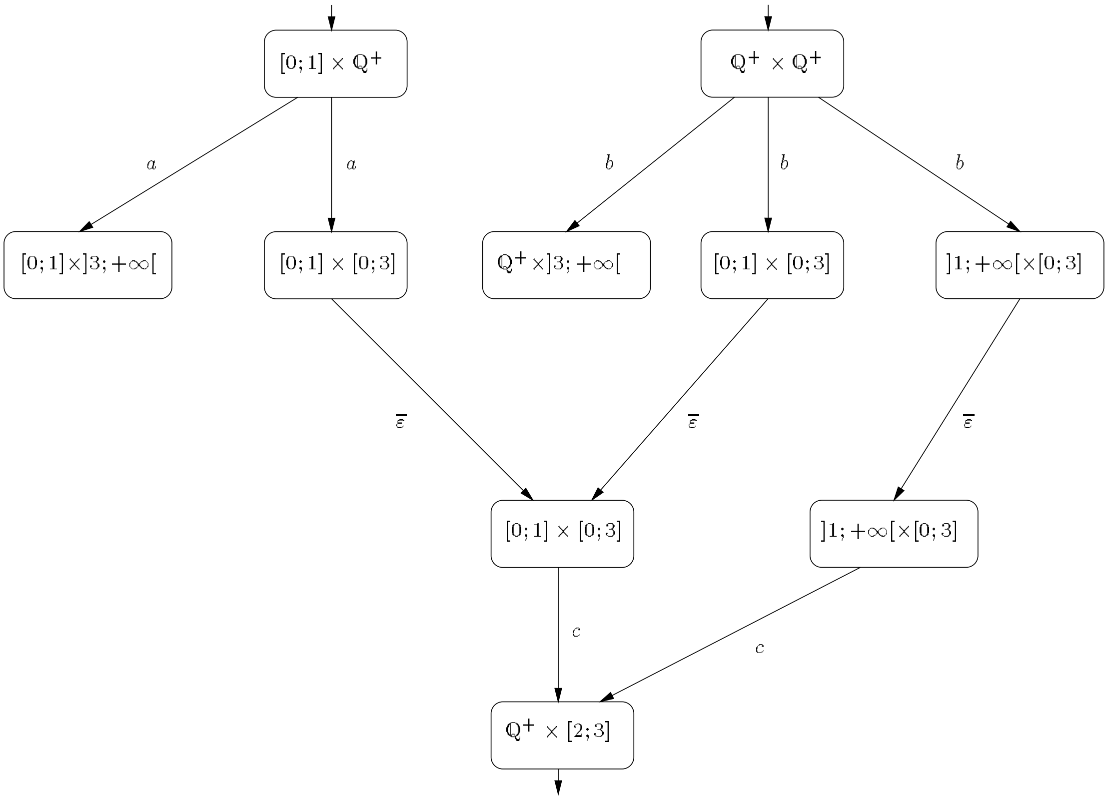


FIG. 22 – *Après rajout des $\overline{\varepsilon}$-transitions*

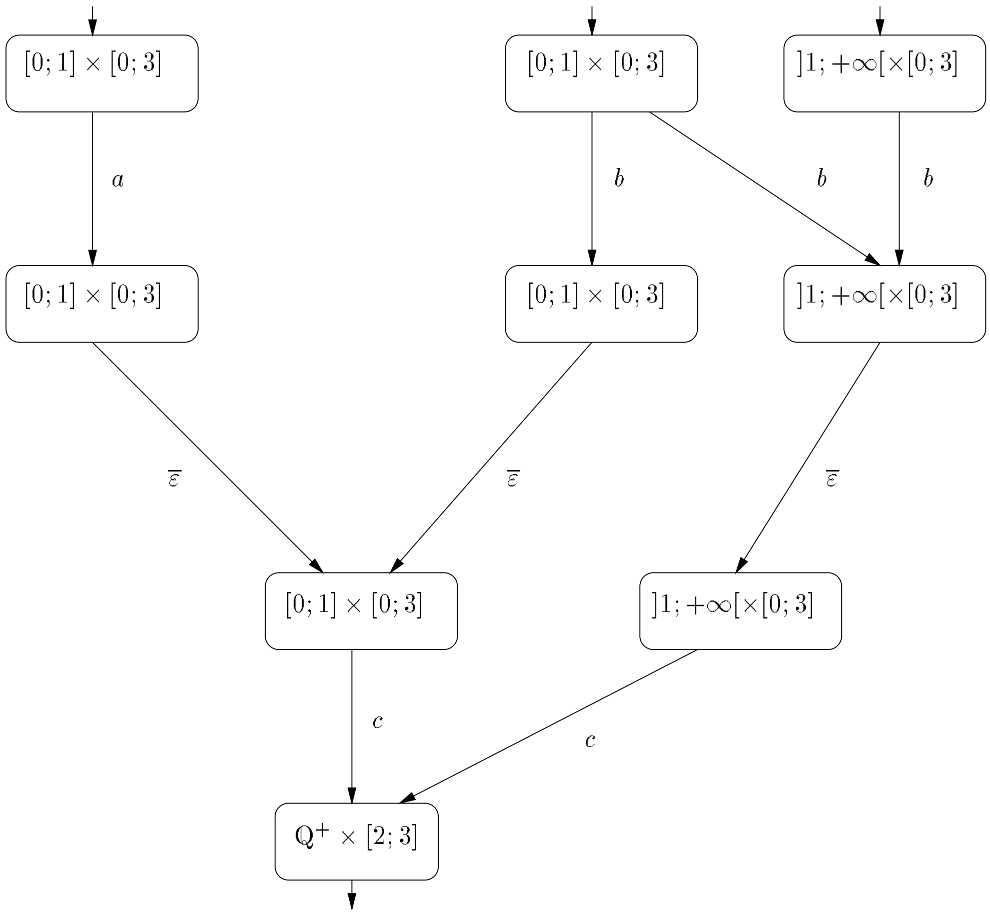


FIG. 23 – *Après raffinement de toutes les transitions*

Bibliographie

[ABL97] Luca Aceto, Augusto Burgueño, et Kim G. Larsen. *Model checking via reachability testing for timed automata. Rapport de recherche RS–97–29*, BRICS, Aalborg University, 1997. A paraître dans ETAPS'98.

[ACD93] Rajeev Alur, Costas Courcoubetis, et David L. Dill. *Model-checking in dense real-time. Information and Computation*, volume 104 pp. 2–34, 1993.

[ACH+92] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, David L. Dill, et Howard Wong-Toi. *Minimization of timed transition systems*. Dans *Proceedings of CONCUR'92*, volume 630 de *Lecture Notes in Computer Science*, pp. 340–354 (Springer-Verlag, 1992).

[ACH94] Rajeev Alur, Costas Courcoubetis, et Thomas A. Henzinger. *The observational power of clocks*. Dans *Proceedings of CONCUR'94*, Lecture Notes in Computer Science, pp. 162–177 (Springer-Verlag, 1994).

[ACM97] Eugene Asarin, Paul Caspi, et Oded Maler. *A Kleene theorem for timed automata. Proceedings of LICS'97*, pp. 160–171, 1997.

[AD90] Rajeev Alur et David L. Dill. *Automata for modeling real-time systems*. Dans *Proceedings of ICALP'90*, volume 443 de *Lecture Notes in Computer Science*, pp. 322–335 (Springer-Verlag, 1990).

[AD94] Rajeev Alur et David L. Dill. *A theory of timed automata. Theoretical Computer Science*, volume 126 pp. 183–235, 1994.

[AFH94] Rajeev Alur, Limor Fix, et Thomas A. Henzinger. *A determinizable class of timed automata*. Dans *Proceedings of CAV'94*, Lecture Notes in Computer Science, pp. 1–13 (Springer-Verlag, 1994).

[AH93] Rajeev Alur et Thomas A. Henzinger. *Real-time logics: Complexity and expressiveness. Information and Computation*, volume 104 pp. 35–77, 1993.

[Alu98] Rajeev Alur. *Timed automata*. Dans *Verification of Digital and Hybrid Systems, Proceedings of NATO-ASI Summer School* (Springer-Verlag, 1998). A paraître.

[Asa98] Eugene Asarin. *Equations on timed languages*, 1998. A paraître dans Proceedings of "Hybrid Systems".

[BLL+96] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, et Wang Yi. UPPAAL — *a tool suite for automatic verification of real–time systems. Rapport de recherche RS-96-58*, BRICS, Aalborg University, 1996.

[DGP97] Volker Diekert, Paul Gastin, et Antoine Petit. *Removing epsilon-transitions in timed automata.* Dans *Proceedings of STACS'97*, volume 1200 de *Lecture Notes in Computer Science*, pp. 583–594 (Springer-Verlag, 1997).

[HKW95] Thomas A. Henzinger, Peter W. Kopke, et Howard Wong-Toi. *The expressive power of clocks.* Dans *Proceedings of ICALP'95*, Lecture Notes in Computer Science, pp. 335–346 (Springer-Verlag, 1995).

[HNSY94] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, et Sergio Yovine. *Symbolic model-checking for real-time systems. Information and Computation*, volume 111 pp. 193–244, 1994.

[HSGL97] Klaus Havelund, Skou Skou, Larsen Kim G., et Kristian Lund. *Formal modeling and analysis of an audio/video protocol: An industrial case study using* UPPAAL. Dans *Proceedings of RTSS'97*. 1997.

[LLW95] François Laroussinie, Kim G. Larsen, et Carsten Weise. *From timed automata to logic—and back.* Dans *Proceedings of MFCS'95*, volume 969 de *Lecture Notes in Computer Science*, pp. 27–41 (Springer-Verlag, 1995).

[PP] Dominique Perrin et Jean-Éric Pin. *Mots infinis.* A paraître.

[Wil94] Thomas Wilke. *Specifying timed state sequences in powerful decidable logics and timed automata.* Dans *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 863 de *Lecture Notes in Computer Science*, pp. 694–715 (Springer-Verlag, 1994).

[WVS83] Pierre Wolper, Moshe Y. Vardi, et A. Prasad Sistla. *Reasoning about infinite computation paths.* Dans *Proceedings of FOCS'83*, pp. 185–194. 1983.