Rapport de stage
DEA Algorithmique 2003/2004

# Détection d'erreurs dans les systèmes temporisés

Fabrice CHEVALIER

sous la direction de Patricia Bouyer

Laboratoire Spécification et Vérification, ENS Cachan

# Table des matières

# Introduction

Il n'est maintenant plus nécessaire d'insister sur l'omniprésence de l'informatique dans notre vie quotidienne (bases de données, systèmes de contrôle/commande, technologies du transport, télécommunications ...). Or la composante logicielle est primordiale dans de nombreux systèmes critiques. Il est alors essentiel de s'assurer que les logiciels utilisés fonctionnent de manière irréprochable. La méthode communément employée par les industriels est de tester un certain nombre de scénarios pour vérifier la validité de l'outil créé. Mais l'adéquation à un jeu de tests ne permet pas de tester l'ensemble des situations pouvant se produire.

Certains échecs dramatiques comme la défaillance de l'appareil médical Therac-25, qui a provoqué la mort de plusieurs personnes par overdose de radiations, ont convaincu de la nécessité de tester de façon formelle les logiciels. C'est l'objectif que se fixe l'informatique de la vérification : construire un modèle formel du système étudié, ainsi qu'une formule exprimant la propriété devant être vérifiée par le système (par exemple la non-accessibilité d'un état de défaillance), et décider si le modèle vérifie ou non la formule.

Un modèle couramment utilisé est celui des sytèmes d'événements discrets (apparentés aux automates finis). Ce type de modèle permet d'étudier l'ordre relatif des actions et d'exprimer des propriétés du type «Déclencher l'alarme après qu'une erreur s'est produite», mais ne permet pas de faire intervenir le temps de manière quantitative ; des propriétés telle que «Déclencher l'alarme au plus 5 secondes après qu'une erreur s'est produite» ne peuvent pas être exprimées. Pour traiter de telles propriétés, le modèle des *automates temporisés* se révèle efficace. Il s'agit d'une extension des automates finis classiques : les automates temporisés sont enrichis d'un ensemble fini d'horloges qui évoluent toutes à la même vitesse.

Dans ce mémoire, nous nous intéressons à la détection d'erreurs dans les systèmes temporisés dont le cadre a été posé dans [Tri02] ; le sujet a été bien étudié dans le cas de systèmes à événements discrets (voir par exemple [SSL$^+$95] et [SSL$^+$96]) où l'outil principal est la déterminisation des automates considérés, mais de nouveaux problèmes se posent lorsque la composante temporelle est ajoutée.

Le problème que nous étudions est le suivant : un système étant donné, il peut être observé via un ensemble de capteurs qui n'est pas exhaustif, de sorte que certaines actions ne sont pas observables. Ce système peut également provoquer des erreurs que nous cherchons à repérer. La question qui se pose est : «Etant donné un système, est-il possible de détecter toute erreur au plus $\Delta$ secondes après qu'elle s'est produite ?».

Dans un premier temps nous présenterons le modèle des automates temporisés, puis nous rappellerons les principaux résultats sur l'observation de systèmes temporisés. Ensuite nous introduirons une méthode d'observation différente dite observation par automates : au lieu de chercher à observer le système en utilisant un algorithme général mais complexe, nous chercherons à observer le système de façon plus simple et plus efficace en utilisant des automates. Enfin, nous étudierons en détail et comparerons ce type d'observation dans deux classes d'automates temporisés : les automates déterministes et les automates *event-recording*.

# Chapitre 1

# Automates temporisés

Dans ce chapitre, nous présentons les modèles et définitions que nous utiliserons par la suite. Dans un premier temps nous définissons les automates temporisés, puis nous rappelons quelques résultats essentiels de décidabilité et d'indécidabilité sur ce modèle. Nous définissons ensuite le produit synchronisé d'automates qui nous servira à établir des algorithmes. Enfin nous présentons une sous-classes intéressante, celle des *event-recordings* et introduisons le concept de ressources bornées.

## 1.1 Le modèle

Le modèle des automates temporisés utilise celui des automates finis dans lequel des contraintes de temps sont ajoutées. Il est 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 tout ce qui suit, $\mathbb{T}$ est un ensemble de dates (par exemple $\mathbb{Q}^+$ ou $\mathbb{R}^+$). 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, aussi 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 :

$$g ::= \top \ | \ x \sim c \ | \ g_1 \wedge g_2$$

où $g_1$ et $g_2$ sont des gardes, $x \in X$, $c \in \mathbb{Q}^+$ 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 laquelle $x_i$ est remplacé par sa valeur $v(x_i)$.

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 $(\sigma, \tau)$ avec $\tau$ séquence de dates et $\sigma$ suite sur l'alphabet $A$ de même longueur que $\tau$. Le mot vide sera noté $\epsilon$.
- Un *langage temporisé* est un ensemble de mots temporisés. Pour $L$ langage temporisé, $L_*$ désigne l'ensemble des mots temporisés finis de $L$ et $L_\omega$ celui des mots temporisés infinis de $L$. Nous avons donc $L = L_* \cup L_\omega$.
- Un *mot symbolique* basé sur $(A, X, \mathcal{C}(X))$ est un élément de $(A \times \mathcal{C}(X) \times 2^X)^*$

Soit $L$ un langage temporisé sur l'alphabet $A$. $\textsc{Untime}(L)$ 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 $0,8$.
$\textsc{Untime}(L)$ est le langage non temporisé $(ab)^\infty = (ab)^* \cup (ab)^\omega$.

*Remarque :*
Pour les mots temporisés, nous utilisons parfois une autre notation où les temps donnés sont des délais plutôt que des dates : un mot temporisé peut être vu comme un élément de $(\mathbb{T}A)^\infty$. Un mot $w = t_1, a_1, t_2, a_2, \cdots$ correspond au mot $(\sigma, \tau)$ où $\forall\, 1 \leq i \leq |w|$ $\sigma_i = w_{2i}$ et $\tau_i = \sum_{j=1}^{i} w_{2j-1}$.
Par exemple les notations $(a; 0,9)(b; 2,1)(a; 3,4)(b; 4,3)$ et
$0.9, a, 1.2, b, 1.3, a, 0.9, b$ représentent le même mot.
Un mot temporisé $w = t_1, a_1, t_2, a_2, \cdots$ est dit *non-zénon* si $\sum t_i$ diverge.

Nous allons maintenant 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 proposée dans les années 1990 par Alur et Dill :

**Definition 1.1.1 [AD90, AD94]**
*Un* automate temporisé *est un 8-uplet* $\langle A, X, Q, I, F, R, T, Inv \rangle$ *où* $A$ *est un alphabet fini,* $X$ *un ensemble fini d'horloges,* $Q$ *est un ensemble fini d'états,* $I \subseteq Q$ *est un ensemble d'états initiaux,* $F \subseteq Q$ *est un ensemble d'états finaux,* $R \subseteq Q$ *est un ensemble d'états répétés,* $T \subseteq Q \times \mathcal{C}(X) \times A \times 2^C \times Q$ *est un ensemble de transitions (avec* $\mathcal{C}(X)$ *ensemble de gardes) et où* $Inv : Q \to \mathcal{C}(X)$ *une fonction qui à tout état associe un invariant.*
$\mathcal{A}ut(A)$ *désigne l'ensemble des automates temporisés sur l'alphabet* $A$*.*

*Remarque :*
Un automate temporisé sans états répétés est parfois vu comme un 7-uplet $\langle A, X, Q, I, F, T, Inv \rangle$

Une transition de l'automate temporisé précédent est de la forme $(q, g, a, r, q')$, 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 $q'$. La sémantique formelle sera donnée par la suite.

**Definition 1.1.2** *Un automate temporisé* $\langle A, X, Q, I, F, R, T, Inv \rangle$ *est dit* déterministe *si :*
- *il a un unique état initial, c'est-à-dire* $|I| = 1$
- *pour tout couple de transitions distinctes* $(q \xrightarrow{g_1,a,r_1} q'_1, q \xrightarrow{g_2,a,r_2} q'_2)$, $g_1$ *et* $g_2$ *s'excluent mutuellement (c'est à dire* $g_1 \wedge g_2$ *n'est pas satisfaisable)*

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 des horloges.

C'est cette dernière constatation qui nous amène à définir ce qu'est une configuration d'un automate temporisé $\mathcal{A}$ : une *configuration* de $\mathcal{A}$ 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.

**Definition 1.1.3** *Soit $\mathcal{A} = \langle A, X, Q, I, F, R, T, Inv\rangle$ un automate temporisé.*

- *Une* exécution *dans $\mathcal{A}$ sur le mot temporisé $(\sigma, \tau)$ est une suite (finie ou infinie) de configurations $(\langle q_i, v_i\rangle)_{i\in J}$ ($J = \mathbb{N}_n$ ou $\mathbb{N}$) vérifiant pour tout i :*

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

- *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 $L(\mathcal{A})$. 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, $L_*(\mathcal{A}) = L(\mathcal{A}) \cap (A \times \mathbb{T})^*$, ensemble des comportements finis de $\mathcal{A}$ et $L_\omega(\mathcal{A}) = L(\mathcal{A}) \cap (A \times \mathbb{T})^\omega$, ensemble des comportements infinis de $\mathcal{A}$.*
- *Le langague symbolique accepté par $\mathcal{A}$, noté $L_{symb}(\mathcal{A})$ est le langage de $\mathcal{A}$ vu comme un automate fini. C'est un ensemble de mots symboliques sur l'alphabet $\mathcal{C}(X) \times A \times 2^C$.*
- *Si $\gamma$ est un mot symbolique fini, nous notons $tw(\gamma)$ le langague temporisé de tout automate temporisé $\mathcal{A}$ tel que $L_{symb}(\mathcal{A}) = \{\gamma\}$. Il s'agit de l'ensemble des mots temporisés engendrés par $\gamma$.*

*Remarque :*
Sur un mot temporisé donné, un automate temporisé déterministe a au plus une exécution.

Voilà un exemple d'automate temporisé, l'état final est représenté par un double cercle, et $q_1$ est considéré comme état répété.

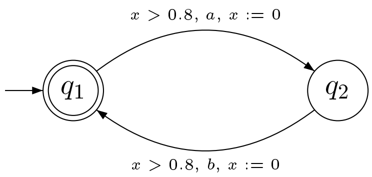



FIG. 1 – Exemple d'automate temporisé

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 $L_\omega$ sont aussi des langages reconnaissables.

Rappelons sans démonstration les propriétés de clôture des automates temporisés :

**Proposition 1.1 ([AD94])** *La classe des langages temporisés reconnaissables est stable par union, par intersection, mais pas par passage au complémentaire.*

## 1.2 Résultats de décidabilité et d'indécidabilité

Nous présentons maintenant quelques résultats de décidabilité et d'indécidabilité pour les automates temporisés dont nous nous servirons par la suite.

### 1.2.1 L'automate des régions

L'automate des régions est un outil puissant pour montrer la décidabilité de certaines propriétés sur les automates temporisés, il permet en particulier de décider le vide, ce que nous rappelons ci-dessous.

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).

**Lemme 1.2 ([AD94])** *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 1.2, 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 $k - 1 < x < k$ et $l - 1 < y < l$, 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. 2. 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 où les comportements possibles futurs vont être les mêmes.

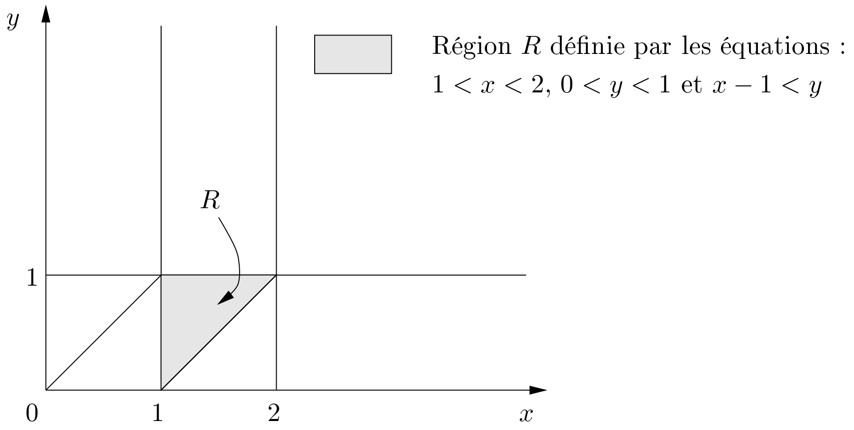



FIG. 2 – Les 28 régions dans le cas de 2 horloges $x$ et $y$ avec $c_x = 2$ et $c_y = 1$

Notation :
Soit $R$ une région, on note $\mathsf{Succ}(R)$ l'ensemble des régions $R'$ telles que il existe une valuation $v \in R$ et $t \in \mathbb{T}$ avec $v + t \in R'$. $\mathsf{Succ}(R)$ est l'ensemble des régions accessibles à partir de $R$ en laissant passer le temps.
Soit $R$ une région et $C \subseteq X$, on note ${}_{[C \leftarrow 0]}(R)$ la région $R'$ telle que il existe une valuation $v \in R$ avec ${}_{[C \leftarrow 0]}v \in R'$. ${}_{[C \leftarrow 0]}(R)$ est la région obtenue à partir de $R$ en remettant les horloges de $C$ à zéro.

Nous allons maintenant définir le graphe des régions :

**Definition 1.2.1** *Soit $\mathcal{A} = \langle A, X, Q, I, F, R, T, Inv \rangle$ un automate temporisé. Soit $\mathcal{R}$ l'ensemble des régions associées à $\mathcal{A}$. L'*automate des régions *associé à $\mathcal{A}$ est l'automate fini $\langle A, Q', I', F', R', T' \rangle$ défini par :*

$$\begin{cases} Q' = & Q \times \mathcal{R} \\ I' = & I \times 0^n \\ F' = & F \times \mathcal{R} \\ R' = & R \times \mathcal{R} \\ T' = & \{((q,R),a,(q',R')) \mid \exists (q,g,a,r,q') \in T \; \exists R'' \in \mathsf{Succ}(R) \text{ tels que} \\ & R, R'' \subseteq Inv(q),\; R' \subseteq Inv(q'),\; g \subseteq R'' \text{ et } R' = {}_{[r \leftarrow 0]}R''\} \end{cases}$$

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'automate des régions. Comme nous allons le voir, il va « reproduire » les comportements de $\mathcal{A}$ (dans le sens du théorème suivant).

**Théorème 1 ([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 $(\sigma, \tau)$ dans $\mathcal{A}$, il existe un chemin $((q_i, R_i))_{i \in J}$ sur $\sigma$ 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 $\sigma$ dans $\mathcal{R}(\mathcal{A})$, il existe un chemin $(\langle q_i, v_i \rangle)_{i \in J}$ sur $(\sigma, \tau)$ 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 1.3** *Soit $\mathcal{A}$ un automate temporisé. Soit $\mathcal{R}(\mathcal{A})$ l'automate des régions qui lui est associé. Alors $\mathcal{L}(\mathcal{R}(\mathcal{A})) = \textsc{Untime}(\mathcal{L}(\mathcal{A}))$.*

**Corollaire 1.4** *Le problème du vide pour les automates temporisés est PSPACE-complet.*

L'appartenance du problème à PSPACE résulte directement du corollaire 1.3 ; la preuve de PSPACE-difficulté se trouve dans [AD94] et utilise une réduction de l'acceptation d'une machine de Turing d'espace linéaire.

### 1.2.2 Le problème de l'universalité

Si le vide est décidable pour les automates temporisés, cela ne permet pas de conclure de même pour l'universalité, les automates temporisés n'étant pas clos par complémentaire. En fait, le problème de l'universalité se trouve être indécidable. La preuve ne sera pas détaillée ici (elle se trouve dans [AD94] et utilise une réduction du problème de l'exécution récurrente d'une machine à deux compteurs), mais le résultat sera utilisé plus loin.

**Théorème 2 ([AD94])** *Le problème suivant est indécidable : étant donné un automate temporisé sur un alphabet $\Sigma$, accepte-t-il tous les mots temporisés sur $\Sigma$ ?*

## 1.3 Produit synchronisé d'automates temporisés

Nous définissons ci-après le produit synchronisé d'automates. L'intuition de cette définition est que les deux automates évoluent indépendamment, sauf pour certaines actions (celles de l'alphabet $\Sigma'$) pour lesquelles ils doivent se synchroniser. Par exemple, une action synchronisée ne peut être prise dans le produit que si elle est possible dans les deux automates.

Soit $\mathcal{A}_1 = \langle \Sigma_1, X_1, Q_1, I_1, F_1, T_1, Inv_1 \rangle$, $\mathcal{A}_2 = \langle \Sigma_2, X_2, Q_2, I_2, F_2, T_2, Inv_2 \rangle$ deux automates temporisés et $\Sigma' \subseteq \Sigma_1 \cap \Sigma_2$.
En posant $\Sigma = \Sigma_1 \cup \Sigma_2$ ; le produit synchronisé $\mathcal{A}_1 ||_{\Sigma'} \mathcal{A}_2$ est défini comme étant l'automate temporisé $\langle \Sigma, X, Q, I, F, T, Inv \rangle$ avec :
- son espace d'états est $Q = Q_1 \times Q_2$
- ses états initiaux sont $I = I_1 \times I_2$
- ses états finaux sont $F = F_1 \times F_2$
- sa fonction de transitions est définie ainsi :

  - si $a \in \Sigma'$, pour $i \in \{1, 2\}$ $q_i, q'_i \in Q_i$ et $q_i \xrightarrow{g_i, a, Y_i} q'_i$ est une transition de $\mathcal{A}_i$ ; alors il existe une transition $(q_1, q_2) \xrightarrow{g_1 \wedge g_2, a, Y_1 \cup Y_2} (q'_1, q'_2)$ dans $\mathcal{A}_1 ||_{\Sigma'} \mathcal{A}_2$.
  - si $a \in \Sigma_1 \backslash \Sigma'$, $q_1, q'_1 \in Q_1$ et $q_1 \xrightarrow{g_1, a, Y_1} q'_1$ est une transition de $\mathcal{A}_1$ ; alors $\forall q_2 \in Q_2$ il existe une transition $(q_1, q_2) \xrightarrow{g_1, a, Y_1} (q'_1, q_2)$ dans $\mathcal{A}_1 ||_{\Sigma'} \mathcal{A}_2$.
  - si $a \in \Sigma_2 \backslash \Sigma'$, $q_2, q'_2 \in Q_2$ et $q_2 \xrightarrow{g_2, a, Y_2} q'_2$ est une transition de $\mathcal{A}_2$ ; alors $\forall q_1 \in Q_1$ il existe une transition $(q_1, q_2) \xrightarrow{g_2, a, Y_2} (q_1, q'_2)$ dans $\mathcal{A}_1 ||_{\Sigma'} \mathcal{A}_2$.

  La première condition correspond à un mouvement synchronisé de $\mathcal{A}_1$ et $\mathcal{A}_2$, les deux suivantes à un mouvement désynchronisé (respectivement dans $\mathcal{A}_1$ puis $\mathcal{A}_2$).

## 1.4 La sous-classe des automates *event-recording*

Nous rappelons dans cette partie les définitions et résultats de [AFH94] sur une sous-classe des automates temporisés : la classe des *event-recording*. Dans [AFH94] sont également présentés les classes non-déterministes des *event-predicting* et *event-recording*, mais nous ne les utiliserons pas car notre but est de considérer des observateurs déterministes.
Un automate *event-recording* sur un alphabet $\Sigma$ est un automate avec $|\Sigma|$ horloges implicites, une pour chaque action de $\Sigma$. Pour $a \in \Sigma$, l'horloge associée à $a$, notée $x_a$ contient le temps écoulé depuis la dernière occurence de l'événement $a$.
Un *event-recording* peut-être vu comme un automate temporisé classique :

toute transition étiquetée par $a$ remet à zéro la seule horloge $x_a$.
Formellement un automate *event-recording* est un automate temporisé $\mathcal{A} = \langle A, X, Q, I, F, R, T, Inv\rangle$ où $X = \{x_a \mid a \in A\}$ tel que si $q \xrightarrow{g,a,Y} q'$ est une transition de $\mathcal{A}$ alors $Y = \{x_a\}$.

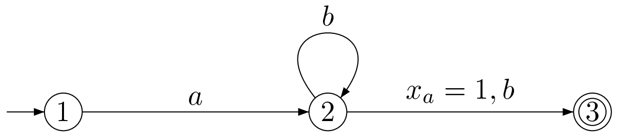



FIG. 3 – Exemple d'*event-recording*

Le langage reconnu par cet automate est $\{t_1, a, t_2, b, \cdots, t_k, b | t_k - t_1 = 1\}$.
Remarque : les remises à zéro $x_a := 0$ sur la première transition et $x_b := 0$ sur la boucle et la deuxième transition sont sous-entendues par les actions $a$ et $b$ dans le cas d'un *event-recording*.

Un automate *event-recording* est dit *déterministe* s'il l'est considéré en tant qu'automate temporisé classique. Nous rappelons maintenant une propriété essentielle des *event-recording* : ils peuvent être déterminisés.

**Proposition 1.5** *Soit $A$ un automate temporisé* event-recording*, alors il existe un automate temporisé* event-recording *déterministe $B$ tel que $L(B) = L(A)$.*

La preuve de [AFH94] ne sera pas explicitée ici, elle adapte la méthode des sous-ensembles. Nous donnons ci-dessous un *event-recording* déterministe équivalent à l'automate de la figure 3.

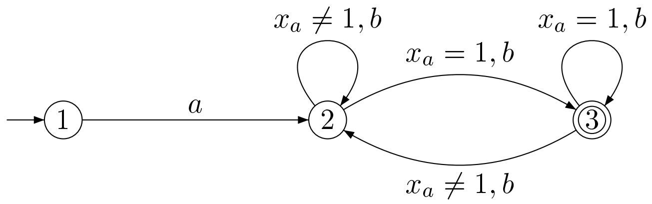



FIG. 4 – Déterminisation d'un *event-recording*

## 1.5 Ressources bornées

Plus tard, nous voudrons considérer des sous-classes d'automates où les constantes et le nombre d'horloges sont fixés. Pour cela nous utilisons les concepts de granularité et d'alphabet symbolique exposés ci-dessous.

Une *granularité* est un triplet $\mu = (X, m, max)$ où $X$ est un ensemble d'horloges, $m$ et $max$ des entiers positifs.

Une contrainte est une contrainte de $\mu = (X, m, max)$ si c'est une combinaison booléenne de contraintes de la forme $x \sim c$ où $x \in X$, $\sim \in \{=, <, >, \leq, \geq\}$ et $c = \frac{k}{m}$, $k \leq max$.

La *granularité d'un ensemble de contraintes* est le triplet $(X, m, max)$ où $X$ est l'ensemble des horloges mentionnées dans les contraintes, $m$ le plus petit multiple commun aux dénominateurs des constantes utilisées dans les contraintes, et $max$ la plus grande constante apparaissant dans les contraintes, multipliée par $m$.

La granularité d'un automate temporisé est celle de l'ensemble des contraintes utilisées sur ses transitions.

Une contrainte $g$ est dite *$\mu$-atomique* si c'est une contrainte non vide de $\mu$ telle que pour toute contrainte $g'$ de $\mu$, $[\![g]\!] \subseteq [\![g']\!]$ ou $[\![g]\!] \cap [\![g']\!] = \emptyset$.

Soit $\mu$ une granularité, *l'alphabet symbolique* sur $\mu$ est l'alphabet constitué de l'ensemble des contraintes $\mu$-atomiques.
Par exemple si $\mu = (\{x\}, 1, 1)$, l'alphabet symbolique de $\mu$ est $\{x = 0, 0 < x < 1, x = 1, x > 1\}$.

Nous définissons de la même façon la notion de granularité et d'alphabet symbolique pour un *event-recording*. Un *event-recording* n'ayant pas d'horloges propres, une granularité est dans ce cas un couple $\mu = (m, max)$ où $m$ et $max$ sont des entiers positifs.

# Chapitre 2

# Définitions et présentation des résultats sur l'observation

Dans ce chapitre nous rappelons les définitions et résultats sur l'observation de [Tri02]. Le cadre est le suivant : un système à observer est donné ; ce système a des événements observables et des événements non observables, un des événements non observable étant une faute. Le but est de construire un algorithme (dit observateur) qui en prenant en entrée la partie observable d'une exécution peut dire si une faute s'est produite dans le système. Dans le cadre de l'observation temporisée on se donne de plus un délai $\Delta$, le but de l'observateur est double : ne pas annoncer de faute s'il ne s'en est pas produit, et annoncer toute faute au plus $\Delta$ unités de temps après qu'elle s'est produite.

## 2.1 Exécutions $\Delta$-fautives

Soit $\Sigma$ un alphabet et $\Sigma' \subseteq \Sigma$, on définit l'opérateur de projection sur $\Sigma'$ ainsi : étant donné le mot temporisé (fini ou infini) $\rho$, $\pi_{\Sigma'}(\rho)$ est le mot temporisé obtenu en effaçant de $\rho$ tous les événements de $\Sigma \backslash \Sigma'$ et en sommant ensuite tous les délais entre deux événements successifs. Par exemple si $\Sigma = \{a, b, c, d, e\}$, $\Sigma' = \{a, c, e\}$ et $\rho = 1, a, 4, b, 1, c, 0, d, 3, e$ alors $\pi_{\Sigma'}(\rho) = 1, a, 5, c, 3, e$. Notons que pour tout $\rho$ et $\Sigma'$, $temps(\rho) = temps(\pi_{\Sigma'}(\rho))$.

**Definition 2.1.1** *Une exécution $\rho = \beta_1, \beta_2, \cdots$ est dite* fautive *s'il existe un $i \in \mathbb{N}$ tel que $\beta_i = f$.*
*Soit $j$ le plus petit $i$ tel que $\beta_i = f$ et soit $\rho' = \beta_j, \beta_{j+1}, \cdots$. Pour $\Delta \in \mathbb{N}$, $\rho$ est dit $\Delta$-fautif si $\Delta$ unités de temps passent après la première occurrence de $f$, c'est à dire si $temps(\rho') \geq \Delta$.*

## 2.2 Observateurs et observabilité

Dans cette partie, nous donnons les définitions d'observabilité pour un système.

Le cadre de travail est le suivant : le système produit une exécution $w$, notre but est de dire si cette exécution contient une faute. Certains événements de cette exécution sont observables et d'autres non ; notons $\Sigma_o$ les événements observables et $\Sigma_u$ les événements non-observables.

Nous n'avons pas accès à $w$ directement, mais à l'observation $\pi_{\Sigma_o}(w)$. A partir de cette observation uniquement, nous devons dire si une faute s'est produite ou pas.

Un sytème est dit *observable* s'il existe une fonction (appelée *observateur*) qui étant donné une observation annonce s'il y a eu une erreur ou pas.

Un *système* est un automate temporisé sur $\Sigma_o \cup \Sigma_u$ sans états finaux ni répétés, et avec la distinction entre alphabet des événements observables $\Sigma_o$ et non-observables $\Sigma_u$.

Formellement un systèmes un 7-uplet $\mathcal{P} = (\Sigma_o, \Sigma_u, Q, I, T, X, Inv)$ tel que $\langle \Sigma_o \cup \Sigma_u, X, Q, I, \emptyset, \emptyset, T, Inv \rangle$ est un automate temporisé.

Notons $\mathcal{MT}_\Sigma$ l'ensemble des mots temporisés finis sur $\Sigma$.

**Definition 2.2.1** *Soit* $\mathcal{P} = (\Sigma_o, \Sigma_u, Q, I, T, X, Inv)$ *un système et* $\Delta \in \mathbb{N}$. *Un* $\Delta$-observateur *de* $\mathcal{P}$ *est une fonction*

$$D : \mathcal{MT}_{\Sigma_o} \to \{0, 1\}$$

*telle que*

- *Pour toute exécution non fautive* $\rho$ *de* $\mathcal{P}$, $D(\pi_{\Sigma_o}(\rho)) = 0$
- *Pour toute exécution* $\Delta$*-fautive* $\rho$ *de* $\mathcal{P}$, $D(\pi_{\Sigma_o}(\rho)) = 1$

Remarque : aucune condition n'est imposée pour des exécutions fautives mais pas $\Delta$-fautives ; en effet un $\Delta$-observateur doit annoncer toutes les fautes au plus $\Delta$ unités de temps après qu'elles se sont produites, pas nécessairement immédiatement.
Un système est dit $\Delta$-*observable* s'il existe un $\Delta$-observateur pour ce système. Un système est dit *observable* s'il est $\Delta$-observable pour au moins un $\Delta$.
A partir de maintenant, lorsqu'il n'y a pas d'ambiguïté, $\pi_{\Sigma_o}$ sera noté $\pi$ (puisque les projections seront toujours faites sur les événements observables).

*Exemple :*
L'automate suivant est 3-observable :

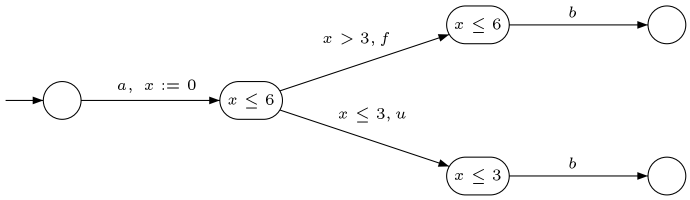



FIG. 5 – Exemple de système observable

Un observateur $D$ convenant est :

$$D : \quad \mathcal{MT}_{\Sigma_o} \longrightarrow \{0,1\}$$
$$w \longmapsto \begin{cases} 1, \text{ si } w = t_1, a, t_2, b \text{ où } t_2 > 3 \\ 0, \text{ sinon} \end{cases}$$

Nous allons maintenant donner une condition nécessaire et suffisante de $\Delta$-observabilité, pour cela nous définissons tout d'abord les ensembles d'exécutions fautives et $\Delta$-fautives :

$L^{\mathcal{P}}_{\neg f} = \{\rho$ exécution de $\mathcal{P}/\ \rho$ est non fautive$\}$

$L^{\mathcal{P}}_{\Delta f} = \{\rho$ exécution de $\mathcal{P}/\ \rho$ est $\Delta$-fautive$\}$

Lorsqu'il n'y a pas d'ambigüïté, ils seront notés simplement $L_{\neg f}$ et $L_{\Delta f}$.

**Proposition 2.1** *Soit $\mathcal{P}$ un système et $\Delta \in \mathbb{N}$, $\mathcal{P}$ est $\Delta$-observable si et seulement si $\pi(L_{\neg f}) \cap \pi(L_{\Delta f}) = \emptyset$.*

*Exemple :*
Cette caractérisation nous permet d'affirmer que l'automate suivant n'est pas 3-observable :

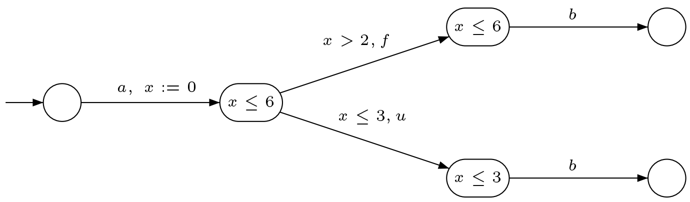



FIG. 6 – Exemple de système non observable

En effet, l'observation $a, 2.6, b, 3$ peut provenir de l'exécution $a, 2.5, u, 0.1, b, 3$ qui est dans $L_{\neg f}$ et de l'exécution $a, 2.5, f, 0.1, b, 3$ qui est dans $L_{\Delta f}$ ($\Delta = 3$). Ainsi $\pi(L_{\neg f}) \cap \pi(L_{\Delta f})$ ($\Delta = 3$) contient donc au moins le mot temporisé $a, 2.6, b, 3$ et donc n'est pas vide. Le système n'est pas 3-observable. Mais un raisonnement analogue fonctionnerait pour tout $\Delta$ (prendre l'observation $a, 2.6, b, 3, \Delta$).

## 2.3 Tester l'observabilité

Dans cette partie, nous présentons deux algorithmes pour tester l'observabilité et la $\Delta$-observabilité. Nous donnons les résultats sans preuves, celles-ci se trouvent dans [Tri02].

Pour éviter d'avoir à traiter des systèmes trop pathologiques, nous supposerons à présent que tous les sytèmes étudiés sont bien-fondés : il existe une exécution non-zénon (c'est à dire sur un mot temporisé non-zénon) à partir de tout état accessible du système. C'est une hypothèse peu restrictive car en pratique le temps peut toujours s'écouler et un modèle de système raisonnable doit vérifier cette propriété.

D'après la proposition 2.1, un système est observable si et seulement si il n'y a pas d'observation pouvant provenir à la fois d'une exécution non fautive et d'une exécution $\Delta$-fautive de $\mathcal{P}$. Pour tester l'observabilité, il semble donc naturel de mettre $\mathcal{P}$ en parallèle avec lui-même en le synchronisant sur les événements observables. Les exécutions dans ce produit synchronisé correspondront alors aux couples d'exécutions de $\mathcal{P}$ ayant la même partie observable.
Nous définissons $(\mathcal{P}||_{\Sigma_o}\mathcal{P})^{-f_2}$ de la manière suivante : les événements non-observables $u \in \Sigma_u$ sont renommés $u_1$ dans la première composante et $u_2$ dans la deuxième. Le produit $\mathcal{P}||_{\Sigma_o}\mathcal{P}$ est alors réalisé comme en 1.3, enfin les transitions étiquetées $f_2$ sont supprimées.
La présence d'une exécution $\Delta$-fautive dans $(\mathcal{P}||_{\Sigma_o}\mathcal{P})^{-f_2}$ correspond alors intuitivement à la non-observabilité : il existe dans ce cas une observation provenant à la fois d'une exécution $\Delta$-fautive (dans la première composante nécessairement puisque les $f_2$ ont été supprimés) et d'une exécution non fautive (correspondant au chemin effectué dans la deuxième composante où il n'y a pas d'erreurs).

**Proposition 2.2** *Soit $\mathcal{P}$ un système bien-fondé, $\mathcal{P}$ est observable si et seulement si toute exécution fautive de $(\mathcal{P}||_{\Sigma_o}\mathcal{P})^{-f_2}$ est zénon.*

La méthode pour tester la $\Delta$-observabilité est analogue mais utilise une troisième composante. Soit $Obs(\Delta)$ l'automate suivant :

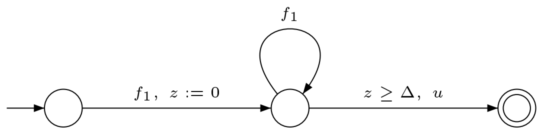



FIG. 7 – L'automate $Obs(\Delta)$

**Proposition 2.3** *Soit $\mathcal{P}$ un système. $\mathcal{P}$ est $\Delta$-observable si et seulement si l'état accepteur de $(\mathcal{P}||_{\Sigma_o}\mathcal{P})^{-f_2}||_{f_1}Obs(\Delta)$ n'est pas accessible.*

Nous rappelons maintenant l'évaluation de la complexité du problème de l'observation :

**Problème 1** *Etant donné un système $\mathcal{P}$ et $\Delta \in \mathbb{N}$, $\mathcal{P}$ est-il $\Delta$-observable ?*

**Problème 2** *Etant donné un système $\mathcal{P}$, $\mathcal{P}$ est-il observable ?*

**Théorème 3 ([Tri02])** *Les problèmes 1 et 2 sont PSPACE-complets.*

## 2.4 Construction d'un observateur

Nous avons vu dans la partie précédente comment tester l'observabilité et la $\Delta$-observabilité. Si un système est observable, il est intéressant en pratique de pouvoir construire un observateur, c'est à dire un algorithme qui prenant en entrée un mot temporisé répond si celui-ci provient d'une exécution fautive ou non.

L'idée de la construction d'un observateur proposée dans [Tri02] est de créer un «estimateur d'états» : à chaque information reçue du système (à savoir l'attente d'un délai ou l'occurrence d'un événement observable), l'observateur estime dans quels états peut se trouver le système. Si à un moment donné le système ne peut être que dans des états fautifs, l'observateur annoncera une faute.

Nous ne donnons pas l'algorithme formellement ici, mais indiquons quelles fonctions sont utilisées.

Soit $\mathcal{R}$ l'ensemble des régions du système ; à tout moment, c'est à dire après avoir lu une observation quelconque, nous voulons savoir dans quel sous-ensemble $S \subseteq \mathcal{R}$ de régions le système peut se trouver sachant qu'il a généré une telle observation. Pour cela nous utilisons les fonctions $R_o$ et $R_u$ qui nous permettent de mettre à jour l'ensemble $S$ de régions possibles après la réception d'une nouvelle information (par exemple le système a généré l'événement $a$).

$R_o : 2^{\mathcal{R}} \times \Sigma_o \longrightarrow 2^{\mathcal{R}}$

$R_o(S, a)$ correspond à l'ensemble des régions possibles après avoir lu un $a$ si l'ensemble des régions possibles avant de lire ce $a$ était $S$.

$R_u : 2^{\mathcal{R}} \times \mathbb{Q} \longrightarrow 2^{\mathcal{R}}$

$R_u(S, \delta)$ correspond à l'ensemble des régions possibles après avoir lu un délai $\delta$ si l'ensemble des régions possibles avant de lire ce délai était $S$. La fonction $R_u$ est calculable en utilisant un algorithme d'accessibilité pour un automate temporisé : $R_u(S, \delta)$ correspond à l'ensemble des régions accessibles à partir de $S$ en exactement $\delta$ unités de temps et en n'utilisant que des actions non observables dans le système.

Avec $R_o$ et $R_u$ nous mettons à jour au fur et à mesure des informations reçues l'ensemble $S$ de régions possibles. Une faute est déclarée si toutes les régions de $S$ sont fautives.

La propriété de correction prouvée dans [Tri02] est que s'il existe un $\Delta$-observateur pour le système, alors cet algorithme est un $\Delta$-observateur.

# Chapitre 3

# Observation par des automates

Dans ce chapitre, nous introduisons le cadre dans lequel nous allons travailler. Alors que la méthode d'observation proposée dans le chapitre précédent propose de construire un observateur qui est un algorithme quelconque, nous proposons de construire des observateurs moins puissants, mais plus efficaces et facilement manipulables : des automates.

Dans un premier temps, nous expliquons la raison de ce choix, puis nous proposons une première définition d'observation par automates : l'observation précise. Nous montrons alors qu'elle n'est pas algorithmiquement satisfaisante et nous posons alors la définition d'observation par automates que nous utiliserons par la suite.

## 3.1 Motivation

Les définitions d'observabilité données dans [Tri02] et présentées brièvement dans le chapitre 2 sont très générales, elles correspondent au problème «Etant donné un système, existe-t-il un algorithme annonçant les fautes de ce système». L'algorithme obtenu peut être complexe et coûteux, c'est pourquoi nous proposons de chercher un algorithme sous forme d'automate annonçant les fautes.

La méthode utilisée dans [Tri02] pour décider si un mot donné est fautif ou non a une complexité qui dépend du mot : plus le mot est long plus l'algorithme est coûteux. Cela peut être un inconvénient si l'on cherche à détecter les erreurs du système en temps-réel, c'est pourquoi nous proposons une méthode de détection des erreurs «à la volée» sous la forme d'un automate : chaque fois qu'une nouvelle observation est générée, l'automate change d'état selon cette observation.

Etude comparative des complexités (on notera $\mathcal{P}$ le système et $w$ le mot lu) :

– L'algorithme utilisé dans [Tri02] crée un graphe des régions de taille

exponentielle en $\mathcal{P}$ et $w$, et pour chaque lettre lue calcule l'ensemble des régions dans lesquelles $\mathcal{P}$ peut se trouver. La complexité totale est donc double exponentielle en $\mathcal{P}$ et en $w$.
- Considérons un observateur sous forme d'automate temporisé tel que nous le proposons existe et soit $N$ sa taille. En supposant que l'on puisse faire la construction de cet automate lors d'un précalcul, la complexité lors de la lecture d'une lettre est logarithmique en $N$.
  Nous verrons que lorsqu'ils existent les observateurs que nous proposons sont de taille double exponentielle en $\mathcal{P}$. La complexité totale est donc exponentielle en $\mathcal{P}$ et linéaire en $w$.

## 3.2 Observation précise par automates

### 3.2.1 Définition

En regardant comment sont construits les observateurs dans la partie 2.4, nous nous apercevons que ceux-ci peuvent à tout moment décider si l'exécution courante est fautive, non fautive, ou encore si les deux cas sont possibles en fonction des informations à leur disposition (i.e. les événements observables).

Ce constat permet de dégager une première définition du rôle d'un observateur : il doit indiquer à tout moment l'ensemble des informations qu'il peut avoir. Plus précisément, l'observateur doit dire si le mot lu provient nécessairement d'une exécution fautive, s'il provient nécessairement d'une exécution non fautive, ou s'il peut provenir à la fois d'une exécution fautive et non fautive.

Nous nous proposons alors de modéliser un tel observateur $\theta$ par un automate temporisé ayant 3 types d'états : $f$, $\neg f$ et ? correspondant respectivement aux 3 cas ci-dessus.
Si $\theta$ est un tel automate, nous notons $L_f(\theta),\ L_{\neg f}(\theta)$ *et* $L_?(\theta)$ les langages de $\theta$ en prenant respectivement pour états finaux les états de type $f$, $\neg f$ et ?.

**Definition 3.2.1** *Soit $C$ une classe d'automates temporisés et $\mathcal{P}$ un système.*
*$\mathcal{P}$ est dit* observable de façon précise dans $C$ *s'il existe $\theta \in C$ tel que :*
- $L_f(\theta) = \pi(L_f)\backslash\pi(L_{\neg f})$
- $L_{\neg f}(\theta) = \pi(L_{\neg f})\backslash\pi(L_f)$
- $L_?(\theta) = \pi(L_{\neg f}) \cap \pi(L_f)$

### 3.2.2 Un résultat d'indécidabilité

Un observateur précis est très puissant : il indique à tout moment l'ensemble des informations disponibles. Malheureusement, nous allons montrer ci-après que la construction d'un tel observateur n'est pas algorithmiquement possible, ce qui justifiera que nous nous intéressions à d'autres types d'observateurs par la suite.

**Problème 3** *Etant donné un système $\mathcal{P}$, $\mathcal{P}$ est-il observable de façon précise par un automate temporisé déterministe $\theta$ ? Si oui construire un tel $\theta$.*

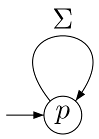
**Proposition 3.1** *Le problème 3 n'est pas calculable.* [1]

**Preuve :**
Raisonnons par l'absurde : supposons que le problème 3 est calculable, et montrons que sous ces hypothèse le problème de l'universalité pour les automates temporisés est décidable.
Soit donc un machine de Turing $M$ qui calcule le problème 3, et notons $M(P)$ le résultat calculé par $M$ sur l'entrée $P$ ($M(P)$ est soit un observateur précis $\theta$ pour $\mathcal{P}$, soit $\perp$). Nous affirmons que l'algorithme suivant résoud le problème de l'universalité.

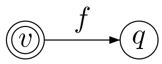
---

\# Entrée($A$ :Automate temporisé sur l'alphabet $\Sigma_A$)

\#     Rajouter l'état et la transition suivante à $A$ :




\#     Rajouter un état $q$ à $A$
\#     Pour tout état final $v$ de $A$, rajouter la transition suivante :




\#     En posant $\Sigma_o = \Sigma_A$ et $\Sigma_u = \{f\}$, calculer $M(A)$
\#     Si $M(A) = \perp$, répondre «$A$ n'est pas universel»
\#     Si $M(A) = \theta \neq \perp$, tester l'universalité de $L_?(\theta)$
\#         Si $L_?(\theta)$ est universel, répondre «$A$ est universel»
\#         Si $L_?(\theta)$ n'est pas universel, répondre «$A$ n'est pas universel»

---

[1] Par abus de langage, on dit qu'un problème est calculable quand la fonction implicitement associée au problème est calculable. Par exemple pour le problème 3, cette fonction prend en entrée un système $\mathcal{P}$ et renvoie soit un observateur précis de $\mathcal{P}$, soit $\perp$ s'il n'en existe pas.

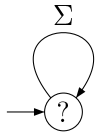
Remarque : L'universalité de $L_?(\theta)$ peut être testée car $\theta$ est un automate déterministe.

*Preuve de l'algorithme :*

– Montrons que si $A$ est universel, alors l'algorithme renvoie «$A$ est universel».

  Supposons $A$ universel et notons $A'$ l'automate obtenu après l'ajout de $p$, $q$ et des transitions correspondantes.

  Dans $A'$ toute observation peut provenir d'une exécution fautive (car $A$ est universel) ou d'une exécution non fautive (qui boucle sur $p$) ; donc $A$ admet pour observateur l'automate suivant :




  $M(A)$ est donc différent de $\perp$. Il reste à montrer que pour un observateur quelconque $\theta$ (pas nécessairement celui ci-dessus), nous avons que $L_?(\theta)$ est universel. C'est une conséquence immédiate de la propriété de $A'$ exposée ci-dessus : toute observation peut provenir d'une exécution fautive ou d'une exécution non fautive.

– Montrons que si $A$ n'est pas universel, alors l'algorithme renvoie «$A$ n'est pas universel».

  Supposons $A$ non universel, il suffit de montrer que le cas où $M(A) = \theta \neq \perp$ et $L_?(\theta)$ est universel est impossible. En effet, dans ce cas, soit $w$ un mot non reconnu par $A$. L'observation $w$ ne peut provenir que d'une exécution non fautive de $A'$ (car $w$ n'est pas reconnu par $A$) ; donc tout observateur de $A'$ doit se trouver dans un état de type $\neg f$ après avoir lu $w$, $L_?(\theta)$ ne peut donc être universel car $w \notin L_?(\theta)$.

$\square$

Remarques :

– la preuve d'indécidabilité ci-dessus s'applique à d'autres classes d'automates, la propriété essentielle étant que l'universalité puisse être testée sur cette classe. En particulier le problème de construction d'un automate *event-recording* observant un système n'est pas algorithmiquement possible.

– plus tard nous nous intéresserons à l'existence d'un observateur à ressources bornées, c'est à dire dans une classe d'automates où sont fixés le nombre d'horloges et les constantes utilisées. Même dans ce cadre restreint, la preuve s'applique toujours et donc l'observation précise n'est pas algorithmiquement possible ; ce qui justifie que nous nous intéressions par la suite à l'observation générale et plus à l'observation précise.

## 3.3 Définition de l'observation par automates

La construction d'un observateur précis n'est pas algorithmiquement possible, nous allons donc introduire une nouvelle notion d'observation appelée «observation par automates». Nous considérons maintenant que le rôle d'un observateur n'est pas de donner l'ensemble des informations pouvant être déduites des observations, mais bien de dire si une exécution est fautive ou non, et ce dans un délai imparti $\Delta$.

Un observateur sera alors un automate temporisé (le plus souvent déterministe) avec des états finaux et non finaux ; si l'automate se trouve dans un état final, cela signifie qu'il annonce une faute. Les conditions que nous imposons sur cet observateur sont : de ne pas annoncer de faute s'il ne s'en n'est pas produit, et d'annoncer toute faute au plus $\Delta$ unités de temps après qu'elle s'est produit.

Nous donnons ci-dessous la définition d'observabilité par une classe quelconque d'automates temporisés ; en pratique les classes que nous considérerons seront : la classe des automates temporisés déterministes, la classe des *event-recording*, la classe des automates temporisés déterministes d'une granularité donnée et la classe des *event-recording* d'une granularité donnée. Dans tous les cas, l'observateur noté $\theta$ sera un automate temporisé sur l'alphabet des observables $\Sigma_o$, il lit les observations et s'il arrive dans un état final annonce une faute.

Rappelons que :
$\pi$ est la projection sur l'ensemble des événements observables $\Sigma_o$
$L_{\neg f} = \{\rho$ exécution de $\mathcal{P}/\ \rho$ est non fautive$\}$
$L_{\Delta f} = \{\rho$ exécution de $\mathcal{P}/\ \rho$ est $\Delta$-fautive$\}$

**Definition 3.3.1** *Soit $C$ une classe d'automates temporisés, $\mathcal{P}$ un système et $\Delta \in \mathbb{N}$.*
*$\mathcal{P}$ est dit* $\Delta$-observable dans *$C$ s'il existe $\theta \in C$ tel que*

$$\pi(L_{\Delta f}) \subseteq L(\theta) \subseteq \pi(L_{\neg f})^c$$

Remarque : $L(\theta)$ correspond à l'ensemble des observations pour lesquelles l'observateur annonce une faute.
La condition $\pi(L_{\Delta f}) \subseteq L(\theta)$ exprime que si une exécution est $\Delta$-fautive, l'observateur annoncera une faute ; la condition $L(\theta) \subseteq \pi(L_{\neg f})^c$ exprime que $\theta$ n'annonce pas une faute s'il n'y en a pas eue. Ces deux conditions correspondent à celles de la définition 2.2.1.

Une première propriété naturelle est que tout système observable dans une classe d'automates est observable dans le cas général.

**Proposition 3.2** *Soit $C$ une classe d'automates temporisés, $\mathcal{P}$ un système et $\Delta \in \mathbb{N}$.*
*Si $\mathcal{P}$ est $\Delta$-observable dans $C$ alors $\mathcal{P}$ est $\Delta$-observable.*

**Preuve :**
$\mathcal{P}$ étant observable dans $C$, il existe $\theta \in C$ tel que $\pi(L_{\Delta f}) \subseteq L(\theta) \subseteq \pi(L_{\neg f})^c$ ; en particulier $\pi(L_{\Delta f}) \cap \pi(L_{\neg f}) = \emptyset$ □

Tout système observable dans une classe d'automates est donc observable dans le cas général, la réciproque n'est pas vraie : nous donnons un exemple de système observable dans le cas général, mais non-observable par automates déterministes, ce qui justifiera que l'on s'intéresse à l'observation par automates déterministes dans la prochaine partie.

**Proposition 3.3** *Pour tout $\Delta > 0$, le système suivant est $\Delta$-observable, mais pas $\Delta$-observable par automates déterministes :*

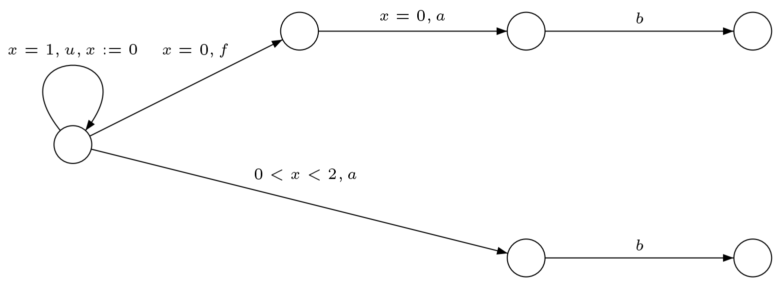



FIG. 8 – Système $\mathcal{P}$ observable, mais pas par automates déterministes

**Preuve :**
Pour tout $\Delta > 0$, $\mathcal{P}$ est $\Delta$-observable : il suffit d'annoncer une faute sur $\{t_1, a, t_2, b \mid t_1 \in \mathbb{N}\}$.

Soit $\Delta \in \mathbb{N}$, raisonnons par l'absurde et supposons que $\mathcal{P}$ soit $\Delta$-observable par un automate déterministe $\theta$. Nous allons alors montrer que le langage temporisé $\{t_1, a, t_2, b \mid t_1 \in \mathbb{N} \text{ et } t_2 \geq \Delta\}$ est reconnu par un automate temporisé déterministe, ce qui est absurde ; en effet, on peut montrer par un raisonnement élémentaire sur les automates temporisés que le langage $\{t_1, a, t_2, b \mid t_1 \in \mathbb{N} \text{ et } t_2 \geq \Delta\}$ ne peut être reconnu par un automate temporisé sans $\varepsilon$-transitions.
Il vient facilement que :
$\pi(L_{\Delta f}) = \{t_1, a, t_2, b \mid t_1 \in \mathbb{N} \text{ et } t_2 \geq \Delta\}$
et $\pi(L_{\neg f}) = \{t_1, a, t_2, b \mid t_1 \notin \mathbb{N}\} \cup \{t_1, a \mid t_1 \notin \mathbb{N}\}$
$\mathcal{P}$ étant $\Delta$-observable par $\theta$, on a $\pi(L_{\Delta f}) \subseteq L(\theta) \subseteq \pi(L_{\neg f})^c$.
Posons $L_0 = \{t_1, a, t_2, b \mid t_2 \geq \Delta\}$, il est facile de voir que $L_0$ est un langage

temporisé déterministe.
Remarquons que $\pi(L_{\Delta f}) \subseteq L_0$ et $L_0 \cap \pi(L_{\neg f})^c = \pi(L_{\Delta f})$.
Il vient :
$L_0 \cap \pi(L_{\Delta f}) \subseteq L_0 \cap L(\theta) \subseteq L_0 \cap \pi(L_{\neg f})^c$,
d'où $\pi(L_{\Delta f}) = L_0 \cap L(\theta)$. Les langages temporisés déterministes étant clos par intersection, $\pi(L_{\Delta f})$ est un langage temporisé déterministe ce qui est absurde. $\mathcal{P}$ n'est donc pas $\Delta$-observables par automates temporisés déterministes, et ce pour tout $\Delta > 0$. $\square$

# Chapitre 4

# Observation par des automates déterministes

Nous nous intéressons dans cette partie à l'existence d'un observateur sous la forme d'un automate temporisé déterministe. Nous n'avons pas réussi à résoudre le problème dans le cas général (c'est à dire demander l'existence d'un observateur déterministe sans spécifier de condition supplémentaire). Nous avons par contre trouvé un algorithme pour décider l'existence d'un observateur déterministe de ressources fixées.

## 4.1 Observation à ressources fixées : résolution en 2-EXPTIME

Dans un premier temps nous présentons le problème de l'observation à ressources fixées ainsi qu'un lemme utile. Ensuite nous rappelons la définition du jeu que nous utiliserons dans notre algorithme, avant de présenter et de prouver l'algorithme lui-même.

### 4.1.1 Le problème

La définition d'observation à ressources fixées est celle donnée en 3.3.1 en prenant pour classe d'automates celle des automates temporisés de la granularité fixée, mais pour plus de clarté nous rappelons la définition dans ce cadre.

**Definition 4.1.1** *Soit* $\mu = (X, \frac{1}{m}, M)$ *une granularité,* $\mathcal{P}$ *un système et* $\Delta \in \mathbb{N}$.
$\mathcal{P}$ *est dit* $\Delta$-observable par automates temporisés déterministes de granularité $\mu$ *s'il existe* $\theta$ *automate temporisé déterministe de granularité* $\mu$ *tel que*

$$\pi(L_{\Delta f}) \subseteq L(\theta) \subseteq \pi(L_{\neg f})^c$$

**Problème 4** *Etant donné une granularité $\mu$, un sytème $\mathcal{P}$ et un entier $\Delta$, $\mathcal{P}$ est-il $\Delta$-observable par automates temporisés déterministes de granularité $\mu$ ?*

Tout d'abord, nous allons démontrer un lemme essentiel qui nous servira dans cette partie mais également par la suite : il existe des automates temporisés avec $\varepsilon$-transition pour les langages $\pi(L_{\neg f})$ et $\pi(L_{\Delta f})$.

**Lemme 4.1** *Soit $\mathcal{P}$ un système, des automates temporisés avec $\varepsilon$-transitions $\mathcal{P}_{\neg f}$ et $\mathcal{P}_{\Delta f}$ tels que $L(\mathcal{P}_{\neg f}) = \pi(L_{\neg f})$ et $L(\mathcal{P}_{\Delta f}) = \pi(L_{\Delta f})$ sont constructibles en temps polynomial.*

**Preuve :**
Pour $\mathcal{P}_{\neg f}$ c'est très simple : il suffit de supprimer les transitions étiquetées par $f$ (pour interdire les fautes) et de remplacer les transitions étiquetées par $u \in \Sigma_u$ par une $\varepsilon$-transition. En prenant tous les états comme finaux, il vient $L(\mathcal{P}_{\neg f}) = \pi(L_{\neg f})$.

Supposons que $\mathcal{P} = (\Sigma_o, \Sigma_u, Q, I, T, X, \mathcal{I})$, $\mathcal{P}_{\Delta f}$ est construit comme suit : il s'agit de 3 copies de $\mathcal{P}$ où sont remplacées par des $\varepsilon$-transitions les transitions non-observables. Toute faute dans la première copie envoie dans la deuxième copie où l'automate passe $\Delta$ unités de temps ; quand ces $\Delta$ unités de temps sont écoulées, l'automate passe dans la troisième copie (dont tous les états sont finaux). Intuitivement, ce sont bien les exécutions $\Delta$-fautives qui sont reconnues, car pour atteindre un état final il faut faire une faute puis attendre au moins $\Delta$ unités de temps.

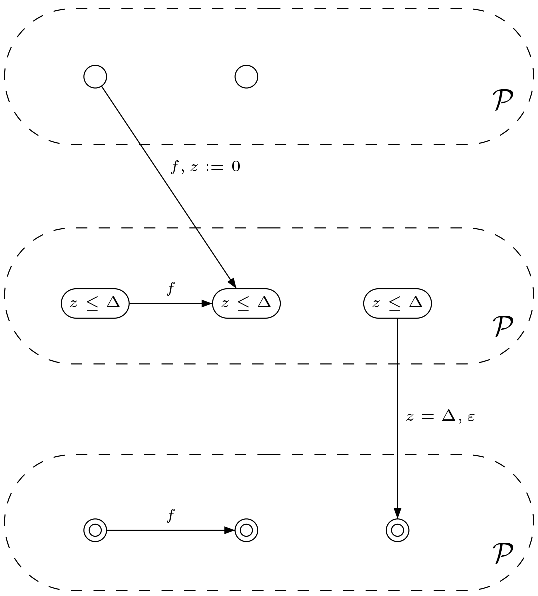



FIG. 9 – L'automate $\mathcal{P}_{\Delta f}$

*Remarque :*
les transitions non-étiquetées correspondent à des transitions de $\mathcal{P}$.

Formalisons cette construction :
Soit $\mathcal{P} = (\Sigma_o, \Sigma_u, Q, I, T, X, Inv)$ ; $\mathcal{P}_{\Delta f} = (\Sigma_o, Q', I, F', T', X', Inv')$ où
- $Q' = Q_1 \cup Q_2 \cup Q_3$ où les $Q_i$ sont des copies de $Q$
- $F' = Q_3$
- $X' = X \cup \{z\}$ où $z$ est une horloge supplémentaire n'appartenant pas à $X$ qui permettra de vérifier que $\Delta$ unités de temps sont passées dans la deuxième composante.
- Les invariants $Inv'$ sont ceux de $Inv$ auxquels sont ajoutés «$z \leq \Delta$» dans les états de $Q_2$

Explicitons la fonction de transition $T'$. Pour $q, q' \in Q$ et $i \in \{1, 2, 3\}$ seront notés $q_i, q'_i$ les copies de $q, q'$ dans la $i^{\text{ème}}$ composante. Soit $\mathcal{P}$' le système $\mathcal{P}$ où toutes les transitions non-observables (sauf $f$) ont été remplacées par des $\varepsilon$-transitions.

- Si $q \xrightarrow{g,a,Y} q'$ dans $\mathcal{P}$' où $a \neq f$ alors $q_i \xrightarrow{g,a,Y} q'_i$ pour $i \in \{1, 2, 3\}$,
- Si $q \xrightarrow{g,f,Y} q'$ dans $\mathcal{P}$' alors $q_i \xrightarrow{g,f,Y} q'_i$, pour $i \in \{2, 3\}$,
- Si $q \xrightarrow{g,f,Y} q'$ dans $\mathcal{P}$' alors $q_1 \xrightarrow{g,f,Y\cup\{z:=0\}} q'_2$,
- $\forall q \in Q$ $q_2 \xrightarrow{z=\Delta,\varepsilon,} q_3$ (le passage dans la troisième composante est possible quand $\Delta$ unités de temps se sont écoulées dans la deuxième composante).

Il est aisé de montrer que $L(\mathcal{P}_{\Delta f}) = \pi(L_{\Delta f})$ (la construction est ad-hoc).
□

*Remarque :*
En fait dans cet automate sont présents $\pi(L_{\Delta f})$ mais aussi $\pi(L_{\neg f})$ : en prenant comme états finaux les états de $Q_1$, $\mathcal{P}_{\Delta f}$ reconnaît $\pi(L_{\neg f})$. Par la suite on qualifiera donc les états de $Q_1$ d'états «non fautifs» et ceux de $Q_3$ d'états «$\Delta$-fautifs».

### 4.1.2 Rappel sur les jeux

Nous résoudrons le problème en le mettant sous forme de jeu, c'est pourquoi nous rappelons ici la définition (inspirée de [Zie98]) d'un jeu classique d'accessibilité.

Un *jeu classique d'accessibilité* est un 5-uplet $\Phi = (V_0, V_1, i, E, W)$ où
- $V_0$ et $V_1$ sont des ensembles finis d'états appartenant respectivement au joueur 0 et au joueur 1. L'union de $V_0$ et $V_1$ est notée $V$.
- $i \in V_0 \cup V_1$ est l'état initial.
- $E \subseteq V_0 \times V_1 \cup V_1 \times V_0$ est l'ensemble des arêtes. $vE = \{v' \in V_0 \cup V_1 | (v, v') \in E\}$ est l'ensemble des successeurs de $v$.

– $W \subseteq Q$ est l'ensemble des états perdants pour le joueur 0.

Le jeu se déroule ainsi : il commence dans l'état initial ; à chaque étape le joueur dont c'est le tour choisit le prochain état selon $E$. Les joueurs construisent ainsi une séquence finie d'états (les séquences infinies ne nous sont pas utiles). Cette séquence est dite gagnante pour le joueur 0 si aucun des états parcourus n'appartient à $W$.

Nous supposons connues les définitions de stratégie gagnante et stratégie positionnelle (ou sans mémoire), voir par exemple [Zie98]. Rappelons le résultat dont nous nous servirons : si un joueur a une stratégie gagnante dans $\Phi$, alors il a une stratégie gagnante positionnelle. Celle-ci peut se calculer par un algorithme de marquage classique.

Graphiquement, le joueur 0 sera représenté par un carré, le joueur 1 par un cercle et les états perdants pour le joueur 0 par un double cercle.
*Exemple :*

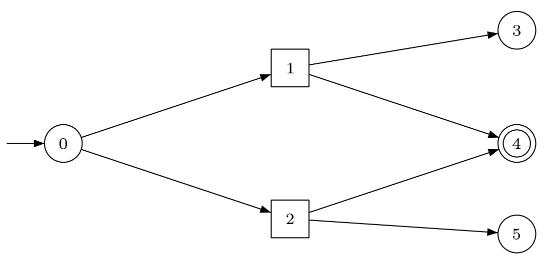



FIG. 10 – Exemple de jeu classique d'accessibilité

Dans le jeu de la figure 10, le joueur 0 (le carré) a une stratégie gagnante positionnelle qui consiste à choisir 3 en 1 et à choisir 5 en 2.

### 4.1.3 Résolution

La méthode utilisée pour résoudre le problème 4 est inspirée de [BDMP03]. Pour résoudre le problème de l'observation, nous allons l'exprimer sous forme de jeu classique d'accessibilité : l'observateur correspondra au joueur 0 et le système jouera le rôle du joueur 1 en essayant de mettre l'observateur en défaut. Le système sera observable si et seulement si le joueur 0 (l'observateur) a une stratégie gagnante dans le jeu.

Soit $\Gamma$ l'alphabet symbolique sur $\mu$ et $\Sigma_o$. $\mathcal{U}_\Gamma$ est l'automate temporisé universel sur $\Gamma$, il a un unique état et reconnaît tous les mots symboliques de $\Gamma^*$.
Soit $\mathcal{R}$ le graphe des régions de $\mathcal{P}_{\Delta f} \mid\mid_{\Sigma_o} \mathcal{U}_\Gamma$. La «projection» de $\mathcal{R}$ sur l'alphabet $\Gamma$ est définie ci-après.

Soit $(g, a, Y)$ une étiquette de transition de $\mathcal{P}_{\Delta f} \; ||_{\Sigma_o} \; \mathcal{U}_\Gamma$, soit $g'$ l'unique garde sur $\mu$ telle que $g \subseteq g'$ et soit $Y' \subseteq Y$ les horloges de $\mu$ incluses dans $Y$ ; alors $(g, a, Y) \upharpoonright \Gamma$ est défini par $(g', a, Y')$ si $a \in \Sigma_o$ et $\varepsilon$ si $a \in \Sigma_u$.
L'automate «projection» $\mathcal{R} \upharpoonright \Gamma$ est l'automate $\mathcal{R}$ dans lequel on a remplacé toute transition $s \xrightarrow{(a,g,Y)} s'$ par $s \xrightarrow{(a,g,Y)\upharpoonright\Gamma} s'$.

Considérons $\mathcal{R} \upharpoonright \Gamma$ comme un automate fini non déterministe avec $\varepsilon$-transitions sur l'alphabet symbolique $\Gamma$, il peut alors être déterminisé par la méthode des sous-ensembles, soit $\mathcal{K}$ l'automate déterministe obtenu.

Pour obtenir le jeu qui nous intéresse $G_{\mathcal{P},\mu,\Delta}$, il ne reste plus qu'une seule transformation à faire sur $\mathcal{K}$. Le joueur 0 correspond à l'observateur, et le joueur 1 à l'environnement (au système) ; il semble naturel que le joueur 1 choisisse les actions discrètes du système et le temps auquel elles sont effectuées (donc la garde vérifiée) et que le joueur 0 connaissant cette action et ce temps, choisisse quelles remises à zéro sont effectuées dans l'observateur. La transformation suivante correspond à cette idée :

- l'espace d'états de $G_{\mathcal{P},\mu,\Delta}$ est $Q_\mathcal{K} \cup Q'$, où $Q'$ est composé de tous les états $(q, g, a)$ , $\forall q \in Q_\mathcal{K}$, $\forall g$ garde de $\mu$, $\forall a \in \Sigma_o$.
- S'il y a dans $\mathcal{K}$

$$q \xrightarrow{\quad g, a, Y \quad} q'$$

  alors il y a dans $G_{\mathcal{P},\mu,\Delta}$

$$q \xrightarrow{\quad g, a \quad} \boxed{q, g, a} \xrightarrow{\quad g, a, Y \quad} q'$$

  C'est donc bien le joueur 1 (rond) qui choisit le comportement du système et le joueur 0 (carré) qui choisit les remises à zéro dans l'observateur.
- les état perdants pour le joueur 0 sont les $\{(q_1, R_1), \cdots, (q_k, R_k)\}$ où il existe $q_i$ non fautif et $q_j$ $\Delta$-fautif.

**Proposition 4.2** *Soit $\mathcal{P}$ un système, $\mu$ une granularité et $\Delta \in \mathbb{N}$, $\mathcal{P}$ est $\Delta$-observable par automates temporisés déterministes de granularité $\mu$ si et seulement si le joueur 0 a une stratégie gagnante dans le jeu $G_{\mathcal{P},\mu,\Delta}$.*

**Preuve :**
Supposons que $\mathcal{P}$ soit $\Delta$-observable par un automate temporisé déterministe $\theta$, montrons que le joueur 0 a une stratégie gagnante dans le jeu $G_{\mathcal{P},\mu,\Delta}$. Quitte à le compléter, $\theta$ est supposé complet en tant qu'automate temporisé.
Nous définissons une stratégie non positionnelle $S$ à partir de $\theta$ : soit $\gamma =$

$(g_1,a_1)(g_1,a_1,Y_1)\cdots(g_k,a_k)$ une exécution dans le jeu, définissons $S(\gamma)$.
$\theta$ étant déterministe, la suite $(g_1,a_1)(g_2,a_2)\cdots(g_k,a_k)$ «mène» à un unique état $q$ de $\theta$ dans le sens où il existe une unique suite de remises à zéro $Y'_1,\cdots,Y'_k$ et un unique chemin $(g_1,a_1,Y'_1)(g_2,a_2,Y'_2)\cdots(g_k,a_k,Y'_k)$ dans $\theta$ qui mène à $q$. En $q$, il y a une unique transition correspondant à la garde $g$ et l'événement $a$ ; soit $(g,a,Y)$ cette transition, alors $S(\gamma)=(g,a,Y)$.
Montrons que cette stratégie est gagnante pour le joueur 0, c'est à dire que toute partie conforme à cette stratégie ne passe jamais par un état perdant pour le joueur 0. Raisonnons par l'absurde, supposons que lors d'une partie $\gamma=(g_1,a_1)(g_1,a_1,Y_1)\cdots(g_k,a_k,Y_k)$ conforme à $S$, l'état de $G_{\mathcal{P},\mu,\Delta}$ soit $\{(q_1,R_1),\cdots,(q_k,R_k)\}$ où $q_i$ est non fautif et $q_j$ est $\Delta$-fautif. Alors $q_i$ et $q_j$ sont accessibles par $\gamma$ dans $\mathcal{R}\restriction\Gamma$, donc il existe $w_1$ non fautif et $w_2$ $\Delta$-fautif tels que $w_1,w_2\in tw(\gamma)$, ce qui est absurde car sur $\gamma$, $\theta$ devrait annoncer à la fois «non fautif» (à cause de $w_1$) et «fautif» (à cause de $w_2$).

Montrons l'autre implication : supposons que le joueur 0 ait une stratégie gagnante dans le jeu $G_{\mathcal{P},\mu,\Delta}$, et montrons que $\mathcal{P}$ est $\Delta$-observable par un automate temporisé déterministe de granularité $\mu$.
S'il existe une stratégie gagnante pour le joueur 0, il existe une stratégie gagnante positionnelle déterministe ; soit $S$ une telle stratégie. A tout état appartenant au joueur 0 elle associe une unique transition de cette état.
Nous allons «compacter» le graphe correspondant à la stratégie $S$ en effaçant les transitions entrant dans un état appartenant au joueur 0 de la manière suivante :
S'il y a dans le graphe de la stratégie :

$$q \xrightarrow{\quad g,a \quad} \boxed{q,g,a} \xrightarrow{\quad g,a,Y \quad} q'$$

nous ajoutons dans le graphe «compacté»

$$q \xrightarrow{\quad g,a,Y \quad} q'$$

Remarque : cela est possible car la stratégie $S$, pour tout état appartenant au joueur 0 associe une unique transition partant de cet état.
Soit $\theta$ le graphe «compacté», en considérant comme états finaux les états $\{(q_1,R_1),\cdots,(q_k,R_k)\}$ où au moins un $q_i$ est $\Delta$-fautif. $\theta$ est un automate temporisé sur la granularité $\mu$, montrons que c'est un observateur de $\mathcal{P}$.
Soit donc $w$ un mot non fautif de $\mathcal{P}$, montrons que $\theta$ annonce «non fautif» sur $w$ ; c'est à dire montrons que si $\theta(\pi(w))=\{(q_1,R_1),\cdots,(q_k,R_k)\}$, aucun $q_i$ n'est $\Delta$ fautif. En effet, comme $w$ est non fautif, $\pi(w)$ peut mener à un état non fautif de $\mathcal{R}\restriction\Gamma$ et ce quelles que soient les remises à zéro faites sur

les horloges supplémentaires de $\Gamma$ ; donc dans $\mathcal{K}$, $\theta(\pi(w))$ contient au moins un $q_i$ non fautif, et $S$ étant une stratégie gagnante pour le joueur 0, le fait de contenir un $q_i$ non fautif implique de ne contenir aucun $q_i$ $\Delta$ fautif.
De même, si $w$ est $\Delta$ fautif, $\theta$ annonce «fautif».
$\theta$ est donc un $\Delta$-observateur de $\mathcal{P}$ de granularité $\mu$. □

De la construction précédente, nous obtenons le résultat de complexité suivant :

**Proposition 4.3** *Le problème 4 est dans 2-EXPTIME.*

**Preuve :**
La caractérisation sous forme de jeu de la proposition 4.2 nous donne immédiatement le résultat : pour tester si $\mathcal{P}$ est $\Delta$-observable par un automate temporisé déterministe de granularité $\mu$, il suffit de tester si le joueur 0 a une stratégie gagnante dans le jeu $G_{\mathcal{P},\mu,\Delta}$. Or $G_{\mathcal{P},\mu,\Delta}$ est le déterminisé de $\mathcal{R} \upharpoonright \Gamma$ il est donc de taille double exponentielle en $\mathcal{P}, \Delta, \mu$. Il s'agit d'un jeu que l'on peut résoudre par un algorithme de marquage classique qui utilise un temps polynomial.

□

Remarque : la preuve de la proposition 4.2 montre que si $\mathcal{P}$ est $\Delta$-observable par un automate automate temporisé déterministe de granularité $\mu$, un observateur $\theta$ de taille double-exponentielle est effectivement constructible à partir d'une stratégie gagnante pour le joueur 0 dans le jeu $G_{\mathcal{P},\mu,\Delta}$.

## 4.2 Observation à ressources fixées : preuve de 2-EXPTIME-difficulté

Pour montrer que l'observation à ressources bornées est 2-EXPTIME-difficile, on va réduire le problème de l'acceptation d'une machine de Turing alternante d'espace exponentiel. Pour cela, comme dans la preuve que SAT est NP-difficile, on va utiliser des formules logiques pour modéliser la suite de transitions de la machine de Turing.

On va procéder en plusieurs étapes : dans un premier temps on va relier la satisfaisabilité d'une formule logique à l'observabilité d'un système ; ensuite, en utilisant ce passage de la logique à l'observabilité, on montrera que le problème de l'observation est EXPTIME-difficile ; enfin, en modifiant cette dernière preuve, on montrera que le problème de l'observation est 2-EXPTIME-difficile.

### 4.2.1 Vérifier une formule logique avec un observateur à une horloge

#### Modéliser une variable du calcul propositionnel

On considère un observateur $\theta$ à une horloge qui observe le système suivant :

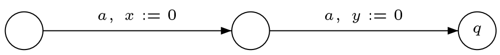



FIG. 11 – Choix d'une variable

Arrivé en $q$, $\theta$ ne peut avoir retenu qu'une seule des deux horloges $x$ ou $y$ ; on prend pour convention que si $\theta$ a retenu $x$ alors il a fait le choix de mettre la variable $x_1$ à vrai, et à faux s'il a retenu $y$.

**Definition 4.2.1** *On dit que $\theta$ a choisi $v$ ($v = x$ ou $y$) s'il a fait sa dernière remise à zéro sur la transition mettant $v$ à zéro.*

#### Tester la valeur de vérité d'une variable

On veut imposer à l'observateur de mettre la variable $x_1$ à vrai. On va construire un système $\mathcal{P}$ tel que tout observateur potentiel de $\mathcal{P}$ n'a pu choisir que $x$.

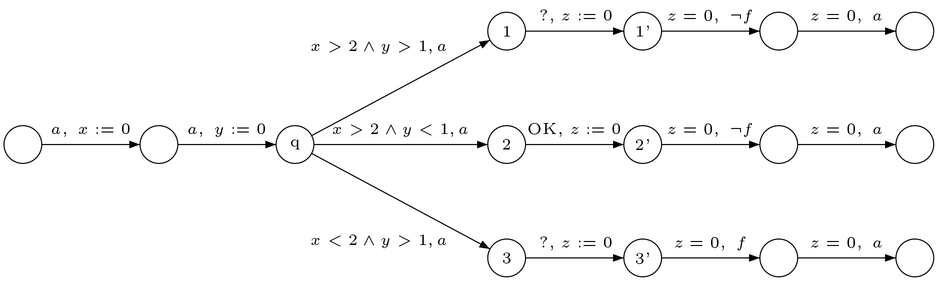



FIG. 12 – Système $\mathcal{P}$ imposant de choisir $x$

**Proposition 4.4** *Il existe un observateur à une horloge choisissant $x$ qui observe $\mathcal{P}$.*

**Preuve :**
Voici un observateur à une horloge pour $\mathcal{P}$ :

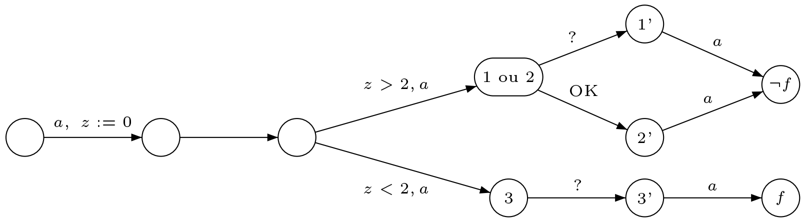



FIG. 13 – Observateur pour $\mathcal{P}$

□

**Proposition 4.5** *Un observateur à une horloge ne peut observer $\mathcal{P}$ que s'il a choisi $x$.*

Preuve informelle :
L'idée générale est que si $\theta$ choisit $x$, il risque de ne pas savoir s'il est en 1 ou en 2, par contre s'il est en 3 il le sait. Si $\theta$ a choisi $y$, il risque d'hésiter entre 1 et 3, mais s'il est en 2 il le sait.
Pour forcer $\theta$ à choisir $x$ on va lui donner une information supplémentaire sous la forme d'un «OK» ou d'un « ?» qui lèvera l'indétermination «1 ou 2» mais pas l'indétermination «1 ou 3».
Ainsi si $\theta$ a choisi $x$ il saura dans quel état il se trouve et pourra observer ; par contre s'il a choisi $y$ l'indétermination empêchera $\theta$ de décider entre fautif ou non fautif.

- Si $\theta$ choisit $x$, après avoir lu $(a, t_1)(a, t_2)(a, t_3)(?, t_4)$ 2 cas sont possibles : soit $t_3 - t_1 < 2$ alors $\theta$ sait qu'il est dans l'état 3' et donc qu'il va se produire une faute, soit $t_3 - t_1 > 2$ alors $\theta$ sait qu'il peut être en 1' ou en 2', mais comme il a lu « ?», il sait qu'il est en 1' (le dernier caractère lu a levé l'indétermination).
- Par contre si $\theta$ choisit $y$, et que $t_3 - t_2 > 1$, $\theta$ sait qu'il peut-être en 1' ou en 3', le « ?» ne levant pas l'indétermination, il ne peut observer correctement.

Remarque : Pour le cas d'une variable, le fait de lever l'indétermination par un «OK» ou d'un « ?» n'est pas nécessaire ; mais elle permet de mieux fixer ce que sait l'observateur et sera nécessaire lorsqu'on aura plusieurs variables.

Preuve formelle :
Soit $\theta$ un observateur ne choisissant pas $x$ et soit $\mu$ son pas. On veut montrer

que $\theta$ ne peut observer $\mathcal{P}$, on va donc exhiber 2 mots temporisés $u$ et $u'$, l'un fautif et l'autre non fautif tels que $\theta$ donne la même réponse sur $u$ et $u'$.

Soit $\varepsilon$ tel que $3\varepsilon < \mu$, on considère les 2 mots suivants $u = (a,t_1)(a,t_2)(a,t_3)(?,t_4)$ et $u' = (a,t'_1)(a,t'_2)(a,t'_3)(?,t'_4)$ où

| | |
|---|---|
| $t_1 = \varepsilon$ | $t'_1 = 3\varepsilon$ |
| $t_2 = 1$ | $t'_2 = 1$ |
| $t_3 = 2 + 2\varepsilon$ | $t'_3 = 2 + 2\varepsilon$ |
| $t_4 = 3$ | $t'_4 = 3$ |

$u$ est non fautif car $t_3 - t_1 > 2$ et $t_3 - t_2 > 1$ ; $u'$ est fautif car $t_3 - t_1 < 2$ et $t_3 - t_2 > 1$.

Un observateur fait des tests lors des transitions et change d'état selon le résultat de ces tests. Comme $\theta$ n'a qu'une horloge, le seul test qu'il peut faire dépend de la valeur courante de son horloge ; lors de la ième transition la valeur de cette horloge est $t_i - t_j (j < i)$ où $j$ est la dernière transition lors de laquelle $\theta$ a remis son horloge à zéro.
Par exemple, un observateur ne remettant jamais aucune horloge à zéro fera des tests sur $t_1, t_2, t_3, t_4$ ; un observateur remettant son horloge à zéro uniquement en même temps que $y$ fera des tests sur $t_1, t_2, t_3 - t_2, t_4 - t_2$.
Une propriété commune à tous les observateurs ne choisissant pas $x$ est qu'ils ne testent pas $t_3 - t_1$.

Informellement, le seul changement significatif entre $u$ et $u'$ pour un observateur de granularité $\mu$ concerne $t_3 - t_1$ ; donc dans un observateur ne choisissant pas $x$ (donc qui ne testera jamais $t_3 - t_1$), $u$ et $u'$ prendront le même chemin ; un tel $\theta$ ne peut donc observer $\mathcal{P}$.

L'argument formel vient de la propriété suivante vérifiée par $u$ et $u'$ :
$\forall j < i$ si $(i,j) \neq (3,1)$ alors $t_i - t_j$ et $t'_i - t'_j$ vérifient les mêmes gardes de $\theta$. Donc $u$ et $u'$ prennent le même chemin dans $\theta$, ce qui prouve que $\theta$ n'est pas un observateur de $\mathcal{P}$.

### Modéliser une clause

On cherche maintenant à construire un système $\mathcal{P}$ correspondant à une clause de type $x_1 \vee x_2 \vee x_3$. On veut que $\theta$ ait 3 choix à faire (correspondant à $x_1, x_2$ et $x_3$) et que seuls les observateurs faisant les choix rendant $x_1 \vee x_2 \vee x_3$ puissent observer correctement $\mathcal{P}$.
On associe à chaque $x_i$ deux horloges : $x_1^i$ qui signifie «mettre $x_i$ à vrai» et $x_0^i$ qui signifie «mettre $x_i$ à faux». Par exemple, si $\theta$ décide de choisir $x_1^1$, $x_0^2$ et $x_0^3$,cela correspond à mettre $x_1$ à vrai et $x_2$ et $x_3$ à faux.

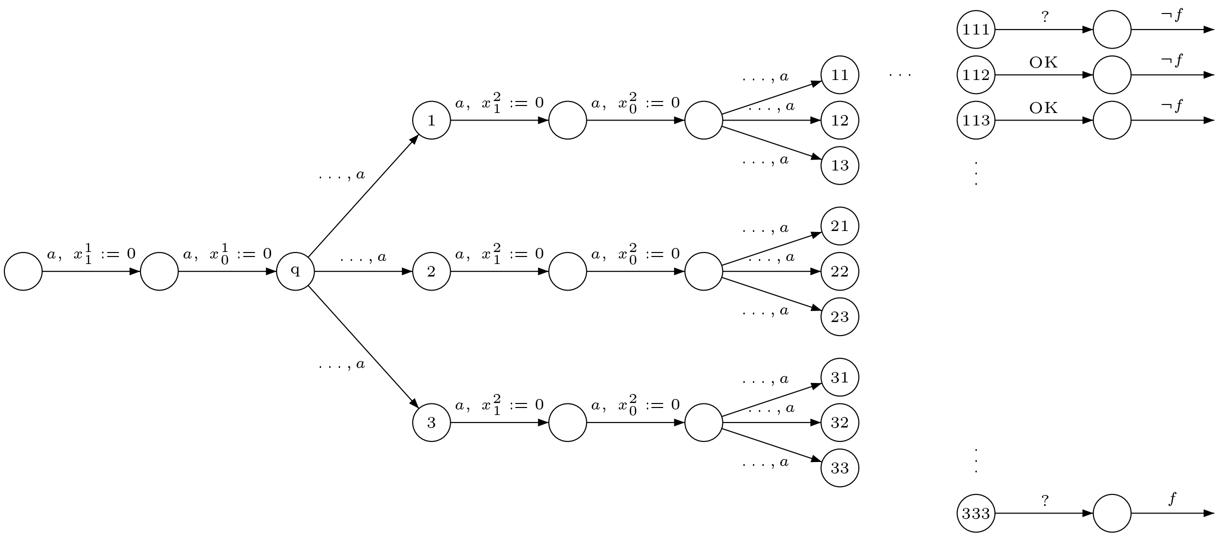



FIG. 14 – Système $\mathcal{P}$ pour une clause

Remarques sur la construction :

- Par souci de lisibilité, les $\ldots, a$ n'ont pas été explicités, ils représentent les contraintes correspondantes à celles utilisées pour une variable, par exemple $x_1^1 - 1 > 2 \wedge x_0^1 > 1, a$.
- De même, on a omis le troisième branchement correspondant à $x^3$.
- A partir de l'état 111 on lit « ?» puis «$\neg f$» ; à partir de 333 on lit « ?» puis «$f$» ; à partir des 25 états restants on lit «OK» puis «$\neg f$».

**Proposition 4.6** *Un observateur à une horloge ne peut observer $\mathcal{P}$ que s'il a choisi de mettre au moins un des $x_i$ à vrai.*

Preuve informelle :
Si $\theta$ décide de choisir tous les $x_i$ faux alors il peut se retrouver dans une situation où il ne sait pas si $\mathcal{P}$ est en 1-1-1 ou en 3-3-3, le « ?» ne lèvera pas l'indétermination et donc $\theta$ ne pourra observer correctement.

Preuve formelle :
Soit $\theta$ un observateur qui choisit de mettre $x_1$, $x_2$ et $x_3$ à faux, c'est à dire qu'il décide dans les 3 cas de stocker la deuxième horloge.
Par le même argument que dans le cas d'une seule variable, $\theta$ ne testera jamais ni $t_3 - t_1$, ni $t_6 - t_4$, ni $t_9 - t_7$. On va donc construire $u$ et $u'$, $u$ non fautif et $u'$ fautif tels que sauf pour $t_3 - t_1$, $t_6 - t_4$ et $t_9 - t_7$, $\forall j < i\ t_i - t_j$

et $t'_i - t'_j$ vérifient les mêmes gardes de $\theta$. $u$ et $u'$ prendront donc le même chemin dans $\theta$, ce qui prouvera que $\theta$ n'est pas un observateur correct de $\mathcal{P}$.

Preuve formelle :
Soit $\mu$ la granularité de $\theta$ et soit $\varepsilon$ tel que $9\varepsilon < \mu$, on considère les 2 mots suivants $u = (a, t_1)(a, t_2)(a, t_3)(a, t_4)(a, t_5)(a, t_6)(a, t_7)(a, t_8)(a, t_9)(?, t_{10})$ et $u' = (a, t'_1)(a, t'_2)(a, t'_3)(a, t'_4)(a, t'_5)(a, t'_6)(a, t'_7)(a, t'_8)(a, t'_9)(?, t'_{10})$ où

| | |
|---|---|
| $t_1 = \varepsilon$ | $t'_1 = 3\varepsilon$ |
| $t_2 = 1$ | $t'_2 = 1$ |
| $t_3 = 2 + 2\varepsilon$ | $t'_3 = 2 + 2\varepsilon$ |
| $t_4 = 2 + 3\varepsilon$ | $t'_4 = 2 + 5\varepsilon$ |
| $t_5 = 3$ | $t'_5 = 3$ |
| $t_6 = 4 + 4\varepsilon$ | $t'_6 = 4 + 4\varepsilon$ |
| $t_7 = 4 + 5\varepsilon$ | $t'_7 = 4 + 7\varepsilon$ |
| $t_8 = 5$ | $t'_8 = 5$ |
| $t_9 = 6 + 6\varepsilon$ | $t'_9 = 6 + 6\varepsilon$ |
| $t_{10} = 7$ | $t'_{10} = 7$ |

Remarque : il s'agit d'une généralisation du contre-exemple utilisé pour une variable, $t'_i = t_i$ sauf pour $t_1$, $t_4$ et $t_7$ où $t'_i = t_i + 2\varepsilon$
On a bien la propriété $\forall j < i$ si $(i, j) \notin \{(3, 1), (6, 4), (9, 7)\}$ alors $t_i - t_j$ et $t'_i - t'_j$ vérifient les mêmes gardes de $\theta$, ce qui contredit que $\theta$ est un observateur correct de $\mathcal{P}$ car $u$ est non fautif et $u'$ est fautif.

**Proposition 4.7** *Pour toute valuation $v$ des variables satisfaisant la clause, il existe un observateur $\theta$ faisant les choix de $v$ qui observe $\mathcal{P}$.*

**Preuve :**
On présente la construction sur un exemple, la construction dans le cas général est analogue.
On considère la clause $C = x_1 \vee x_2 \vee x_3$ et la valuation $v$ telle que
$v(x_1) = T,\ v(x_2) = F,\ v(x_3) = T$.
L'idée de la construction de $\theta$ est la suivante :

- Si on lit un «OK», on est sûr qu'il n'y a pas eu de faute.
- Si on lit un « ?», celui-ci permet de lever l'indétermination sur l'état (et ce car $v$ est une valuation satisfaisant $\mathcal{P}$) ; sachant dans quel état $\mathcal{P}$ se trouve, on sait si un erreur va se produire ou non.

Remarques sur la construction :

- On n'a pas représenté les arêtes «OK» qui vont toutes vers l'état non fautif.
- Le nom des états correspond à la connaissance qu'a $\theta$ de l'état actuel.

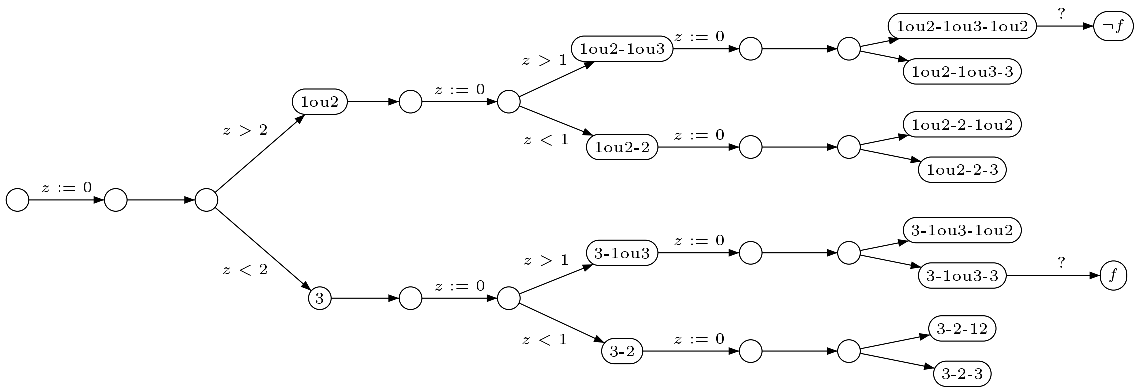



Fig. 15 – Observateur pour $x_1 \vee x_2 \vee x_3$ choisissant $v$

$\square$

Remarque : on a montré ici comment construire un système pour $x_1 \vee x_2 \vee x_3$ ; on peut construire de façon analogue un système pour toute clause de 3-SAT. Il s'agit du même $\mathcal{P}$ avec 3 branchements, la seule différence étant l'état qui fait la faute :

- Quelle que soit la clause, à partir de l'état 111 on lit « ?» puis «$\neg f$»
- L'état à partir duquel on lit « ?» puis «$f$» dépend de la clause, et plus précisément de l'unique valuation qui ne satisfait pas la clause. Par exemple pour $x_1 \vee x_2 \vee x_3$ il s'agissait de 333, pour $x_1 \vee \neg x_2 \vee x_3$ il s'agirait de 323.
- à partir des 25 états restants on lit «OK» puis «$\neg f$».

### Modéliser une formule de 3-SAT

Le but de ce paragraphe est de réduire 3-SAT à l'observation à ressources bornées ; à partir d'une formule $\phi$ de 3-SAT, on va construire $\mathcal{P}_\phi$ tel que $\mathcal{P}_\phi$ est observable par un observateur à une horloge si et seulement si $\phi$ est satisfaisable.

On a montré ci-dessus comment modéliser une clause de la forme $x_1 \vee x_2 \vee x_3$. Le système $\mathcal{P}$ construit pour cette clause à 3 variables a environ $3^3$ états car pour chaque variable il y a un branchement. Pour une formule de 3-SAT avec $n$ variables, une généralisation brutale de $\mathcal{P}$ conduit à un système de taille exponentielle, ce qui n'est pas souhaitable.
Pour éviter cette explosion, on va utiliser le fait que chaque clause d'une formule de 3-SAT ne fait appel qu'à 3 variables, dans le gadget correspondant

à une clause, les autres variables ne sont pas pertinentes, on ne va donc faire un branchement que sur les variables de la clause obtenant finalement 3 branchements au lieu de $n$.

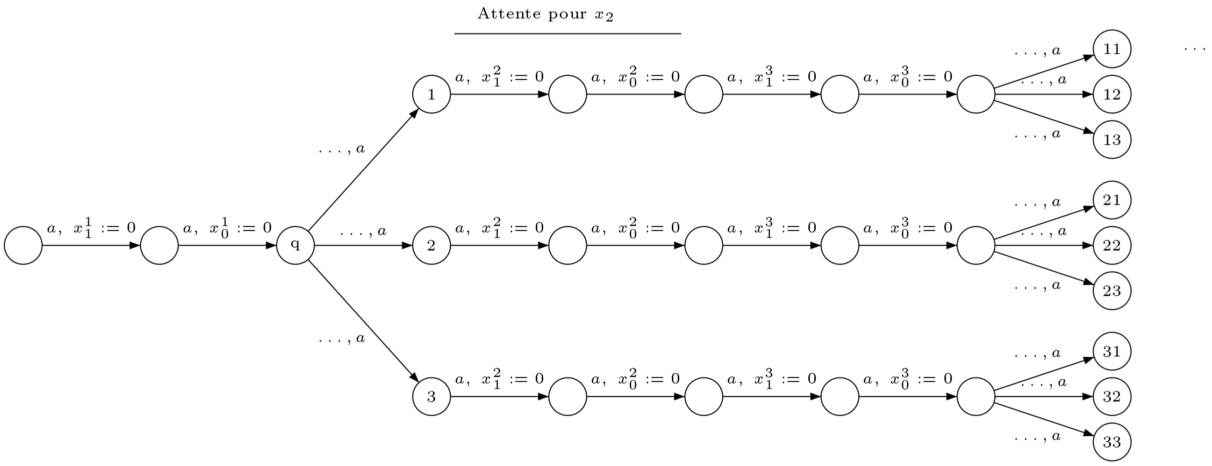



FIG. 16 – Système $\mathcal{P}$ pour une clause contenant $x_1$, $x_3$ et ne contenant pas $x_2$

Pour $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_k$ on utilise le système suivant :

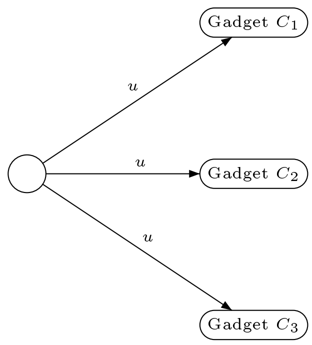



FIG. 17 – Système pour une formule de 3-SAT

$u$ étant non observable, l'observateur $\theta$ ne sait pas dans quel gadget $\mathcal{P}_\phi$ se trouve, et doit faire ses choix (correspondant aux valeurs de vérité des variables) sans savoir quelle clause il doit vérifier ; le seul moyen que $\theta$ a de le faire est donc de choisir une valuation des variables qui satisfait toutes les clauses.

On a donc bien que $\mathcal{P}_\phi$ est observable par un observateur à une horloge si et seulement si $\phi$ est satisfaisable. On en déduit la proposition suivante.

**Proposition 4.8** *L'observation à ressources bornées est NP-difficile*

### 4.2.2 Preuve d'EXPTIME-difficulté

On va dans un premier temps montrer que l'observation à ressources bornées est EXPTIME-difficile avant de montrer qu'elle est 2-EXPTIME-difficile, ce qui nous permettra de présenter la méthode de réduction d'une machine alternante à l'observation à ressources bornées dans un cas plus simple sans avoir les difficultés techniques de la preuve de 2-EXPTIME–difficulté.

On part donc d'une machine de Turing alternante d'espace $n^k$ et de $w$ de longueur $n$, et on va construire $\mathcal{P}_{M,w}$ de taille polynomiale en $n$ tel que $M$ accepte $w$ si et seulement si $\mathcal{P}_{M,w}$ est observable par un observateur à une horloge. Pour plus de lisibilité, $\mathcal{P}_{M,w}$ sera simplement noté $\mathcal{P}$.

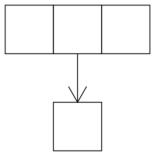
Remarque : on fait les hypothèses suivantes sur $M$ :
$M$ a au plus 2 successeurs. $M$ boucle sur son état final $q_{yes}$ et sur son état non accepteur $q_{no}$.
En s'inspirant de [CKS81], on peut de plus supposer que $M$ n'a pas de boucle non triviale ; plus précisément, les seules boucles de $M$ sont sur $q_{yes}$ et $q_{no}$ et sont des boucles de période 1.
Pour cela, $M$ garde un compteur du nombre de pas effectués et si celui-ci dépasse son nombre de configurations distinctes, elle va dans $q_{no}$.
En particulier après un nombre de pas supérieur au nombre de configurations de $M$, $M$ est en train de boucler ; soit sur $q_{yes}$ si elle accepte $w$, soit sur $q_{no}$ si elle rejette $w$.

Comme dans le cas d'une formule logique, l'observateur $\theta$ va faire des choix (en choisissant de stocker la première ou la deuxième horloge) et $\mathcal{P}$ va vérifier que les choix de $\theta$ sont corrects.
Ici les choix de $\theta$ ne vont plus correspondre à la valeur de vérité d'une variable mais à une lettre. $\theta$ va donc choisir une suite de lettres formant un mot, ce mot correspondant à une suite de configurations acceptantes de $M$ sur $w$ de type $C_1\#C_2\#\cdots\#C_k$ où les $C_i$ sont des configuration de $M$ (avec le codage classique) et $C_k$ est une configuration finale.

Le rôle de $\mathcal{P}$ va être de vérifier que $\theta$ choisit bien une suite de configurations acceptante, par exemple $\mathcal{P}$ doit empêcher $\theta$ de choisir une suite de configurations $C_1\#C_2\#\cdots\#C_k$ où $C_2$ n'est pas un successeur de $C_1$ dans $M$.

L'idée de la preuve va être de coder par des formules logiques le fait que $C_1\#C_2\#\cdots\#C_k$ soit une suite de configurations correcte ; ensuite $\mathcal{P}$ empêchera $\theta$ de choisir des suites de configurations incorrectes par la même technique que celle utilisée dans la partie précédente où par exemple $\mathcal{P}$ pouvait empêcher $\theta$ de choisir des valeurs pour $x_1, x_2, x_3$ telles que $x_1 \vee x_2 \vee x_3$ soit faux (c'est à dire que si $\theta$ ne fait pas les bons choix il ne sera pas un observateur). On aura donc «$\theta$ fait les bons choix» $\Longleftrightarrow$ «$C_1\#C_2\#\cdots\#C_k$ jouée par $\theta$ est une suite de configurations acceptantes» ; d'où «il existe un observateur à une horloge $\theta$ » $\Longleftrightarrow$ «il existe une suite de configurations acceptantes» $\Longleftrightarrow$ «$M$ accepte $w$».

On rappelle qu'il existe une formule de 3-SAT (à 4 variables) qui est vraie si et seulement si le motif suivant respecte les transitions de $M$ (voir [Pap94] page 172) :



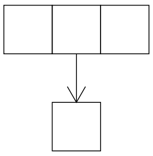
Les trois cases en haut représentent les cases $i-1$, $i$, et $i+1$ du ruban de la configuration précédente de $M$ et la case en dessous représente la $i^{eme}$ case du ruban de la configuration actuelle de $M$ ; on peut en effet déduire le contenu de cette case uniquement à partir des trois cases du haut et de la table de transition de $M$.

Le comportement de $\mathcal{P}$ est le suivant : $\mathcal{P}$ va demander à $\theta$ de faire ses choix (en proposant à $\theta$ de choisir entre retenir la première ou la deuxième horloge et en recommençant), puis de façon non déterministe et sans que $\theta$ puisse savoir quand (c'est à dire de façon non observable) va se mettre en phase de test et vérifier l'un des points ci-dessous. $\theta$ sera donc obligé de choisir une suite de configurations correcte car même si $\mathcal{P}$ ne peut faire qu'un test, le fait qu'il puisse le faire «n'importe quand» interdit à $\theta$ de tricher.
Voici les points vérifiés par $\mathcal{P}$ :

- Comment $\mathcal{P}$ fait-il pour tester que la première configuration choisie par $\theta$ est bien la configuration initiale de $M$ ?
  La formule logique correspondante est $\forall\ 1 \leq i \leq n^k\ (choix)_i = w_i$
  $\mathcal{P}$ va vérifier de façon non déterministe et non observable que $(choix)_i = w_i$ par le même gadget utilisé pour imposer qu'une variable logique soit choisie vraie (cf partie précédente) ; comme $\theta$ ne saura pas quel $i$ va

être vérifié, le seul moyen qu'il a d'être un observateur valide est d'avoir «$\forall\ 1 \leq i \leq n^k\ choix_i = w_i$».

- Comment $\mathcal{P}$ fait pour vérifier que $C_{i+1}$ est un sucesseur de $C_i$ ?
  La formule logique correspondante est $\forall$ *position* de $C_i$ le motif



  est respecté.
  $\mathcal{P}$ vérifie de façon non déterministe et non observable que le motif est bien respecté. Comme $\theta$ ne peut savoir sur quel motif il va être interrogé, il doit tous les respecter.
  Pour vérifier que le motif est bien respecté, $\mathcal{P}$ fait les branchements correspondant aux choix de $\theta$ qu'il doit tester ; mais comme le dernier choix peut-être assez loin (la case du bas du motif sera annoncée $n^k$ étapes après celles du dessus), $\mathcal{P}$ a besoin d'un compteur. Ici cela ne pose pas de problème car les configurations sont de taille polynomiale.
- Comment $\mathcal{P}$ fait pour prendre en compte l'alternance ?
  Au début de chaque configuration (après le $\#$), $\mathcal{P}$ choisit de façon non déterministe de faire un «1» ou un «2», qui indique lequel des 2 successeurs de $M$ doit être pris si l'état courant est de type universel.
  Par exemple si $\mathcal{P}$ a fait un «1», que l'état courant est de type universel et que $\theta$ choisit la configuration suivante selon le deuxième successeur, $\mathcal{P}$ doit imposer que $\theta$ ne soit pas un observateur correct.
  Pour cela, on modifie la formule utilisée pour vérifier que $C_{i+1}$ est un sucesseur de $C_i$ de la manière suivante : on note $c$ le choix fait par $\mathcal{P}$ ($c \in \{1, 2\}$), la formule utilisée sera

  $\{(q$ de type universel$)\wedge(c = 1) \Rightarrow$ respecter le motif selon le premier sucesseur$\}\ \bigwedge$

  $\{(q$ de type universel$)\wedge(c = 2) \Rightarrow$ respecter le motif selon le deuxième sucesseur$\}$
- Comment fait $\mathcal{P}$ pour vérifier que la suite de configurations choisie par $\theta$ est acceptante ?
  En fait $\mathcal{P}$ va s'assurer que $q_{no}$ ne peut être atteint, si c'est le cas c'est que $M$ accepte $w$ (dans le cas contraire, $M$ finit par aller dans $q_{no}$).
  Pour vérifier que $q_{no}$, $\mathcal{P}$ force $\theta$ à ne jamais annoncer une configuration contenant cet état par le même gadget utilisé pour imposer qu'une variable logique soit choisie vraie.

### 4.2.3 Preuve de 2-EXPTIME-difficulté

On utilise la même méthode que pour la preuve d'EXPTIME-difficulté : on réduit l'acceptation d'une machine de Turing alternante à l'observation à ressources bornées.

On part donc d'une machine de Turing alternante d'espace $2^n$ et de $w$ de longueur $n$, et on va construire $\mathcal{P}_{M,w}$ de taille polynomiale en $n$ tel que $M$ accepte $w$ si et seulement si $\mathcal{P}_{M,w}$ est observable par un observateur à une horloge.

On utilise la même réduction, $\theta$ doit choisir une suite de configurations acceptante de $M$ sur $w$ et $\mathcal{P}$ doit vérifier que $\theta$ fait des choix valides.

La méthode employée par $\mathcal{P}$ pour vérifier que la suite de choix est valide est la même que précédemment, la seule difficulté supplémentaire est que les configurations de $M$ sont plus longues puisqu'elles sont maintenant de taille exponentielle.

Pour tester que le motif de la machine de Turing était respecté, $\mathcal{P}$ devait faire un test sur les trois premiers caractères du motifs, puis attendre le caractère suivant qui était à distance $n^k$ ; pour cela $\mathcal{P}$ avait un compteur interne de 1 à $n^k$ ; par exemple un compteur de type automate fini stocké dans les états.
Maintenant $\mathcal{P}$ doit pouvoir compter jusqu'à $2^n$, on ne peut donc plus utiliser un tel compteur car sinon $\mathcal{P}$ sera de taille exponentielle. Une solution simple, si on considère que les constantes dans le système sont codées en binaire, consiste à faire que chaque annonce de caractère de la configuration dure une unité de temps et à rajouter une horloge $z$ qui sera remise à zèro lors de la lecture du premier caractère. Pour tester le caractère suivant situé à distance $2^n$, $\mathcal{P}$ fera le test $z = 2^n$ et saura qu'il est arrivé au bon caractère. Si on admet que les constantes dans les automates temporisés sont codées en binaire, la preuve est alors finie.

Si on impose que ces constantes soient codées en unaire, la preuve fonctionne également, mais avec une construction plus technique adaptée de [DM02] que nous exposons ci-dessous.

A la place de $z$, $\mathcal{P}$ va utiliser $n+1$ horloges $c_0, \cdots, c_n$ modélisant un compteur binaire, $\mathcal{P}$ pourra alors compter jusqu'à $2^n$, sa taille restant polynomiale même si ses constantes sont codées de façon unaire.
$< c_0, \cdots, c_n >$ représentent un compteur de la manière suivante : au début de chaque phase d'annonce de caractère, $\mathcal{P}$ génère un événement annonçant le début de cette phase (par exemple \$). Au moment où cet événement spécial est annoncé, la valeur du compteur est la valeur du nombre représenté en binaire par $< b(c_n), \cdots, b(c_0) >$ où $b(c_i) = 1$ si $c_i > 1$ et $b(c_i) = 0$ si $c_i \leq 1$.

Tester si le compteur a atteint $2^n$ est facile, il suffit de tester $c_n > 1$ au moment du \$ ; la difficulté est d'incrémenter le compteur à la fin de chaque phase. Pour cela, on rajoute $n+1$ actions «inutiles» et non-observables étiquetées $d$ générées par $\mathcal{P}$ à la fin de chaque phase. Ces actions $d$ servent uniquement à propager la retenue dans les $c_i$.

On appelle *d-bloc* cette suite de $n+1$ $d$. Chaque phase durant une unité de temps, $\mathcal{P}$ peut s'assurer que le $j^{\text{ème}}$ $d$ d'un *d-bloc* a lieu exactement une unité de temps après le $j^{\text{ème}}$ $d$ du *d-bloc* précédent. La $j^{\text{ème}}$ occurence de $d$ va servir à mettre à jour la position de $c_j$ par rapport à 1 lors de l'incrémentation. A la fin du *d-bloc*, le compteur aura été incrémenté.

Supposons que lors du \$, la valeur du compteur soit $k$. Notons qu'alors $c_j$ représente le bit 1 si et seulement si $c_j > 1$. Nous allons lire au fur et à mesure les $d$ et mettre à jour les $c_j$ en faisant les remises à zéro appropriées et en stockant la retenue dans un état de $\mathcal{P}$. Lors de la lecture du $j^{\text{ème}}$ $d$, nous connaissons la retenue et savons que le bit représenté est 1 si $c_j > 1$ et 0 si $c_j \leq 1$ (car la dernière fois que $c_j$ a éventuellement été remis à zéro c'était lors du $j^{\text{ème}}$ $d$ du *d-bloc* précédent il y a une unité de temps). Nous pouvons maintenant mettre le bit représenté par $c_j$ à zéro en remettant $c_j$ à zéro (alors lors du prochain \$, nous aurons que $c_j \leq 1$). Nous pouvons mettre le bit représenté par $c_j$ à 1 en ne remettant pas $c_j$ à zéro (alors lors du prochain \$, $c_j$ n'aura pas été du tout remis à zéro lors de cette phase qui dure une unité de temps et donc nous aurons $c_j > 1$.

A la fin du *d-bloc*, le compteur a bien été incrémenté. Remarquons que les horloges $c_i$ ne sont testées que par rapport à la valeur 1. Les constantes utilisées dans $\mathcal{P}$ sont donc de taille polynomiale.

Nous avons donc montré :

**Théorème 4** *Le problème 4 est 2-EXPTIME-complet.*

Ce résultat établit la complexité de l'observation par automates déterministes. Il est assez surprenant que la complexité soit si élevée. En effet, l'observation temporisée peut être vue comme un cas particulier a priori plus simple du contrôle temporisé ; or il est prouvé dans [BDMP03] que le contrôle par des automates temporisés déterministes est 2-EXPTIME-complet. Il semblait alors raisonnable de supposer que la complexité soit moindre pour l'observation ; le théorème 4 montre que ce n'est pas le cas et que pour la classe des automates temporisés déterministes le contrôle et l'observation ont la même complexité.

# Chapitre 5

# Observation par des automates *event-recording*

Dans le chapitre précédent, nous avons étudié l'existence d'un observateur automate temporisé déterministe. Nous nous intéressons maintenant à l'observation par des automates plus simples, les automates *event-recording*, en espérant obtenir une meilleure complexité. Comme pour les automates déterministes, nous n'avons pas réussi à résoudre le problème dans le cas général (c'est à dire demander l'existence d'un observateur *event-recording* sans spécifier de condition supplémentaire). Mais dans ce cas également, nous avons trouvé un algorithme pour décider l'existence d'un observateur de ressources fixées.

Alors que le problème de l'existence d'un observateur était 2-EXPTIME-complet, nous verrons que pour les *event-recordings* il est PSPACE-complet. La classe des *event-recordings* apparaît donc comme une classe intéressante d'observateurs. [1]

## 5.1 Cadre de travail

Nous rappelons la définition de $\Delta$-observabilité dans le cadre des *event-recording* :

**Definition 5.1.1** *Soit* $\mu = (\frac{1}{m}, M)$ *une granularité,* $\mathcal{P}$ *un sytème et* $\Delta \in \mathbb{N}$. *On dit que* $\mathcal{P}$ *est* $\Delta$-observable event-recordings de granularité $\mu$ *s'il existe* $\theta$ event-recording *de granularité* $\mu$ *tel que tel que*

$$\pi(L_{\Delta f}) \subseteq L(\theta) \subseteq \pi(L_{\neg f})^c$$

[1] La complexité semble même optimale puisque sauf pour des classes pathologiques d'observateurs (où le vide n'est pas reconnaissable), le problème de l'observation est toujours PSPACE-difficile.

**Problème 5** *Etant donnés une granularité $\mu$, un sytème $\mathcal{P}$ et un entier $\Delta$, $\mathcal{P}$ est-il $\Delta$-observable par* event-recordings *de granularité $\mu$ ?*

Dans un premier temps, remarquons que tout système $\Delta$-observable par *event-recordings* l'est par automates temporisés déterministes. La réciproque n'est pas vraie comme le montre la proposition suivante, ce qui justifie que nous nous intéressions à l'observation par *event-recordings.*

**Proposition 5.1** *Pour tout $\Delta > 0$, le système suivant est $\Delta$-observable par automates temporisés déterministes, mais pas $\Delta$-observable par* event-recording*, $\forall \Delta > 0$ :*

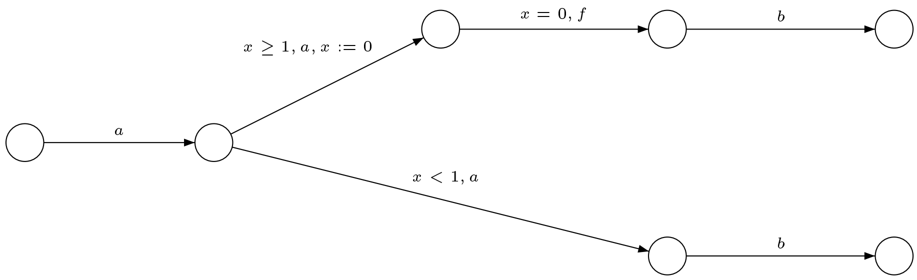



FIG. 18 – Système $\mathcal{P}$ observable par automates temporisés déterministes, mais pas par *event-recordings*

**Preuve :**
La preuve est la même que celle de la proposition 3.3 page 26 en utilisant le fait que le langage $\{t_1, a, t_2, a, t_3, b \mid t_1 + t_2 \geq 1 \text{ et } t_3 \geq \Delta\}$ n'est pas reconnaissable par *event-recordings.*

## 5.2 Résolution du problème en PSPACE

Nous allons montrer que le problème 5 peut être résolu en PSPACE et que la preuve est effective : s'il existe un observateur, il sera constructible.

On pose dans cette partie $\mu = (\frac{1}{m}, M)$ une granularité, $\Sigma_o$ et $\Sigma_u$ les alphabets respectifs des actions observables et non observables, et $\Gamma$ l'alphabet symbolique *event-recording* sur $\mu$ et $\Sigma_o$. Rappelons que la projection sur $\Sigma_o$ est notée $\pi$.
Comme dans le chapitre précédent, $\mathcal{U}_\Gamma$ est l'automate temporisé universel sur $\Gamma$, il a un unique état et reconnaît tous les mots symboliques de $\Gamma^*$.

Remarquons tout d'abord une propriété élémentaire de l'alphabet symbolique event-recording :

**Lemme 5.2** *Soit $w$ un mot temporisé sur $\Sigma_o$, il existe un unique $\gamma \in \Gamma^*$ tel que $w \in tw(\gamma)$. On le note $symb(w)$.*

Dans un premier temps on définit $S_{\Delta f}$ qui sera l'observateur *event-recording* s'il en existe un et on donne une propriété qui lie son langage symbolique à $\pi(L_{\Delta f})$. Intuitivement, $S_{\Delta f}$ reconnaît l'ensemble des comportements $\Delta$ fautifs du système $\mathcal{P}$ «projetés sur $\Gamma$». Cette projection $\upharpoonright \Gamma$ correspond à la double perte d'information due aux actions non observables et au fait que seule la granularité $\mu$ est visible.

**Proposition 5.3** *Soit $\mathcal{P}$ un système et soit $S_{\Delta f} = \mathcal{R}(\mathcal{P}_{\Delta f}||\mathcal{U}_\Gamma) \upharpoonright \Gamma$, alors $L_{symb}(S_{\Delta f}) = symb(\pi(L_{\Delta f}))$*

**Preuve :**
Les propriétés du graphe des régions assurent que :
$L(\mathcal{R}(\mathcal{P}_{\Delta f}||\mathcal{U}_\Gamma)) = L(\mathcal{P}_{\Delta f}) = L_{\Delta f}$ et $\forall \gamma_0 \in L_{symb}(\mathcal{R}(\mathcal{P}_{\Delta f}||\mathcal{U}_\Gamma))$ on a $tw(\gamma_0) \neq \emptyset$.

Montrons que $L_{symb}(S_{\Delta f}) \subseteq symb(\pi(L_{\Delta f}))$.
Soit $\gamma \in L_{symb}(S_{\Delta f})$, montrons que $\gamma \in symb(\pi(L_{\Delta f}))$.
Il existe $\gamma_0$ dans $L_{symb}(\mathcal{R}(\mathcal{P}_{\Delta f}||\mathcal{U}_\Gamma))$ tel que $\gamma = \gamma_0 \upharpoonright \Gamma$
$tw(\gamma_0) \neq \emptyset$, soit donc $w \in tw(\gamma_0)$, $w \in L(\mathcal{P}_{\Delta f}) = L_{\Delta f}$
Comme $w \in tw(\gamma_0)$, $\pi(w) \in tw(\gamma)$ donc $\gamma = symb(\pi(w))$.
De $\gamma = symb(\pi(w))$ et $w \in L_{\Delta f}$ il vient $\gamma \in symb(\pi(L_{\Delta f}))$.

Montrons que $symb(\pi(L_{\Delta f})) \subseteq L_{symb}(S_{\Delta f})$.
Soit $\gamma \in symb(\pi(L_{\Delta f}))$, montrons que $\gamma \in L_{symb}(S_{\Delta f})$.
Il existe $w \in L_{\Delta f}$ tel que $\gamma = symb(\pi(w))$.
Comme $L(\mathcal{R}(\mathcal{P}_{\Delta f}||\mathcal{U}_\Gamma) = L_{\Delta f}$, $w$ est accepté par $\mathcal{R}(\mathcal{P}_{\Delta f}||\mathcal{U}_\Gamma)$ ; il existe donc $\gamma_0 \in L_{symb}(\mathcal{R}(\mathcal{P}_{\Delta f}||\mathcal{U}_\Gamma))$ tel que $w \in tw(\gamma_0)$.
Montrons que $\gamma_0 \upharpoonright \Gamma = \gamma$.
En effet, $w \in tw(\gamma_0) \Rightarrow \pi(w) \in tw(\gamma_0 \upharpoonright \Gamma) \Rightarrow \gamma_0 \upharpoonright \Gamma = symb(\pi(w)) = \gamma$.
De $\gamma_0 \in L_{symb}(\mathcal{R}(\mathcal{P}_{\Delta f}||\mathcal{U}_\Gamma))$ et $\gamma = \gamma_0 \upharpoonright \Gamma$, on déduit
$\gamma \in L_{symb}(\mathcal{R}(\mathcal{P}_{\Delta f}||\mathcal{U}_\Gamma) \upharpoonright \Gamma) = L_{symb}(S_{\Delta f})$.
$\square$

Par une preuve analogue on montre :

**Proposition 5.4** *Soit $S_{\neg f} = \mathcal{R}(\mathcal{P}_{\neg f}||\mathcal{U}_\Gamma) \upharpoonright \Gamma$, alors $L_{symb}(S_{\neg f}) = symb(\pi(L_{\neg f}))$*

On donne maintenant une caractérisation de l'observabilité par *event-recording*. Intuitivement l'opérateur *symb* correspond au filtre imposé par le fait que l'on observe sous la granularité $\mu$ ; $symb(\pi(L_{\Delta f}))$ est l'ensemble des observations pouvant résulter d'un comportement $\Delta$-fautif quand on regarde sous une granularité $\mu$, et $symb(\pi(L_{\neg f}))$ les observations pouvant résulter d'un comportement non fautif. La condition $symb(\pi(L_{\Delta f}))\cap symb(\pi(L_{\neg f})) = \emptyset$ apparaît donc comme nécessaire pour l'observation par *event-recording* de granularité $\mu$ : si elle n'est pas vérifiée, il existe une observation pouvant résulter à la fois d'un comportement $\Delta$-fautif et d'un comportement non fautif. Ce qui est plus étonnant c'est que cette condition soit suffisante ; c'est dû au fait que $S_{\Delta f}$ (qui est un *event-recording* de granularité $\mu$) capture dans leur intégralité les observations résultant de comportements $\Delta$-fautifs.

**Proposition 5.5** *Les conditions suivantes sont équivalentes :*

*(i)* $\mathcal{P}$ *est* $\Delta$*-observable par* event-recordings *de granularité* $\mu$

*(ii)* $symb(\pi(L_{\Delta f})) \cap symb(\pi(L_{\neg f})) = \emptyset$

*(iii)* $L_{symb}(S_{\Delta f}) \cap L_{symb}(S_{\neg f}) = \emptyset$

**Preuve :**
Le fait que $(ii) \Leftrightarrow (iii)$ est évident à partir des propositions 5.3 et 5.4 ; en effet $L_{symb}(S_{\Delta f}) = symb(\pi(L_{\Delta f})$ et $L_{symb}(S_{\neg f}) = symb(\pi(L_{\neg f}))$.

Montrons que $(i) \Rightarrow (ii)$.
Supposons que $\mathcal{P}$ est $\Delta$-observable par *event-recordings* de granularité $\mu$ et montrons que $symb(\pi(L_{\Delta f})) \cap symb(\pi(L_{\neg f})) = \emptyset$.
Soit $\theta$ un $\Delta$-observateur *event-recording* de granularité $\mu$ de $\mathcal{P}$. On a $\pi(L_{\Delta f}) \subseteq L(\theta) \subseteq \pi(L_{\neg f})$.
On va montrer que $symb(\pi(L_{\Delta f})) \subseteq L_{symb}(\theta)$ et que $L_{symb}(\theta) \cap symb(\pi(L_{\neg f})) = \emptyset$, d'où on déduira immédiatemment que $symb(\pi(L_{\Delta f})) \cap symb(\pi(L_{\neg f})) = \emptyset$.
Pour la première inclusion : comme $\theta$ est de granularité $\mu$, on a que $symb(L(\theta)) = L_{symb}(\theta)$. D'où $symb(\pi(L_{\Delta f})) \subseteq L_{symb}(\theta)$.
Montrons maintenant que $L_{symb}(\theta) \cap symb(\pi(L_{\neg f})) = \emptyset$. En effet, dans le cas contraire, soit $\gamma \in L_{symb}(\theta) \cap symb(\pi(L_{\neg f}))$ et $w \in \pi(L_{\neg f})$ tel que $tw(w) = \gamma$. Alors $w \in L(\theta) \cap \pi(L_{\neg f}) = \emptyset$, ce qui est absurde.

Montrons que $(ii) \Rightarrow (i)$.
Supposons que $symb(\pi(L_{\Delta f})) \cap symb(\pi(L_{\neg f})) = \emptyset$ et cherchons $\theta$ *event - recording* de granularité $\mu$ tel que $\pi(L_{\Delta f}) \subseteq L(\theta) \subseteq \pi(L_{\neg f})^c$.
Nous prenons comme candidat $\theta = S_{\Delta f}$. Remarquons tout d'abord que $S_{\Delta f}$ est un *event-recording* de granularité $\mu$ ; en effet $S_{\Delta f}$ a certes des $\varepsilon$-transitions, mais il n'y a pas de gardes ni de remise à zéro sur ces transitions ; on peut donc les enlever de façon symbolique en considérant $S_{\Delta f}$ comme un automate fini.
Montrons que $\pi(L_{\Delta f}) \subseteq L(S_{\Delta f})$ : en effet $\pi(L_{\Delta f}) \subseteq tw(symb(\pi(L_{\Delta f}))) = tw(L_{symb}(S_{\Delta f})) = L(S_{\Delta f})$. Remarquons que pour obtenir l'avant dernière égalité, on a utilisé la proposition 5.3.
Montrons que $L(S_{\Delta f}) \cap \pi(L_{\neg f}) = \emptyset$ en raisonnant par l'absurde.

Soit $w \in L(S_{\Delta f}) \cap \pi(L_{\neg f})$ et $\gamma = symb(w)$. Comme $S_{\Delta f}$ est un *event-recording* de granularité $\mu$,
$\gamma \in L_{symb}(S_{\Delta f}) \cap symb(\pi(L_{\neg f})) = symb(\pi(L_{\Delta f})) \cap symb(\pi(L_{\neg f})) = \emptyset$, ce qui est absurde. □

De la preuve on peut également déduire :

**Corollaire 5.6** *Soit $\mathcal{P}$ un système, si $\mathcal{P}$ est $\Delta$-observable, alors $S_{\Delta f}$ est un $\Delta$-observateur de $\mathcal{P}$.*

La proposition 5.5 permet de déduire un algorithme pour tester la $\Delta$-observabilité par *event-recordings* :

**Proposition 5.7** *Le problème 5 est dans PSPACE.*

**Preuve :**
On rappelle le problème 5 :
Etant donné une granularité $\mu$, un sytème $\mathcal{P}$ et un entier $\Delta$, $\mathcal{P}$ est-il $\Delta$-observable par *event-recordings* de granularité $\mu$ ?

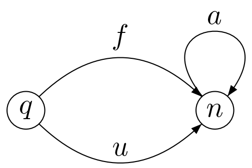
Par la proposition 5.5, il suffit de tester si $L_{symb}(S_{\Delta f}) \cap L_{symb}(S_{\neg f}) = \emptyset$. On montre que ce test est faisable en co-NPSPACE=PSPACE. On cherche donc un algorithme non déterministe d'espace polynomial pour tester si $L_{symb}(S_{\Delta f}) \cap L_{symb}(S_{\neg f}) = \emptyset$. Il s'agit en fait de tester le vide sur l'automate fini avec $\varepsilon$-transitions $S_{\Delta f} \times S_{\neg f}$ ; le problème du vide étant dans NLOG et $S_{\Delta f} \times S_{\neg f}$ de taille exponentielle, on obtient un algorihme non déterministe d'espace polynomial. □

On a donc résolu le problème d'existence d'un $\Delta$-observateur *event-recording* de granularité $\mu$. De plus si un observateur existe, on peut en construire un (par exemple $S_{\Delta f}$). Notons que $S_{\Delta f}$ est a priori de taille exponentielle, mais peut être non-déterministe ; en déterminisant $S_{\Delta f}$, on obtient un observateur déterministe de taille doublement exponentielle.

On va maintenant donner la complexité du problème 5 :

**Théorème 5** *Le problème 5 est PSPACE-complet.*

**Preuve :**
Par la proposition 5.7, le problème 5 est dans PSPACE ; montrons qu'il est également PSPACE-dur.
On réduit l'accessibilité dans les automates temporisés au problème 5.
Soit $A$ un automate temporisé sur l'alphabet $\Sigma$ et $q$ état de $A$. On considère comme événements observables $\Sigma_o = \Sigma$ et comme événements non observables $\Sigma_u = \{f, u\}$ et on construit le système $\mathcal{P}$ ainsi : on rajoute un nouvel état $n$ à $A$ et les deux transitions suivantes (où $a \in \Sigma_o$) :




Alors on a : $q$ est accessible dans $A \iff \mathcal{P}$ n'est pas 1-observable.
En effet, si $q$ est accessible dans $A$ par le mot $w$, le mot $w, 1, a$ est dans $\pi(L_{\Delta f}) \cap \pi(L_{\neg f})$ ; $\mathcal{P}$ n'est donc pas observable dans le cas général donc à fortiori par *event-recordings* de granularité $\mu$.
Réciproquement, si $a$ n'est pas accessible dans $A$, alors aucune faute ne se produit jamais et donc $\mathcal{P}$ est observable par un automate reconnaissant le langage vide.
On a donc montré que le problème 5 est co-PSPACE-dur donc PSPACE-dur.
Remarque : à la place du délai 1, on aurait pu prendre tout délai $\Delta \in \mathbb{N}^*$. □

## 5.3 Existence d'un observateur optimal

Dans la partie précédente, nous avons montré comment résoudre le problème de l'existence d'un observateur *event -recording* de granularité $\mu$. Nous avons prouvé que si la réponse était positive, alors $S_{\Delta f}$ était un tel observateur. Nous allons maintenant montrer un résultat plus fin : $S_{\Delta f}$ est en fait le meilleur observateur *event -recording* de granularité $\mu$ pour le système, dans le sens où il annonce le moins souvent des erreurs parmi tous les observateurs *event-recording* de granularité $\mu$ qui annoncent toutes les exécutions $\Delta$-fautives.

**Proposition 5.8** *Soit $\mathcal{P}$ un système $\Delta$-observable par les* event-recordings *de granularité $\mu$. Soit $\theta = S_{\Delta f}$ et $\theta'$ un $\Delta$-observateur* event-recording *de $\mathcal{P}$ de granularité $\mu$. Alors $L(\theta) \subseteq L(\theta')$.*

**Preuve :**
Sous les hypothèse de l'énoncé, montrons que $L(\theta) \subseteq L(\theta')$.
Soit $w \in L(S_{\Delta f})$, montrons que $w \in L(\theta')$.
Soit $\gamma = symb(w)$, par la proposition 5.3, $L_{symb}(S_{\Delta f}) = symb(\pi(L_{\Delta f}))$, donc $\gamma \in symb(\pi(L_{\Delta f}))$. Il existe donc $w' \in \pi(L_{\Delta f})$ tel que $w' \in tw(\gamma)$ (en particulier $symb(w') = \gamma$).
Comme $\theta'$ est un observateur de $\mathcal{P}$, $\pi(L_{\Delta f}) \subseteq L(\theta')$ donc $w' \in L(\theta')$. Soit $\gamma'$ le mot symbolique correspondant à $w'$ dans $\theta'$ ; on a $\gamma' \in L_{symb}(\theta')$ et $w' \in \gamma'$. Mais en fait $\gamma' = symb(w') = \gamma$, donc $\gamma \in L_{symb}(\theta')$ d'où $w \in L(\theta')$.
□

*Remarque :*
De manière analogue, on obtient que $S_{\neg f}$ est l'observateur de $\mathcal{P}$ ayant « le

plus grand langage » parmis tous les observateurs event-recording de granularité $\mu$ de $\mathcal{P}$.

## 5.4 Existence d'un délai d'observation

Nous avons traité le problème suivant : en fixant les ressources, étant donné un sytème $\mathcal{P}$ et un délai $\Delta$, $\mathcal{P}$ est-il $\Delta$-observable ? Une autre question peut-être plus naturelle se posant pour un système est de savoir s'il existe un délai sous lequel le système peut être observé, et si oui calculer le plus petit délai possible.

**Problème 6** *Etant donné un sytème $\mathcal{P}$ et une granularité $\mu$, existe-t-il un $\Delta$ tel que $\mathcal{P}$ est $\Delta$-observable par un* event-recording *de granularité $\mu$ ?*

Remarque : la résolution de ce problème permet de déduire un algorithme pour trouver le $\Delta$ minimum tel que $\mathcal{P}$ est $\Delta$-observable : dans le cas où il existe un $\Delta$ sous lequel $\mathcal{P}$ est $\Delta$-observable, en se servant des résultats de la partie 5.2 qui nous permettent de décider si un $\Delta$ donné fonctionne, il suffit de faire un recherche dichotomique pour trouver le $\Delta$ minimum convenant (car nous avons supposé $\Delta$ entier).

Pour résoudre le problème 6, on va se servir de l'automate $S_{\neg f}$ déjà utilisé dans la proposition 5.5. On va caractériser la non-$\Delta$-observabilité par l'existence d'une exécution $\Delta$ fautive acceptante dans $\mathcal{P}||_{\Sigma_o}S_{\neg f}$. Intuitivement $S_{\neg f}$ caractérise les observations pouvant provenir d'une exécution non fautive ; une exécution $\Delta$ fautive acceptante dans $\mathcal{P}||_{\Sigma_o}S_{\neg f}$ nous donnerait une observation pouvant provenir à la fois d'une exécution $\Delta$-fautive (par sa composante dans $\mathcal{P}$) et d'une exécution non fautive (par sa composante dans $S_{\neg f}$).

La méthode semble analogue à celle employée dans la partie 5.2, mais dans cette dernière partie les automates construits (dont on testait la disjonction des langages) dépendaient du délai $\Delta$, il semblait alors difficile de répondre à une question du type « Existe-t-il un $\Delta$ tel que ... » puisqu'il aurait fallu construire des automates pour chaque $\Delta$.
Au contraire, ici l'automate $\mathcal{P}||_{\Sigma_o}S_{\neg f}$ ne dépend pas de $\Delta$ et pourtant la non-$\Delta$-observabilité est équivalente à l'existence d'une exécution $\Delta$ fautive acceptante dans cet automate. Pour décider l'existence d'un $\Delta$ sous lequel le sytème est observable, il suffit donc de tester si, pour tout $\Delta$ il existe une exécution $\Delta$ fautive acceptante dans $\mathcal{P}||_{\Sigma_o}S_{\neg f}$, ce qui, nous le verrons, est algorithmiquement faisable.

**Lemme 5.9** *Soit $w$ mot temporisé sur $\Sigma_o \cup \Sigma_u$. Alors*

$$\pi(w) \in L(S_{\neg f}) \iff symb(\pi(w)) \in symb(\pi(L_{\neg f})$$

**Preuve :**
Supposons que $\pi(w) \in L(S_{\neg f})$ et montrons que $symb(\pi(w)) \in symb(\pi(L_{\neg f})$. Comme $S_{\neg f}$ est un event-recording sur $\Gamma$, $\pi(w)$ ne peut être reconnu dans $S_{\neg f}$ que par le mot symbolique $symb(\pi(w))$. Donc $symb(\pi(w)) \in L_{symb}(S_{\neg f}) = symb(\pi(L_{\neg f}))$.

Supposons que $symb(\pi(w)) \in symb(\pi(L_{\neg f}))$ et montrons que $\pi(w) \in L(S_{\neg f})$.
Comme $L_{symb}(S_{\neg f}) = symb(\pi(L_{\neg f}))$, $symb(\pi(w)) \in L_{symb}(S_{\neg f})$ ; or $\pi(w) \in tw(symb(\pi(w)))$, donc $\pi(w) \in tw(L_{symb}(S_{\neg f})) = L(S_{\neg f})$. □

**Proposition 5.10** *$\mathcal{P}$ n'est pas $\Delta$-observable par les* event -recordings *de granularité $\mu$ si et seulement si $\mathcal{P}||_{\Sigma_o}S_{\neg f}$ a une exécution $\Delta$-fautive acceptante.*

**Preuve :**
$\mathcal{P}$ n'est pas $\Delta$-observable par les *event-recordings* de granularité $\mu$
$\iff symb(\pi(L_{\Delta f})) \cap symb(\pi(L_{\neg f})) \neq \emptyset$
$\iff \exists w$ exécution de $\mathcal{P}$ $\Delta$-fautive telle que $symb(\pi(w)) \in symb(\pi(L_{\neg f}))$
$\iff \exists w$ exécution de $\mathcal{P}$ $\Delta$-fautive telle que $\pi(w) \in L(S_{\neg f})$
$\iff \mathcal{P}||_{\Sigma_o}S_{\neg f}$ a une exécution $\Delta$-fautive acceptante

Remarques : la première équivalence résulte de la proposition 5.5, la troisième résulte du lemme 5.9. □

**Corollaire 5.11** *$\forall\Delta$, $\mathcal{P}$ n'est pas $\Delta$-observable par les* event-recordings *de granularité $\mu$ si et seulement si $\forall\Delta$ $\mathcal{P}||_{\Sigma_o}S_{\neg f}$ a une exécution $\Delta$ fautive acceptante.*

**Théorème 6** *Le problème 6 est PSPACE-complet.*

**Preuve :**
En utilisant la même réduction que celle de la preuve du théorème 5, on montre que le problème 6 est PSPACE-dur.

Montrons que le problème 6 est dans PSPACE : il suffit de montrer que les hypothèses du corrolaire 5.11 sont testables en PSPACE. On note $\mathcal{R}$ le graphe des régions de $\mathcal{P}||S_{\neg f}$ et $\mathcal{R}_f$ l'ensemble des régions de $\mathcal{R}$ accessibles par un chemin fautif.
On dit qu'une composante fortement connexe $\Lambda$ de $\mathcal{R}$ est de temps non borné si et seulement si pour toute horloge $x$ de $\mathcal{P}||_{\Sigma_o}S_{\neg f}$, soit $x$ est remise à zéro et peut croître au delà de zéro dans $\Lambda$, soit $x$ reste non bornée dans $\Lambda$. La technique employée dans [AD94] permet de montrer qu'une composante fortement connexe $\Lambda$ de $\mathcal{R}$ est de temps non borné si et seulement si on peut

construire des exécutions de temps arbitrairement long restant dans $\Lambda$.
Le lemme suivant nous permettra de conclure :

**Lemme 5.12** $\forall\Delta$ $\mathcal{P}||_{\Sigma_o}S_{\neg f}$ *a une exécution* $\Delta$ *fautive acceptante* $\Longleftrightarrow$ *Il existe une composante fortement connexe de* $\mathcal{R}_f$ *accessible, de temps non borné qui peut mener à un état final.*

**Preuve du lemme :**
Montrons l'implication gauche par la contraposée : supposons que pour toute composante fortement connexe de $\mathcal{R}_f$ le temps soit borné ou qu'on ne puisse atteindre un état final. Soit $\mathcal{B}_f$ l'ensemble des composantes fortements connexes de $\mathcal{R}_f$ pouvant mener à un état final ; pour $\Lambda \in \mathcal{B}_f$, on note $temps(\Lambda)$ le temps maximal d'une exécution dans $\Lambda$. Alors il découle facilement que la durée d'une exécution après une faute est bornée par $\sum\limits_{\Lambda\in\mathcal{B}_f} temps(\Lambda)$.
Montrons l'implication droite : supposons qu'il existe une composante fortement connexe $\Lambda$ de $\mathcal{R}_f$ accessible, de temps non borné qui peut mener à un état final et soit $\Delta \in \mathbb{N}$. Alors il existe une exécution fautive de $\mathcal{P}||_{\Sigma_o}S_{\neg f}$ qui passe $\Delta$ unités de temps dans $\Lambda$ puis va dans un état final ; une telle exécution est une exécution $\Delta$ fautive acceptante. $\square$

Le corrolaire 5.11 et le lemme 5.12 nous donnent que :
$\forall\Delta$, $\mathcal{P}$ n'est pas $\Delta$-observable par les *event-recordings* de granularité $\mu$
$\Longleftrightarrow$ Il existe une composante fortement connexe de $\mathcal{R}_f$ accessible, de temps non borné qui peut mener à un état final
Il ne nous reste plus qu'à décider en PSPACE l'existence d'une telle composante fortement connexe : en effet cela peut être fait en NLOG et la taille de $\mathcal{R}$ est bien exponentielle en $\mu$ et $\mathcal{P}$ puisque son nombre d'états est $|Q_{\mathcal{P}}||Q_{S_{\neg f}}||(2M_{\mathcal{P}}+2)^{|X_{\mathcal{P}}|}|X_{\mathcal{P}}|!$ (où $X_{\mathcal{P}}$ et $M_{\mathcal{P}}$ sont respectivement le nombre d'horloges et la constante maximale de $\mathcal{P}$). $\square$

# Conclusion

L'objet de ce travail était d'étudier la détection d'erreurs pour les systèmes temporisés. Dans un premier temps, nous avons rappelé les résultats existants de [Tri02] qui répondaient à la question «Est-il possible d'observer un système donné ?». Nous avons ensuite introduit deux notions pour poser le problème suivant : «Est-il possible d'observer un système donné avec un automate temporisé ? ». La première notion, l'observation précise, même si très expressive s'est révélée algorithmiquement impossible. Nous nous sommes alors intéressés au problème de l'observation par automates à ressources fixées que nous avons étudié pour deux classes : les automates déterministes où le problème s'est avéré soluble mais coûteux (2-EXPTIME-complet), et les *event-recording* où nous avons trouvé des algorithmes efficaces (les problèmes sont PSPACE-complets).

Le travail effectué nous a permis de dresser une partie du panorama de l'observation dans le cadre temporisé : certains problèmes sont indécidables (l'observation précise), d'autres solubles (l'observation dans le cas général, l'observation par déterministes ou *event-recording* à ressources bornées) avec des complexités variables. Il reste encore beaucoup de points à étudier pour dessiner les frontières de décidabilité et trouver de bons compromis entre complexité et expressivité.

Les perspectives de ce travail sont donc diverses :

- le problème de l'observation à ressources non-bornées reste ouvert ; de façon plus générale on pourrait essayer de ne fixer qu'une partie des ressources, par exemple fixer le nombre d'horloges et le pas des observateurs mais pas leur constante maximale
- une implémentation et des tests pourraient permettre de confirmer ou d'infirmer l'avantage en terme de complexité de l'observation par automates sur l'observation générale
- les liens entre contrôle et observation pourraient être approfondis ; en effet certaines techniques employées dans ce mémoire sont adaptées des résultats sur le contrôle temporisé de [DM02] et [BDMP03]. L'observation peut être vue comme un cas particulier du contrôle et les résultats du chapitre 5 semblent montrer que pour certaines classes d'automates le contrôle est strictement plus difficile que l'observation

- les méthodes utilisées dans le cadre de l'observation par *event-recordings* peuvent certainement se généraliser à une classe plus vaste d'automates temporisés : les automates « input-déterminé » de [DT04]

# Bibliographie

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

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

[AFH94] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. A determinizable class of timed automata. In *Proc. 6th International Conference on Computer Aided Verification (CAV'94)*, volume 818 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 1994.

[BDMP03] Patricia Bouyer, Deepak D'Souza, P. Madhusudan, and Antoine Petit. Timed control with partial observability. In *Proc. 15th International Conference on Computer Aided Verification (CAV'2003)*, volume 2725 of *Lecture Notes in Computer Science*, pages 180–192. Springer, 2003.

[CKS81] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery (JACM)*, 28(1) :114–133, 1981.

[DM02] Deepak D'Souza and P. Madhusudan. Timed control synthesis for external specifications. In *Proc. 19th International Symposium on Theoretical Aspects of Computer Science (STACS'02)*, volume 2285 of *Lecture Notes in Computer Science*, pages 571–582. Springer, 2002.

[DT04] Deepak D'Souza and Nicolas Tabareau. On timed automata with input-determined guards. In *Proc. Joint Conference on Formal Modelling and Analysis of Timed Systems and Formal Techniques in Real-Time and Fault Tolerant System (FORMATS+FTRTFT'04)*, Lecture Notes in Computer Science. Springer, 2004. To appear.

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[SSL+95] Meera Sampath, Raja Sengupta, Stéphane Lafortune, Kasim Sinnamohideen, and Demosthenis C. Teneketzis. Diagnosability of discrete event systems. *IEEE Transactions on Automatic Control*, 40(9) :1555–1575, 1995.

[SSL+96] Meera Sampath, Raja Sengupta, Stéphane Lafortune, Kasim Sinnamohideen, and Demosthenis C. Teneketzis. Failure diagnosis using discrete event systems. *IEEE Transactions on Control Systems Technology*, 4(2) :105–124, 1996.

[Tri02] Stavros Tripakis. Fault diagnosis for timed automata. In *Proc. 7th International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT'02)*, volume 2469 of *Lecture Notes in Computer Science*, pages 205–224. Springer, 2002.

[Tri03] Stavros Tripakis. Folk theorems on the determinization and minimization of timed automata. In *Proc. 1st International Workshop on Formal Modeling and Analysis of Timed Systems (FORMATS'03)*, volume 2791 of *Lecture Notes in Computer Science*, pages 182–188. Springer, 2003.

[Zie98] Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1–2) :135–183, 1998.