Chapitre 6

Diagnostic des systèmes temporisés

Dans ce chapitre nous donnons les principaux résultats concernant le diagnostic de fautes dans les systèmes temporisés. Le modèle de systèmes temporisés que nous prenons est celui des automates temporisés introduits dans les chapitre 3 et chapitre 4. Nous invitons le lecteur qui n'est pas familier avec ce modèle à consulter ces chapitres préalablement à la lecture de celui-ci.

6.1. Introduction

De nombreux systèmes informatiques (par exemple embarqués) pilotent des appareils où une défaillance peut conduire à des pertes de vies humaines (transports ferroviaire, aéronautique, etc.) ou à des désastres économiques en cas de panne (téléphonie mobile par exemple).

Depuis plus de vingt ans, des recherches ont donc été menées pour développer des outils et techniques en vue d'améliorer la fiabilité des logiciels. Parmi l'une des plus connues, on trouve le *model checking* [SCH 99] qui consiste à construire un système S et à en vérifier *formellement* des propriétés ϕ. Dans certains cas, on peut aussi faire de la *synthèse de contrôleur* (voir chapitre 5), c'est-à-dire calculer un contrôleur C pour S tel que le système en boucle fermé $C(S)$ satisfasse la propriété donnée ϕ.

Les techniques précédentes supposent que l'on peut construire un modèle complet du système (à vérifier ou contrôler), et que l'on peut tout observer. Ceci n'est pas toujours possible, même si les modèles développés dans les vingt dernières années sont de plus en plus expressifs (automates temporisés [ALU 94a] et hybrides [HEN 96]).

Chapitre rédigé par Franck CASSEZ et Stavros TRIPAKIS.

On a donc toujours recours à des développements industriels reposant sur le *test* en vue d'éradiquer les erreurs logicielles. On ne peut cependant pas imaginer toutes les erreurs possibles et assurer qu'un système est complètement fiable. Une approche très utilisée consiste à détecter des comportements anormaux du système, des *fautes*, et en cas de fautes, à prendre une décision de traitement (arrêt du système, réparation, etc.).

La notion de *diagnostic* des systèmes a été introduite au milieu des années 1990 dans [SAM 95, SAM 96] pour les systèmes à événements discrets (SED). Ces deux articles donnent les définitions de base du diagnostic des SED et ont été la source de nombreux travaux. Le contexte du diagnostic des SED est le suivant :

– on dispose d'un modèle (automate fini) d'un système S, dans lequel des *fautes* peuvent se produire. Ces fautes correspondent à des événements spéciaux $f_1, \ldots, f_n$. Cependant les fautes ne peuvent pas être observées (on a des capteurs que pour certains événements mais pas tous et pas pour les fautes) ; dans S il y a des événements observables et d'autres (comme les fautes) inobservables ;

– le diagnostic consiste à détecter les fautes en observant uniquement les événements observables. On demande aussi de détecter une faute dans un *temps* fixé à l'avance, et on pose la question « est-il possible de détecter une faute f_i dans S dans un délai maximum de Δ unités de temps ? » Dans le cas des SED, le temps est mesuré en nombre de transitions du système après une faute.

Pour être plus précis on peut bien entendu prendre un modèle temporisé du système S, par exemple un automate temporisé [ALU 94a]. Le problème de diagnostic pour les automates temporisés a été introduit dans [TRI 02] et des résultats complémentaires ont récemment été publiés dans [BOU 05]. Il reste quelques problèmes ouverts dans le cadre du diagnostic temporisé qui sont donnés dans la section 6.6.

Dans la section 6.2 qui suit, on rappelle brièvement les notions de base concernant les langages temporisés, automates temporisés. Le lecteur peut se référer aux chapitre 3 et chapitre 4 pour un exposé détaillé de ces concepts.

Dans la section 6.3, on donne la définition formelle d'un *diagnostiqueur* qui est un observateur chargé de détecter les fautes dans un système. On définit aussi les problèmes de diagnostic qui seront étudiés dans la suite et une condition (générale) nécessaire et suffisante de *diagnosticabilité*. La section 6.4 est consacrée au diagnostic des systèmes à événements discrets (SED). Les algorithmes de cette section sont ensuite étendus aux cas des systèmes temporisés dans la section 6.5.

6.2. Rappels sur les systèmes temporisés

Dans cette section, nous rappelons les définitions de base concernant les systèmes temporisés qui sont utiles dans ce chapitre. Pour une exposition détaillée, elles ont été

introduites dans les chapitre 3 et chapitre 4. L'ensemble $\mathbb{R}$ est l'ensemble des réels, $\mathbb{R}_{\geq 0}$ désigne l'ensemble des réel positifs ou nuls. $\mathbb{B} = \{\top, \bot\}$ est l'ensemble des booléens, $\mathbb{N}$ est l'ensemble des entiers naturels, $\mathbb{Z}$, des entiers relatifs et $\mathbb{Q}$, des rationnels. Si X est un ensemble d'horloges, on note $\mathcal{C}(X)$ l'ensemble des *contraintes rectangulaires* sur l'ensemble de variables X, c'est-à-dire des conjonctions de contraintes de la forme $x \bowtie c$ avec $c \in \mathbb{Z}$ et $\bowtie \in \{\leq, <, =, >, \geq\}$. Une valuation des horloges de X est une fonction $v : X \to \mathbb{R}_{\geq 0}$. Pour $\delta \in \mathbb{R}$, on note $v + \delta$ la valuation définie par $(v + \delta)(x) = v(x) + \delta$.

6.2.1. *Mots temporisés et langages temporisés*

Soit Σ un alphabet fini. Un mot temporisé fini (resp. infini) sur Σ est un mot de $(\mathbb{R}_{\geq 0}.\Sigma)^*.\mathbb{R}_{\geq 0}$ (resp. $(\mathbb{R}_{\geq 0}.\Sigma)^\omega$). On note *Duration*$(\rho)$ la durée d'un mot temporisé qui est la somme[1] des durées apparaissant dans ρ et *Unt*(ρ) la projection de ρ sur Σ. Dans cette section on écrit les mots temporisés sous la forme $0.4\ a\ 1.0\ b\ 2.7\ c \cdots$ où les valeurs réelles sont les durées qui s'écoulent entre deux événements : ainsi c a lieu à la date absolue 4.1. Lorsque l'on projette un mot temporisé ρ sur un alphabet $\Sigma' \subseteq \Sigma$, on doit donc « recalculer » les valeurs de durée séparant deux événements. On note $\boldsymbol{\pi}_{/\Sigma'}(\rho)$ la projection sur l'alphabet Σ'. Par exemple, $\boldsymbol{\pi}_{/\{a,c\}}(0.4\ a\ 1.0\ b\ 2.7\ c) = 0.4\ a\ 3.7\ c$. Pour un langage L, $\boldsymbol{\pi}_{/\Sigma'}(L) = \{\boldsymbol{\pi}_{/\Sigma'}(\rho) \mid \rho \in L\}$.

TW$^*(\Sigma)$ est l'ensemble des mots temporisés finis sur Σ, *TW*$^\omega(\Sigma)$, l'ensemble des mots temporisés infinis et *TW*$^\infty(\Sigma) =$ *TW*$^*(\Sigma) \cup$ *TW*$^\omega(\Sigma)$. Un langage temporisé L est un sous-ensemble de *TW*$^\infty(\Sigma)$ et *Unt*$(L) = \{$*Unt*$(\rho) \mid \rho \in L\}$.

6.2.2. *Automate temporisé*

DÉFINITION 6.2.1 (AUTOMATE TEMPORISÉ) *Un* automate temporisé *A est un tuple* $(L, \ell_0, X, \Sigma, E, Inv, F, R)$ *où :*

- L *est un ensemble fini de* localités *;*
- ℓ_0 *est la* localité initiale *;*
- X *est un ensemble fini d'*horloges *à valeurs réelles positives ;*
- Σ *est un ensemble fini d'*actions *;*
- $E \subseteq L \times \mathcal{C}(X) \times \Sigma \times 2^X \times L$ *est un ensemble fini de* transitions *;*
- $Inv \in \mathcal{C}(X)^L$ *associe un* invariant *à chaque localité ;*
- $F \subseteq L$ *et* $R \subseteq L$ *sont respectivement les localités* finales *et* répétées *de* A. □

1. Si cette somme est infinie, la durée du mot est ∞. La durée d'un mot infini n'est pas forcément ∞ car il y a des mots *zénon*.

La sémantique des automates temporisés en termes de systèmes de transitions temporisés est donnée dans les chapitre 3 et chapitre 4. Une exécution ρ de A à partir de (q_0, v_0) est une séquence de la forme :

$$\rho = (q_0, v_0) \xrightarrow{\delta_0} (q_0, v_0 + \delta_0) \xrightarrow{a_1} (q_1, v_1) \cdots \xrightarrow{a_n} (q_n, v_n) \xrightarrow{\delta_n} (q_n, v_n + \delta),$$

où $q_i \in L$, v_i est une valuation des horloges de X, $a_i \in \Sigma$ et $\delta_i \in \mathbb{R}_{\geq 0}$. On note $\mathbf{0}$ la valuation où toutes les horloges sont nulles. L'ensemble des exécutions finies (resp. infinies) à partir d'un état $s = (q, v)$ est noté $\mathit{Runs}^*(s, A)$ (resp. $\mathit{Runs}^\omega(s, A)$) et on pose $\mathit{Runs}(A) = \mathit{Runs}((\ell_0, \mathbf{0}), A)$ et $\mathit{Runs}^\omega(A) = \mathit{Runs}^\omega((\ell_0, \mathbf{0}), A)$. Si ρ est finie et se termine dans s_n, on définit $\mathit{last}(\rho) = s_n$.

La *trace*, $\mathit{tr}(\rho)$, d'une exécution ρ est le mot temporisé $\delta_0 a_0 \delta_1 a_1 \cdots a_n \delta_n$. La durée de ρ est $\mathit{Duration}(\rho) = \mathit{Duration}(\mathit{tr}(\rho))$. Pour $V \subseteq \mathit{Runs}(A)$, on définit $\mathit{Tr}(V) = \{\mathit{tr}(\rho) \mid \rho \in V\}$, qui est l'ensemble des traces des exécutions de V.

Un mot temporisé fini (resp. infini) ρ est *accepté* par A s'il est la trace d'une exécution de A se terminant dans une localité de F (resp. s'il passe infiniment souvent par une localité de R). $\mathcal{L}^*(A)$ (resp. $\mathcal{L}^\omega(A)$) est l'ensemble des traces de mots temporisés finis (resp. infinis) acceptés par A, et $\mathcal{L}(A) = \mathcal{L}^*(A) \cup \mathcal{L}^\omega(A)$ est l'ensemble des mots temporisés acceptés par A.

Dans ce chapitre on utilise des automates temporisés avec action une action *invisible*, notée τ et on note $\Sigma_\tau = \Sigma \cup \{\tau\}$ l'alphabet Σ complété par τ. On omet les ensembles d'états R et F lorsqu'ils ne sont pas utiles.

Le *graphe des régions* $\mathit{RG}(A)$ d'un automate temporisé A (voir chapitre 4) est un automate fini sur l'alphabet (symbolique) $E' \cup \{\tau\}$, avec $E' = \mathcal{C}(X) \times \Sigma \times 2^X$. Ses états sont constitués de couples (ℓ, r) où $\ell \in L$ est une localité de A et r une région de $\mathbb{R}^X_{\geq 0}$. Les arcs du graphe sont des triplets (s, t, s') avec s, s' des états de $\mathit{RG}(A)$ et $t \in E' \cup \{\tau\}$. L'étiquette τ indique le passage du temps. Le franchissement d'une transition discrète de A d'étiquette τ, correspond dans $\mathit{RG}(A)$ à une transition d'étiquette (g, τ, R) où est g est une garde et $R \subseteq X$. L'état initial de $\mathit{RG}(A)$ est $(\ell_0, \mathbf{0})$. Un état final (resp. répété) de $\mathit{RG}(A)$ est un état dont la localité est finale (resp. répétée) de A. Une propriété essentielle du graphe des régions est que $\mathcal{L}(\mathit{RG}(A)) = \mathit{Unt}(\mathcal{L}(A))$ ce qui peut être reformulé en : (i) si w est accepté par $\mathit{RG}(A)$ alors on peut « temporiser » w en v tel que $\mathit{Unt}(v) = w$ et v est accepté par A, et (ii) si w est accepté par A alors $\mathit{Unt}(w)$ est accepté par $\mathit{RG}(A)$.

6.2.3. *Produit d'automates temporisés*

Le produit des automates temporisés est défini de manière standard : les automates se synchronisent sur les actions communes exceptée l'action invisible τ.

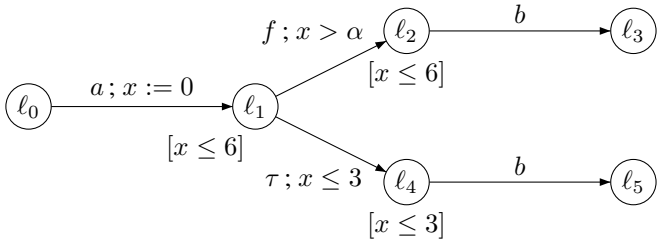


Figure 6.1. *L'automate* $\mathcal{A}(\alpha)$

DÉFINITION 6.2.2 (PRODUIT SYNCHRONISÉ) *Soient* $A_i = (L_i, \ell_0^i, X_i, \Sigma_\tau^i, E_i, \mathit{Inv}_i)$, $i \in \{1,2\}$, *deux automates temporisés tels que* $X_1 \cap X_2 = \emptyset$. *Le* produit synchronisé *de* A_1 *et* A_2 *est l'automate temporisé* $A_1 \times A_2 = (L, \ell_0, X, \Sigma_\tau, E, \mathit{Inv})$ *défini par :*

– $L = L_1 \times L_2$;

– $\ell_0 = (\ell_0^1, \ell_0^2)$;

– $\Sigma = \Sigma^1 \cup \Sigma^2$;

– $X = X_1 \cup X_2$;

– $E \subseteq L \times \mathcal{C}(X) \times \Sigma \times 2^X \times L$ *et* $((\ell_1, \ell_2), g_{1,2}, \sigma, R, (\ell_1', \ell_2')) \in E$ *si :*

- soit $\sigma \in \Sigma_1 \cap \Sigma_2$, *et (i)* $(\ell_k, g_k, \sigma, r_k, \ell_k') \in E_k$ *pour* $k = 1$ *et* $k = 2$; *(ii)* $g_{1,2} = g_1 \wedge g_2$ *et (iii)* $R = r_1 \cup r_2$;

- soit pour $k = 1$ *ou* $k = 2$, $\sigma \in (\Sigma_k \setminus \Sigma_{3-k}) \cup \{\tau\}$, *et (i)* $(\ell_k, g_k, \sigma, r_k, \ell_k') \in E_k$; *(ii)* $g_{1,2} = g_k$ *et (iii)* $R = r_k$;

– $\mathit{Inv}(\ell_1, \ell_2) = \mathit{Inv}(\ell_1) \wedge \mathit{Inv}(\ell_2)$. □

6.2.4. *Automates temporisés pour le diagnostic*

Pour modéliser les systèmes temporisés à diagnostiquer, nous prenons les automates temporisés sur l'alphabet $\Sigma_{\tau,f} = \Sigma_\tau \cup \{f\}$ où f est l'événement de faute et τ est l'événement inobservable. Les actions inobservables sont toutes considérées comme l'action τ : par définition elles ont toutes le même traitement lors des observations (elles sont « effacées ») et il est donc possible d'utiliser un seul symbole τ pour modéliser toutes ces actions. Un modèle du système à diagnostiquer est un automate temporisé $A = (L, \ell_0, X, \Sigma_{\tau,f}, E, \mathit{Inv})$. Un exemple d'automate temporisé est donnée sur la figure 6.1 (α est un paramètre qui a une valeur entière positive). On indique les invariants entre crochets « [» et «] » sous la forme $[x \leq 3]$. Soit $\Delta \in \mathbb{N}$. Une exécution ρ de A telle que :

$$\rho = (q_0, v_0) \xrightarrow{\delta_0} (q_0, v_0 + \delta_0) \xrightarrow{a_1} (q_1, v_1) \cdots \xrightarrow{a_n} (q_n, v_n) \xrightarrow{\delta_n} (q_n, v_n + \delta)$$

est Δ-fautive si (1) il existe un indice i tel que $a_i = f$ et (2) la durée de l'exécution $\rho' = (q_i, v_i) \xrightarrow{\delta_i} \cdots \xrightarrow{\delta_n} (q_n, v_n + \delta_n)$ est supérieure à Δ c'est-à-dire $\mathit{Duration}(\rho') \geq$

Δ. *Faulty*$_{\geq\Delta}(A)$ représente les exécutions Δ-fautives de A. Il s'ensuit que si $\Delta' \geq \Delta$ alors *Faulty*$_{\geq\Delta'}(A) \subseteq$ *Faulty*$_{\geq\Delta}(A)$. On note *Faulty*$(A) = \cup_{\Delta\geq 0}$*Faulty*$_{\geq\Delta}(A) =$ *Faulty*$_{\geq 0}(A)$ l'ensemble des exécutions fautives de A, et *NonFaulty*(A) l'ensemble complémentaire de *Faulty*(A), donc contenant les exécutions non fautives de A. Enfin, on note :

$$\textit{Faulty}^{tr}_{\geq\Delta}(A) = \textit{Tr}(\textit{Faulty}_{\geq\Delta}(A)) \text{ et } \textit{NonFaulty}^{tr}(A) = \textit{Tr}(\textit{NonFaulty}(A)),$$

qui sont les ensembles de traces des exécutions fautives et non fautives.

6.3. Les problèmes de diagnostic

Dans cette section, on définit la notion de *diagnostiqueur*. On formule ensuite les problèmes de diagnostic, et on donne une condition nécessaire et suffisante de *diagnosticabilité* qui sera utile dans la suite.

6.3.1. *Diagnostiqueur*

Un *diagnostiqueur* est une machine qui observe le système et doit déclencher une alarme quand une faute est détectée. On suppose que l'on ne peut observer que les événements de Σ et que les fautes (f) et certains autres événements (τ) sont inobservables. Le diagnostiqueur doit donc déduire l'occurrence d'une faute à l'aide de l'observation des autres événements, et des délais écoulés entre leurs occurrences. Si le système à observer génère le mot temporisé w, on ne pourra observer que $\boldsymbol{\pi}_{/\Sigma}(w)$.

Il se peut qu'un diagnostiqueur ne déclenche pas l'alarme immédiatement après l'occurrence d'une faute. On demande cependant qu'il le fasse dans un délai borné. Dans les systèmes temporisés, ce délai est un entier $\Delta \in \mathbb{N}$.

DÉFINITION 6.3.1 ((Σ, Δ)-DIAGNOSTIQUEUR) *Soit A un automate temporisé sur $\Sigma_{\tau,f}$ et $\Delta \in \mathbb{N}$. Un* (Σ, Δ)-diagnostiqueur *pour A est une fonction $D : TW^*(\Sigma) \to \{0, 1\}$ telle que :*

– *pour tout $\rho \in$ NonFaulty(A), $D(\boldsymbol{\pi}_{/\Sigma}(tr(\rho))) = 0$,*

– *pour tout $\rho \in$ Faulty$_{\geq\Delta}(A)$, $D(\boldsymbol{\pi}_{/\Sigma}(tr(\rho))) = 1$.* □

A est (Σ, Δ)-diagnosticable s'il existe un (Σ, Δ)-diagnostiqueur pour A. A est Σ-diagnosticable s'il existe un $\Delta \in \mathbb{N}$ tel que A est (Σ, Δ)-diagnosticable. Dans la suite on omet parfois de mentionner l'alphabet Σ.

REMARQUE 6.3.1 *Rien n'est requis pour les mots Δ'-fautifs avec $\Delta' < \Delta$. Ainsi un diagnostiqueur peut « changer d'avis » répondre* 1 *pour un mot Δ'-fautifs,* 0 *pour un mot Δ''-fautif avec $\Delta' < \Delta'' < \Delta$.*

EXEMPLE 6.3.1 *L'automate $\mathcal{A}(3)$ de la figure 6.1 (extrait de [TRI 02]) est 3-diagnosticable. Pour les mots du type $t.a.\delta.b.t'$ avec $\delta \leq 3$, aucune faute n'est survenue, alors que pour les mots où $\delta > 3$ une faute est forcément survenue. Il suffit donc de construire la fonction D correspondante. Comme on doit attendre un b pour annoncer une faute, D ne peut pas diagnostiquer en 2 unités de temps si f survient à la date $x = 3.1$ et b à la date $x = 6$. Si $\alpha = 2$, dans $\mathcal{A}(2)$ on les exécutions suivantes :*

$$\begin{aligned}
\rho_1(\delta) &= (\ell_0,0) \xrightarrow{a} (\ell_1,0) \xrightarrow{2.5} (\ell_1,2.5) \xrightarrow{f} (\ell_2,2.5) \\
&\qquad \xrightarrow{0.2} (\ell_2,2.7) \xrightarrow{b} (\ell_3,2.7) \xrightarrow{\delta} (\ell_2,2.7+\delta) \\
\rho_2(\delta) &= (\ell_0,0) \xrightarrow{a} (\ell_1,0) \xrightarrow{2.5} (\ell_1,2.5) \xrightarrow{\tau} (\ell_4,2.5) \\
&\qquad \xrightarrow{0.2} (\ell_4,2.7) \xrightarrow{b} (\ell_5,2.7) \xrightarrow{\delta} (\ell_5,2.7+\delta)
\end{aligned}$$

Elles satisfont $\boldsymbol{\pi}_{/\Sigma}(tr(\rho_1(\delta))) = \boldsymbol{\pi}_{/\Sigma}(tr(\rho_2(\delta)))$, et ceci pour tout $\delta \geq 0$. Pour tout $\Delta \in \mathbb{N}$, il existe donc deux exécutions $\rho_1(\Delta)$ et $\rho_2(\Delta)$ qui produisent les mêmes observations. Il ne peut donc exister de diagnostiqueur pour $\mathcal{A}(2)$.

6.3.2. *Problèmes de diagnostic*

Les problèmes de diagnostic auxquels on s'intéresse sont les suivants :

PROBLÈME 6.3.1 (DIAGNOSTICABILITÉ)
ENTRÉE : *Un automate temporisé $A = (L, \ell_0, X, \Sigma_{\tau,f}, E, Inv)$.*
PROBLÈME : *Est-ce que A est Σ-diagnosticable ?*

PROBLÈME 6.3.2 (DÉLAI MAXIMUM DE DÉTECTION DE FAUTES)
ENTRÉE : *Un automate temporisé $A = (L, \ell_0, X, \Sigma_{\tau,f}, E, Inv)$.*
PROBLÈME : *Si A est Σ-diagnosticable, quel est le plus petit entier Δ tel que A est (Σ, Δ)-diagnosticable ?*

PROBLÈME 6.3.3 (SYNTHÈSE DE DIAGNOSTIQUEUR)
ENTRÉE : *Un automate temporisé $A = (L, \ell_0, X, \Sigma_{\tau,f}, E, Inv)$.*
PROBLÈME : *Si A est (Σ, Δ)-diagnosticable, calculer un (Σ, Δ)-diagnostiqueur D.*

6.3.3. *Test de diagnosticabilité*

D'après la définition 6.3.1, A est diagnosticable, si et seulement si, il existe $\Delta \in \mathbb{N}$ tel que A est (Σ, Δ)-diagnosticable. Ceci est équivalent à :

$$A \textbf{ n'est pas } \text{diagnosticable} \Longleftrightarrow \forall \Delta \in \mathbb{N}, A \text{ n'est pas } (\Sigma, \Delta)\text{-diagnosticable.} \quad [6.1]$$

On peut donner une définition en termes de langages temporisés de la (Σ, Δ)-diagnosticabilité : A est (Σ, Δ)-diagnosticable si et seulement si :

$$\boldsymbol{\pi}_{/\Sigma}(\mathit{Faulty}^{tr}_{\geq \Delta}(A)) \cap \boldsymbol{\pi}_{/\Sigma}(\mathit{NonFaulty}^{tr}(A)) = \emptyset. \quad [6.2]$$

En effet, si l'équation [6.2] est satisfaite on peut définir un diagnostiqueur D par : $D(\rho) = 1$ si et seulement si $\rho \in \boldsymbol{\pi}_{/\Sigma}(\mathit{Faulty}^{tr}_{\geq \Delta}(A))$. Si elle n'est pas vérifiée, il y a deux exécutions (ρ_1, ρ_2) avec $\rho_1 \in \mathit{Faulty}_{\geq \Delta}(A)$, $\rho_2 \in \mathit{NonFaulty}(A)$ dont les traces sont telles que $tr(\rho_1)$ et $tr(\rho_2)$ produisent la même observation une fois projetées sur Σ ; donc aucun diagnostiqueur ne peut exister car il devrait annoncer à la fois 1 et 0. On peut en déduire une condition nécessaire et suffisante de non-diagnosticabilité :

$$A \textbf{ n'est pas } \text{diagnosticable} \Longleftrightarrow \forall \Delta \in \mathbb{N}, \begin{cases} \exists \rho \in \mathit{NonFaulty}(A) \\ \exists \rho' \in \mathit{Faulty}_{\geq \Delta}(A) \\ \textit{tels que } \boldsymbol{\pi}_{/\Sigma}(tr(\rho)) = \boldsymbol{\pi}_{/\Sigma}(tr(\rho')) \end{cases} \quad [6.3]$$

ou en d'autres termes, il n'y a pas deux exécutions (ρ_1, ρ_2) avec $\rho_1 \in \mathit{Faulty}_{\geq \Delta}(A)$, $\rho_2 \in \mathit{NonFaulty}(A)$ dont les traces sont telles que $tr(\rho_1)$ et $tr(\rho_2)$ produisent la même observation une fois projetées sur Σ.

Dans la section suivante, on montre comment décider les problèmes 6.3.1, 6.3.2 et 6.3.3 pour les systèmes non temporisés. Dans la section 6.5 on montre comment étendre les solutions au cas des automates temporisés.

6.4. Diagnostic des systèmes à événements discrets

Le diagnostic des systèmes à événements discrets a été introduit dans [SAM 95, SAM 96]. Dans cette approche, on dispose d'un modèle du système à observer. Ce modèle est une description comportementale du système, incluant tous les scénarios possibles y compris ceux avec des fautes.

6.4.1. *Système à évènements discrets pour le diagnostic*

Dans les systèmes non temporisés le temps qui passe est compté en termes du nombre d'actions (observables et non observables) survenues dans le système. Pour une exécution ρ, la durée de ρ est donc le nombre de transitions de ρ.

Les systèmes à événements discrets sont modélisés par des automates finis. De plus, ces automates n'acceptent que des mots finis et le langage accepté est clos par préfixe[2]. En utilisant la définition 6.2.1 du paragraphe 6.2.2, ceci revient à prendre $X = \emptyset$ (pas d'horloge, donc pas d'invariant ni de garde dans les transitions), $F = L$ et $R = \emptyset$.

Pour indiquer que A est un automate fini, on note $A = (Q, q_0, \Sigma_{\tau,f}, \delta)$. Une *exécution* ρ à partir de l'état s_0 dans A est une séquence finie ou infinie de transitions :

$$\rho = s_0 \xrightarrow{\lambda_1} s_1 \xrightarrow{\lambda_2} s_2 \cdots s_{n-1} \xrightarrow{\lambda_n} s_n \cdots \qquad [6.4]$$

telles que $\lambda_i \in \Sigma_{\tau,f}$. Dans le cas des systèmes non temporisés, on pose *Duration*$(\rho) = n$ (∞ si ρ est infinie).

Les définitions du paragraphe 6.2.4 (exécutions k-fautives, etc.) et de la section 6.3 (diagnostiqueur, etc.) s'appliquent au cas non temporisé en utilisant la définition de durée des mots *Duration*$(\rho) = n$ si ρ est un exécution de longueur n. Les problèmes de diagnostic dans le cas non temporisé sont ceux exposés dans la section 6.3.

REMARQUE 6.4.1 *Prendre un automate temporisé où les actions discrètes successives sont séparés de une unité de temps, n'est pas équivalent à prendre un système complètement non temporisé. Par exemple, un automate temporisé de ce type qui génère les mots temporisés* $1.f.1.a$ *et* $1.\tau.1.\tau.1.a$ *est* 1*-diagnosticable : après le mot temporisé* $2.a$ *on annonce une faute. Cet automate n'est pas diagnosticable si l'on ne voit pas les durées de* 1 *unité de temps : les mots* $f.a$ *et* $\tau^2.a$ *ont la même observation. Le diagnostic des systèmes discrets ne se réduit pas à un cas particulier de diagnostic des automates temporisés.*

On suppose que toute exécution fautive de A de longueur n peut être prolongée en une exécution fautive de longueur $n + 1$. Cette hypothèse est utile pour les preuves de certains lemmes et théorèmes de cette section. Si A ne satisfait pas cette hypothèse, il est facile d'ajouter des boucles d'action τ sur les états *deadlock* de A et on rentre alors dans ce cadre. L'ajout de boucles d'action τ ne change pas les observations de A.

2. L est clos par préfixe si $\forall w \in \Sigma^*$, s'il existe $w' \in \Sigma^*$ tel que $w.w' \in L$ alors $w \in L$.

6.4.2. *Vérification de la diagnosticabilité*

Pour décider si A est diagnosticable, on construit un produit synchronisé d'automates, $A_1 \times A_2$, tel que A_1 se comporte exactement comme A mais mémorise quand une faute est survenue. A_2 se comporte comme A mais ne génère que les comportements non fautifs de A.

Plus précisément $A_1 = (Q \times \{0,1\}, (q_0, 0), \Sigma_\tau, \rightarrow_1)$ avec :
– $(q,n) \xrightarrow{l}_1 (q',n)$ si $q \xrightarrow{l} q'$ et $l \in \Sigma \cup \{\tau\}$;
– $(q,n) \xrightarrow{\tau}_1 (q',1)$ si $q \xrightarrow{f} q'$, (n est mis à 1 après une faute) ;

et $A_2 = (Q, q_0, \Sigma_\tau, \rightarrow_2)$ avec : $q \xrightarrow{l}_2 q'$ si $q \xrightarrow{l} q'$ et $l \in \Sigma \cup \{\tau\}$. Soit $A_1 \times A_2$ le produit synchronisé de A_1 et A_2 (ils ne se synchronisent que sur les actions communes sauf τ, voir définition 6.2.2) . On note $\rightarrow_{1,2}$ la relation de transition de $A_1 \times A_2$.

On définit le prédicat A_1*Move* (resp. A_2*Move*) sur les transitions de $A_1 \times A_2$ qui indique si A_1 (resp. A_2) participe à une transition du produit. Pour une exécution $\rho \in$ *Runs*$(A_1 \times A_2)$ on note $\rho_{|1}$, l'exécution de A_1 qui correspond à la séquence de transitions t_i satisfaisant A_1*Move*(t_i), et $\rho_{|2}$ l'exécution de A_2 correspondant à la séquence de transitions t_k satisfaisant A_2*Move*(t_k). Par définition de $A_1 \times A_2$, on a $\boldsymbol{\pi}_{/\Sigma}(tr(\rho_{|1})) = \boldsymbol{\pi}_{/\Sigma}(tr(\rho_{|2}))$. Soit *Runs*$_{\geq k}(A_1 \times A_2)$ l'ensemble des exécutions de $A_1 \times A_2$ contenant un état fautif $((q_1,1),q_2)$ suivi par au moins k actions de A_1. On a les lemmes suivants :

LEMME 6.4.1 *Soit $\rho \in$ Runs$_{\geq k}(A_1 \times A_2)$. Alors les projections $\rho_{|1} \in$ Faulty$_{\geq k}(A)$ et $\rho_{|2} \in$ NonFaulty(A). De plus $\boldsymbol{\pi}_{/\Sigma}(tr(\rho_{|1})) = \boldsymbol{\pi}_{/\Sigma}(tr(\rho_{|2}))$.*

LEMME 6.4.2 *Soit $\rho_1 \in$ Faulty$_{\geq k}(A)$ et $\rho_2 \in$ NonFaulty(A) tels que $\boldsymbol{\pi}_{/\Sigma}(tr(\rho_1)) = \boldsymbol{\pi}_{/\Sigma}(tr(\rho_2))$. Alors il existe une exécution $\rho \in$ Runs$_{\geq k}(A_1 \times A_2)$ telle que $\rho_{|1} = \rho_1$ et $\rho_{|2} = \rho_2$.*

PREUVE 6.4.1 *Par définition de $A_1 \times A_2$, il suffit de remarquer qu'une exécution de $A_1 \times A_2$ peut être décomposée en deux exécutions, une de A_1 et une de A_2, qui ont même projection sur Σ. Les preuves des lemmes 6.4.1 et 6.4.2 découlent directement de cette observation.* □

Pour décider du problème 6.3.1, on va construire un automate $\mathcal{B}$ tel que $\mathcal{B}$ accepte un mot infini si et seulement si A n'est pas diagnosticable.

$\mathcal{B}$ est une version « étendue » de $A_1 \times A_2$: dans $\mathcal{B}$, il y a une variable booléenne z qui indique si A_1 a participé lors de la dernière transition franchie (c'est-à-dire, la dernière transition franchie t satisfait $A_1\mathit{Move}(t)$). Un état de $\mathcal{B}$ est donc un couple (s, z) où s est un état de $A_1 \times A_2$. $\mathcal{B}$ est un tuple $((Q \times \{0,1\} \times Q) \times \{0,1\}, ((q_0, 0), q_0, 0), \Sigma_\tau, \longrightarrow_{\mathcal{B}}, \emptyset, R_{\mathcal{B}})$ avec :

– $(s, z) \xrightarrow{\sigma}_{\mathcal{B}} (s', z')$ si (i) il existe une transition $t : s \xrightarrow{\sigma}_{1,2} s'$ dans $A_1 \times A_2$, et (ii) $z' = 1$ si $A_1\mathit{Move}(t)$ et $z' = 0$ sinon ;

– $R_{\mathcal{B}} = \{(((q, 1), q'), 1) \mid ((q, 1), q') \in A_1 \times A_2\}$.

$\mathcal{B}$ accepte donc un langage de mots infinis $\mathcal{L}(\mathcal{B}) = \mathcal{L}^\omega(\mathcal{B}) \subseteq \Sigma^\omega$. Ce langage satisfait le théorème suivant :

THÉORÈME 6.4.1 *$\mathcal{L}^\omega(\mathcal{B}) \neq \emptyset \iff A$ n'est pas Σ-diagnosticable.*

PREUVE 6.4.2
Preuve de $\Longrightarrow$. Supposons que $\mathcal{L}^\omega(\mathcal{B}) \neq \emptyset$. Soit $\rho \in \mathcal{L}^\omega(\mathcal{B})$: ρ a une infinité d'états fautifs à cause de la définition de $R_{\mathcal{B}}$. Soit $k \in \mathbb{N}$. Soit $\rho[i_k]$ un préfixe fini de ρ qui contient plus de k actions de A_1 (pour tout k un tel indice i_k existe car ρ contient une infinité d'actions de A_1.) $\rho[i_k] \in \mathit{Runs}_{\geq k}(A_1 \times A_2)$ et par le lemme 6.4.1, il s'ensuit que $\rho[i_k]_{|1} \in \mathit{Faulty}_{\geq k}(A)$ et $\rho[i_k]_{|2} \in \mathit{NonFaulty}(A)$ et aussi $\pi_{/\Sigma}(\mathit{tr}(\rho_{|1})) = \pi_{/\Sigma}(\mathit{tr}(\rho_{|2}))$. L'équation [6.3] *est donc satisfaite et A n'est pas diagnosticable.*

Preuve de $\Longleftarrow$. Réciproquement, supposons que A n'est pas Σ-diagnosticable. Alors par l'équation [6.3] *et le lemme 6.4.2, pour tout $k \in \mathbb{N}$, il existe une exécution $\rho_k \in \mathit{Runs}_{\geq k}(A_1 \times A_2)$. Comme $A_1 \times A_2$ est fini, il y a au moins un état $(q_1, 1), q_2$ de $A_1 \times A_2$ qui est source d'un nombre infini d'exécutions ρ'_k qui contiennent chacune plus de k actions de A_1. Comme $A_1 \times A_2$ est aussi à branchement fini, par le lemme de König, il y a une exécution infinie à partir de $(q_1, 1), q_2$ qui contient une infinité d'actions de A_1. Donc $\mathcal{L}^\omega(\mathcal{B}) \neq \emptyset$.* □

REMARQUE 6.4.2 *La condition d'acceptation impliquant z impose que A_1 fasse infiniment souvent une transition, et ainsi que le temps progresse dans A_1. Sans cette condition, il est possible que A_1 refuse d'avancer, bloque le temps (compté en termes de nombre de pas discrets de A_1), et que A_2 fasse une infinité de transitions τ. Dans ce cas, on pourrait déclarer A non diagnosticable alors qu'il n'y a pas d'éxécutions fautives arbitrairement grande de A, ce qui ne correspond pas à la définition* [6.3].

Le théorème 6.4.1 permet d'obtenir un algorithme polynomial pour décider le problème 6.3.1 :

COROLLAIRE 6.4.1 *Le problème 6.3.1 est décidable en temps polynomial* $O(|A|^2)$.

PREUVE 6.4.3
On peut décider le vide sur les automates de Büchi en temps polynomial et avec des algorithmes « à la volée » [HOL 96] : si un automate H *a* n *états et* m *transitions, alors décider si* $\mathcal{L}^\omega(H) = \emptyset$ *peut se faire en temps* $O(n+m)$. *La taille de* $\mathcal{B}$ *est* $4 \times |Q|^2$ *soit* $O(|Q|^2)$. *On peut donc décider le problème 6.3.1 en temps* $O(|Q|^2)$. □

Des algorithmes polynomiaux sont décrits dans [JIA 01, YOO 02] pour décider le problème 6.3.1. Dans ces deux articles, le système à étudier ne doit pas contenir de cycles d'événements inobservables. L'algorithme précédent reposant sur le test du vide des automates de Büchi n'a pas cette limitation : il se peut en effet qu'il y ait des cycles d'événements inobservables sans que cela empêche le diagnostic. Dans l'approche que nous proposons, on complète même l'automate de départ avec des boucles de τ pour assurer la *vivacité* du système. Dans [JIA 01, YOO 02], on construit aussi un produit d'automates pour décider la diagnosticabilité. Cependant, la construction est « symétrique », et c'est comme si l'on composait A_1 avec lui même. Si des cycles fautifs d'actions de A_1 sont dans le produit, ils y sont donc en double exemplaire. Dans l'algorithme présenté ici, la construction n'est pas symétrique : A_1 génère tous les comportements de A, et A_2 uniquement les non fautifs.

La réduction du problème de diagnosticabilité à un problème de « test du vide sur un automate de Büchi » est donc intéressante à plusieurs titres :

– d'un point de vue théorique, l'algorithme est simple et la preuve de correction est très facile ;

– pour l'aspect pratique et l'implémentation, on peut utiliser les outils de vérification très efficaces (par exemple SPIN [HOL 05]) pour décider de la diagnosticabilité. Ces algorithmes fournissent un contre-exemple dans le cas où le système n'est pas diagnosticable.

6.4.3. *Calcul du délai maximal de détection des fautes*

Pour calculer le plus petit k tel que A est (Σ, k)-diagnosticable, on procède comme suit : si A est Σ-diagnosticable, dans $\mathcal{B}$, il n'y a pas d'exécution *fautive* qui passe par une infinité d'actions de A_1. Ainsi toute exécution qui débute dans un état fautif de $A_1 \times A_2$ est suivie par un nombre borné de transitions impliquant A_1 : dans le cas contraire on aurait une exécution infinie avec une infinité d'actions de A_1 et donc A ne serait pas Σ-diagnosticable. Soit k le nombre maximum d'actions de A_1 qui peuvent survenir à partir d'un état fautif de $A_1 \times A_2$. Alors A est $(\Sigma, k+1)$-diagnosticable : sinon, il y aurait une exécution avec $k+1$ actions de A_1 après un état fautif. De plus, A n'est pas (Σ, k)-diagnosticable car il y a une exécution de $\mathcal{B}$ à partir d'un état

fautif suivie de k actions de A_1 et par construction de A_2, il y un exécution non fautive donnant la même observation (lemme 6.4.1). On peut donc conclure que le délai maximum après lequel une faute sera détectée est $k+1$. Soit $Max_d(A)$ la valeur de ce délai maximum. Calculer $Max_d(A)$ revient à trouver le nombre maximum d'actions de A_1 qui peuvent suivre un état fautif (atteignable) de $\mathcal{B}$. Ceci peut être fait en temps polynomial.

Une approche possible, consiste à chercher par dichotomie la borne maximale. Pour cela, on commence avec $k = |Q|^2$ (on peut montrer que si A est diagnosticable, alors il l'est pour $k \leq |Q|^2$, voir [YOO 02]). On construit donc un produit $A_1(k) \times A_2$ où $A_1(k)$ est défini comme suit : $A_1(k) = (Q \times \{0, k+1\}, (q_0, 0), \Sigma_\tau, \rightarrow_1)$ avec :

– $(q,n) \xrightarrow{l}_1 (q', n')$ si $q \xrightarrow{l} q'$ et $l \in \Sigma \cup \{\tau\}$;

– $(q,n) \xrightarrow{\tau}_1 (q', \min(n+1, k+1))$ si $q \xrightarrow{f} q'$.

On considère alors $\mathcal{B}(k) = A_1(k) \times A_2$ avec pour $\mathcal{B}(k)$ les ensembles d'états $R(k) = \emptyset$ et $F(k) = Q \times \{k+1\} \times Q$. Il s'ensuit que A est (Σ, k)-diagnosticable si et seulement si $\mathcal{L}^*(\mathcal{B}(k)) = \emptyset$. On peut tester l'accessibilité d'un état $q \in F$ en temps linéaire et donc vérifier que A est (Σ, k)-diagnosticable en temps polynomial[3]. Pour calculer le délai maximum, il suffit de tester (par dichotomie sur k) au maximum $\log |A|$ systèmes $A_1(k) \times A_2$. On a donc un algorithme polynomial pour le problème 6.3.2 :

THÉORÈME 6.4.2 *Le problème 6.3.2 peut être résolu en temps* $O(\log |A| \cdot |A|^4)$.

Une autre approche a été proposée dans [YOO 03] pour le calcul du délai maximum de détection des fautes. Ce calcul est réalisé sur un automate « à poids » construit à partir du produit $A_1 \times A_1 \times A_2$ et les auteurs montrent que le calcul du délai maximum peut être fait en $O(|Q|^3)$.

6.4.4. *Synthèse d'un diagnostiqueur*

Pour synthétiser un diagnostiqueur quand A est Σ-diagnosticable, on va déterminiser A_1. Pour cela on fait une *subset construction* classique. On note F l'ensemble des états fautifs de A_1 c'est-à-dire, $F = Q \times \{1\}$.

On définit $E_a(q)$ comme étant l'ensemble des états q' tels qu'il existe un chemin dans A_1 de q à q' constitué d'un nombre fini de τ transitions suivi de a :

$$E_a(q) = \{q'' \mid q \xrightarrow{\tau*} q' \xrightarrow{a} q''\}. \qquad \text{[E]}$$

3. La taille de $A_1(k) \times A_2$ est en $O(|A|^4)$.

On définit ensuite l'automate fini déterministe $\mathit{Det}(A_1) = (S, s_0, \Sigma, \delta', F', \emptyset)$ par :
– $S = 2^Q$;
– $s_0 = \{q' \in Q \times \{0,1\} \mid (q_0, 0) \xrightarrow{\tau*} q'\}$;
– pour $s \in S, a \in \Sigma, \delta'(s,a) = \cup_{x \in s} E_a(x)$;
– $F' = \{s \in S \mid s \subseteq F\}$.

$\mathit{Det}(A_1)$ est par construction déterministe. Soit w un mot de $\boldsymbol{\pi}_{/\Sigma}(\mathcal{L}(A))$, $\mathit{last}(w)$ est déterminé de façon unique dans $\mathit{Det}(A_1)$. Un $(\Sigma, \mathit{Max}_d(A))$-diagnostiqueur pour A peut être défini par : $D(w) = 1$ si $\mathit{last}(w) \in F'$ et 0 sinon. Ce diagnostiqueur détecte les fautes au plus tard $\mathit{Max}_d(A)$ pas discrets après leur apparition. Il est au pire de taille exponentielle en la taille de A.

THÉORÈME 6.4.3 *Le problème 6.3.3 peut être résolu en temps* $O(2^{|A|})$.

6.5. Diagnostic des systèmes temporisés

Dans cette section on montre comment décider si un système modélisé par un automate temporisé est diagnosticable ou non. En ce qui concerne la synthèse de diagnostiqueur, nous considérons les *ressources* (horloges) nécessaires pour diagnostiquer un système temporisé. En effet, une différence essentielle avec le cas non temporisé est la suivante : si un automate temporisé est diagnosticable, il ne l'est pas toujours à l'aide d'une machine du type automate temporisé. Ceci est donc radicalement différent du cas non temporisé, pour lequel, lorsqu'un système est diagnosticable, il existe un diagnostiqueur qui est un automate fini (même si sa taille est exponentielle en la taille du système de départ). Les résultats de cette section sont parus dans [TRI 02] en ce qui concerne les pararaphes 6.5.1 à 6.5.3, et dans [BOU 05, CHE 04] pour le paragraphe 6.5.4.

6.5.1. *Vérification de la diagnosticabilité*

Pour décider si A est diagnosticable, on construit comme dans le paragraphe 6.4.2 un produit synchronisé d'automates temporisés, $A_1 \times A_2$, tel que A_1 se comporte exactement comme A mais mémorise qu'une faute est survenue. A_2 se comporte comme A mais ne génère que les comportements non fautifs de A. Soit $A_1 = (L \times \{0,1\}, (\ell_0, 0), X, \Sigma_\tau, E_1, \mathit{Inv}_1)$ défini par :
– $((\ell, n), g, \lambda, r, (\ell', n)) \in E_1$ si $(\ell, g, \lambda, r, \ell') \in E, \lambda \in \Sigma \cup \{\tau\}$;
– $((\ell, n), g, \tau, r, (\ell', 1)) \in E_1$ si $(\ell, g, f, r, \ell') \in E$;
– $\mathit{Inv}_1((\ell, n)) = \mathit{Inv}(\ell)$;

et $A_2 = (L, \ell_0, X_2, \Sigma_\tau, E_2, \mathit{Inv}_2)$ avec :

– $X_2 = \{x_2 \mid x \in X\}$ (renommage des horloges de X) ;

– $(\ell, g_2, \lambda, r_2, \ell') \in E_2$ si $(\ell, g, \lambda, r, \ell') \in E, \lambda \in \Sigma \cup \{\tau\}$ avec : g_2 égale à g dans lequel on a remplacé les horloges x de g par leur renommage x_2, et r_2 égale à r avec le même remplacement ;

– $\mathit{Inv}_2(\ell) = \mathit{Inv}(\ell)$.

On considère maintenant $A_1 \times A_2$. Soit $\mathit{Runs}_{\geq\Delta}(A_1 \times A_2)$ les exécutions de $A_1 \times A_2$ passant par un état fautif de A_1 (donc du type $((\ell, 1), \ell')$), et telles que, au moins Δ unités de temps se sont écoulées après cet état. Les lemmes 6.4.1 et 6.4.2 sont encore valides dans le cas temporisé. Ils indiquent que, à toute exécution fautive ρ de $A_1 \times A_2$, correspondent deux exécutions, $\rho_{|1}$ et $\rho_{|2}$ de A, telles que $\boldsymbol{\pi}_{/\Sigma}(\mathit{tr}(\rho_{|1})) = \boldsymbol{\pi}_{/\Sigma}(\mathit{tr}(\rho_{|2}))$ (elles sont donc de même durée que ρ). Réciproquement, pour deux exécutions ρ_1 fautives de A et ρ_2 non fautives, de même durée, il existe une exécution ρ fautives de $A_1 \times A_2$, de même durée que ρ_1 et telle que $\rho_{|1} = \rho_1$ et $\rho_{|2} = \rho_2$.

Comme dans le cas non temporisé, on va construire un automate temporisé $\mathcal{D}$, qui est une version étendue de $A_1 \times A_2$. Dans le cas non temporisé, on devait s'assurer que $A_1 \times A_2$ pouvait générer des exécutions de durée (nombre de pas discrets de A_1) arbitrairement grande. Dans le cas temporisé, à cause des possibilités d'exécutions *zénons* ou de blocage du temps, plusieurs types de problèmes peuvent survenir par exemple :

– il existe des exécutions de A_2 (non fautives) qui bloquent le temps. Dans ce cas, l'équation [6.1] ne peut être satisfaite, mais ceci est artificiel : pour un Δ assez grand il n'y aura pas d'exécutions non fautives de même observation qu'une fautive, parce que le temps est bloqué dans A_2. On annoncerait donc que A est diagnosticable compte tenu de l'équation [6.1] alors que pratiquement ce n'est pas vrai ;

– il se peut aussi que A_1, après une faute, fasse converger le temps, et le bloque lui aussi. Il peut faire cela en tirant un nombre infini de transitions τ et dans ce cas créer une exécution zénon. Cette exécution zénon peut avoir la même observation qu'une exécution non fautive de A_2. Dans ce cas, on a une exécution avec un nombre infini de transitions discrètes de A_1. Cependant, d'un point de vue « physique », une telle exécution de A_1 n'est pas réaliste. Pour annoncer la non diagnosticabilité de A, il faut qu'il existe une exécution réalisable physiquement, c'est-à-dire, où le temps diverge comme l'impose l'équation [6.1].

Pour résoudre le problème de diagnostic dans le cas temporisé, on doit donc imposer deux conditions :

C_1 **:** dans A_2, toute exécution peut être prolongée en une exécution où le temps diverge. A_2 ne peut donc pas bloquer le temps. Pour assurer cette condition on peut vérifier que A_2 satisfait une propriété de *non-zenoness* (voir chapitre 4). On suppose dans la suite de cette section que cette condition est vérifiée.

C_2 **:** pour que le système ne soit pas diagnosticable, il faut que $A_1 \times A_2$ produisent des exécutions de durée arbitrairement grande, et pas seulement avec un nombre arbitrairement grand de transitions discrètes de A_1 (comme dans le cas non temporisé).

Pour C_2, on doit imposer que $A_1 \times A_2$ génère des exécutions où le temps ne converge pas c'est-à-dire non-zénons. Cette condition peut être imposée par la synchronisation de $A_1 \times A_2$ avec un automate temporisé *Div*(x) défini par *Div*$(x) = (\{0,1\}, 0, \{x\}, E, \mathit{Inv})$ donné sur la figure 6.2. Si l'on munit cet automate des ensembles $F = \emptyset$ et $R = \{1\}$, toute exécution ayant un nombre infini de pas discrets est *divergente*, c'est-à-dire que le temps progresse au-delà de n'importe quel entier donné. On suppose que l'horloge x de *Div*(x) n'est pas dans X.

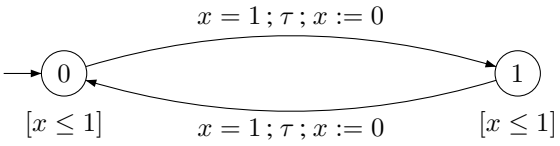


Figure 6.2. *L'automate Div*(x)

On définit formellement $\mathcal{D} = (A_1 \times A_2) \times \mathit{Div}(x)$ avec les ensembles $F_{\mathcal{D}} = \emptyset$ et $(((\ell,1),v),(\ell',v'),(1,\nu)) \in R_{\mathcal{D}}$ si $((\ell,1),v),(\ell',v'))$ est un état (fautif) de $A_1 \times A_2$ et $(1,\nu)$ est un état de *Div*(x). On peut prouver le théorème suivant qui est l'équivalent dans le cas temporisé du théorème 6.4.1 :

THÉORÈME 6.5.1 $\mathcal{L}^\omega(\mathcal{D}) \neq \emptyset \iff$ *A n'est pas* Σ*-diagnosticable.*

PREUVE 6.5.1
Preuve de $\Longrightarrow$*. Supposons que* $\mathcal{L}^\omega(\mathcal{D}) \neq \emptyset$*. Soit* $\rho \in \mathcal{L}^\omega(\mathcal{D})$*. A cause de la définition de* $R_{\mathcal{D}}$*,* ρ *est une exécution divergente :* $\forall \alpha \in \mathbb{N}$*, il existe un préfixe* $\rho[\alpha]$ *de* ρ *tel que Duration*$(\rho[\alpha]) \geq \alpha$*. En utilisant le lemme 6.4.1 (version automate temporisé), il existe donc* $\rho_{\alpha,1} = \rho[\alpha]_{|1}$ *et* $\rho_{\alpha,2} = \rho[\alpha]_{|2}$ *tels que* $\boldsymbol{\pi}_{/\Sigma}(tr(\rho_{\alpha,1})) = \boldsymbol{\pi}_{/\Sigma}(tr(\rho_{\alpha,2}))$ *et* $\rho_{\alpha,1} \in \mathit{Faulty}_{\geq\alpha}(A)$, $\rho_{\alpha,2} \in \mathit{NonFaulty}(A)$*. Pour tout* α*, l'équation* [6.3] *est satisfaite et donc A n'est pas* Σ*-diagnosticable.*

Preuve de $\Longleftarrow$*. Supposons que A n'est pas* Σ*-diagnosticable. Alors, par l'équation* [6.3]*, pour tout* $\alpha \in \mathbb{N}$*, il existe* $\rho_1 \in \mathit{Faulty}_{\geq\alpha}(A)$, $\rho_2 \in \mathit{NonFaulty}(A)$, $\boldsymbol{\pi}_{/\Sigma}(tr(\rho_1)) = \boldsymbol{\pi}_{/\Sigma}(tr(\rho_2))$*. Par le lemme 6.4.2, on a la propriété* (P) *suivante : pour tout* α*, il existe*[4] *une exécution* $\rho[\alpha] \in \mathit{Runs}_{\geq\alpha}((A_1 \times A_2) \times \mathit{Div}(x))$ *telle que* $\rho[\alpha]_{|1} = \rho_1$

4. *Div*(x) ne contraint pas les automates A_1 et A_2 et donc toute exécution de $A_1 \times A_2$ peut être entrelacée avec une exécution de *Div*(x) pour donner une exécution de $(A_1 \times A_2) \times \mathit{Div}(x)$.

et $\rho[\alpha]_{|2} = \rho_2$. Pour reproduire la preuve du théorème 6.4.1, on va utiliser une propriété du graphe des régions *introduit dans le chapitre 4. Soit RG($\mathcal{D}$) le graphe des régions de $\mathcal{D}$. Ce graphe est fini et a la propriété suivante : ρ est une exécution de RG($\mathcal{D}$) si et seulement s'il existe une* temporisation *de ρ, $\widetilde{\rho}$ qui est une exécution de $\mathcal{D}$. La propriété (P) ci-dessus, implique qu'il existe des exécutions fautives, de taille arbitrairement grande dans le graphe des régions RG($\mathcal{D}$). Ce graphe étant fini, par le lemme de König, on obtient qu'il existe une exécution infinie*[5] *dans RG($\mathcal{D}$) à partir d'un état fautif. A cause de la propriété du graphe des régions, il y a une temporisation de cette exécution infinie dans $\mathcal{D}$. Ainsi $\mathcal{L}^{\omega}(\mathcal{D}) \neq \emptyset$.* □

Décider si $\mathcal{L}(A) \neq \emptyset$ est PSPACE-complet [ALU 94a]. Décider la diagnosticabilité de A est donc dans PSPACE aussi.

On peut aussi réduire le problème de l'atteignabilité des automates temporisés à un problème de diagnosticabilité [TRI 02]. Soit A un automate temporisé sur l'alphabet Σ et *Fin* une localité particulière dont on veut tester l'accessibilité. On construit A' sur l'alphabet $\Sigma_{\tau,f}$ en ajoutant à A les transitions : $(\mathit{Fin}, \top, \lambda, \emptyset, \mathit{Fin})$ pour $\lambda \in \{\tau, f\}$. Il s'ensuit que : A' n'est pas Σ-diagnosticable si et seulement si *Fin* est atteignable dans A. On obtient donc :

THÉORÈME 6.5.2 ([TRI 02]) *Le problème 6.3.1 est PSPACE-complet pour les automates temporisés.*

EXEMPLE 6.5.1 *Pour l'automate $\mathcal{A}(\alpha)$ de la figure 6.1, le produit $\mathcal{D}$ est donné sur la figure 6.3. On a pris l'horloge y pour Div(y). L'automate $\mathcal{A}(\alpha)$ n'est pas diagnosticable si et seulement si il n'y a pas d'exécution atteignant R_1 : en effet dans ce cas il y a une exécution fautive divergente. Comme dans tous les états $x = x_2$, on peut en déduire que R_1 est atteignable si et seulement si $\alpha < 3$.*

6.5.2. *Calcul du délai maximal de détection des fautes*

Pour calculer le délai maximum, on peut procéder comme dans le cas non temporisé, par dichotomie. On suppose que l'on a vérifié que A était Σ-diagnosticable. On teste si A est (Σ, Δ)-diagnosticable en commençant avec $\Delta = 1 = 2^0$, puis avec $\Delta = 2^k$, en augmentant k, jusqu'à ce que A soit $(\Sigma, 2^k)$-diagnosticable (ce qui arrivera forcément si l'on a vérifié auparavant que A est Σ-diagnosticable). Soit k_0 le plus petit k tel que A soit $(\Sigma, 2^{k_0})$-diagnosticable. On va ensuite chercher la valeur du délai maximum dans l'intervalle $[2^{k_0-1} + 1, 2^{k_0}]$ par recherche dichotomique.

5. Le raisonnement est le même que dans la preuve du théorème 6.4.1.

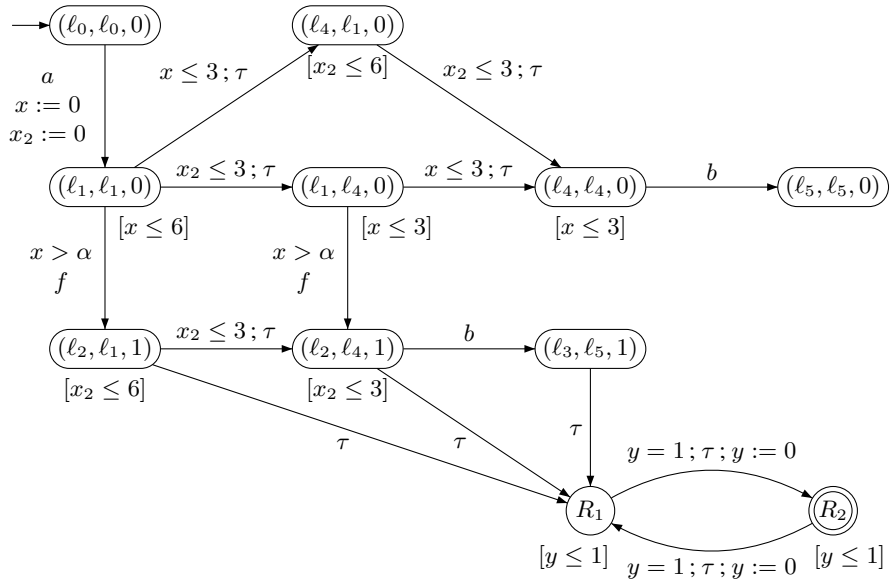


Figure 6.3. *Le produit $\mathcal{D}$ pour l'automate $\mathcal{A}(\alpha)$ de la figure 6.1*

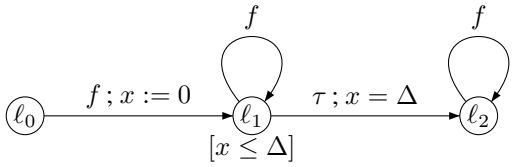


Figure 6.4. *L'automate Obs*(Δ)

Pour vérifier que A est (Σ, Δ)-diagnosticable on construit le produit $G = (A'_1 \times A_2) \times \mathit{Obs}(\Delta)$ où $\mathit{Obs}(\Delta)$ est donné sur la figure 6.4. Cet automate accepte les exécutions Δ-fautives (on omet sur la figure les transitions sur l'alphabet Σ qui bouclent sur les localités ℓ_i). On suppose aussi que l'horloge x de $\mathit{Obs}(\Delta)$ n'est pas une horloge de A. A'_1 est le même que A_1 mais on ne renomme pas l'étiquette f dans les transitions fautives, afin de la synchroniser avec elle de $\mathit{Obs}(\Delta)$. L'automate G a pour états « final » les états où la localité de *Obs* est ℓ_2. Si un tel état est atteignable, alors il y a une exécution dans G, qui est fautive et Δ unités de temps se sont écoulées après la faute. Par la définition de G, A n'est pas (Σ, Δ)-diagnosticable dans ce cas. Tester la (Σ, Δ)-diagnosticabilité de A revient donc à tester si un état final de G est accessible (ce qui est PSPACE, voir chapitre 4). On peut donc deviner k_0 et tester si A est

$(\Sigma, k_0 + 1)$-diagnosticable et pas (Σ, k_0)-diagnosticable. Comme NPSPACE = PSPACE on obtient :

THÉORÈME 6.5.3 *Pour les automates temporisés, le problème 6.3.2 est dans PSPACE.*

REMARQUE 6.5.1 *Pour calculer le délai maximum, on peut faire le même raisonnement que dans le cas non temporisé et trouver une borne k telle A est (Σ, k)-diagnosticable si et seulement si A est Σ-diagnosticable. En effet, A est Σ-diagnosticable si et seulement si le graphe des régions de $\mathcal{D}$ ne comporte pas de cycles d'états fautifs (où le temps diverge). Comme la taille du graphe des régions de $\mathcal{D}$ dans la partie fautive est (voir [ALU 94a]) de m=$|L| \cdot \big(|X|! \cdot 2^{|X|} \cdot \Pi_{x \in X}(2 \max_x + 2)\big)$, avec $\max_x$ la valeur absolue de la constante maximale à laquelle x est comparée dans A, et que le temps qui peut s'écouler dans chaque région* fautive *est au maximum de 1 unité de temps, il s'ensuit que A est Σ-diagnosticable si et seulement si A est (Σ, m)-diagnosticable. On peut donc tester la diagnosticabilité en vérifiant que A est (Σ, m)-diagnosticable. La diagnosticabilité est donc réduite à un problème d'accessibilité dans les automates temporisés. Une autre conséquence est que le problème de déterminer si A est (Σ, k)-diagnosticable est lui aussi PSPACE-complet pour les automates temporisés.*

6.5.3. *Synthèse d'un diagnostiqueur*

Pour définir un diagnostiqueur si A est Σ-diagnosticable, on va définir une fonction qui estime l'ensemble des états possibles de A. En effet, on ne peut pas reproduire la construction du cas non temporisé (déterminisation de A) car les automates temporisés ne sont pas (toujours) déterminisables [ALU 94a]. En plus, tester si un automate temporisé donné est déterminisable est indécidable [FIN 05, TRI 06].

Pour les classes d'automates temporisés déterminisables (par exemple [ALU 94b]), on peut faire la même construction que dans le cas non temporisé.

Cependant, il existe des automates temporisés non déterministes pour lesquels il n'y a pas de diagnostiqueur qui peut être représenté par un automate temporisé déterministe. Soit C l'automate temporisé de la figure 6.5 (cet automate est extrait de [BOU 05]). Les exécutions fautives de C sont celles de la forme $\delta.a.t$ où $\delta \in \mathbb{N}$ et $t \geq 0$, et les exécutions non fautives sont de la forme $\delta.a.t$ où $\delta \notin \mathbb{N}$. L'équation [6.2] est donc vérifiée pour tout délai Δ. Mais il n'existe pas d'automate temporisé déterministe acceptant le langage $\delta.a$ où $\delta \in \mathbb{N}$ (voir [BER 98]). On peut pourtant construire un 0-diagnostiqueur pour C qui est la fonction $D : TW^*(\{a\}) \rightarrow \{0, 1\}$ définie par $D(\delta.a.t) = 1$ si $\delta \in \mathbb{N}$ et $D(\delta.a.t) = 0$ sinon.

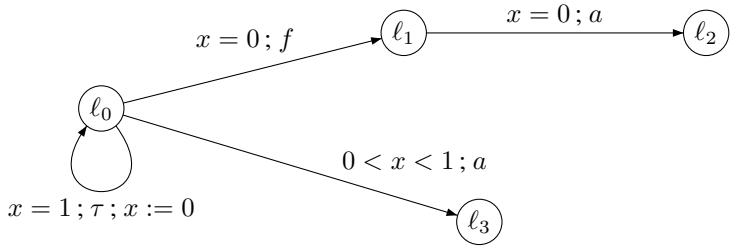


Figure 6.5. *L'automate C*

Pour construire un diagnostiqueur dans le cas général (automates temporisés non déterministes) on procède comme suit. Dans un premier temps, on suppose que l'on a ajouté aux états de A un bit d'information qui indique si une faute est survenue ou pas. Ceci est facile à faire et revient à construire A_1 défini précédemment. Dans A, on a donc deux types d'états : les états fautifs où le bit est 1 et les états non fautifs où il est à 0. On va construire une fonction D qui va estimer l'état de A après avoir « lu » un mot temporisé. On va ensuite mettre à jour cette estimation après chaque occurrence d'un événement temporisé du type (δ, a). Ceci revient à déterminiser *en ligne* A pour un mot temporisé donné. Si l'ensemble des états possibles de A après un mot temporisé w est fautif, alors on annonce une faute. S'il y a des états fautifs et non fautifs on ne peut rien dire.

Formellement la construction d'un tel diagnostiqueur est détaillé dans [TRI 02]. Un diagnostiqueur dans le cas général est un algorithme (une machine de Turing) qui va calculer l'ensemble des états possibles de A après un mot temporisé w. En prenant un algorithme qui détermine l'ensemble des régions dans lequel l'automate A peut se trouver après un mot w, on obtient un algorithme exponentiel dans la taille de A et de l'observation w.

Cette grande complexité de calcul peut être prohibitive surtout si l'on doit faire le calcul *en ligne*. On peut donc s'intéresser au problème de diagnosticabilité à l'aide d'automates temporisés déterministes (puisqu'un diagnostiqueur est une fonction déterministe).

6.5.4. *Diagnostic à l'aide d'automates temporisés déterministes*

Le problème de diagnostic à l'aide d'automates temporisés déterministes (ATD) a été proposé et étudié dans [BOU 05, CHE 04]. Un problème similaire a été étudié dans [KRI 04b] pour la synthèse de *testeurs temporisés* sous forme d'automates temporisés déterministes. Un automate temporisé est déterministe s'il ne contient pas de transitions étiquetées par τ, et si pour toute transition (ℓ, g, a, r, ℓ') et $(\ell, g', a, r', \ell'')$,

$g \wedge g' \equiv \bot$. Il est complet si de toute localité et toute action il existe une garde qui est satisfaite. Dans la suite on note $Inv_\top$ la fonction qui a toute localité associe l'invariant $\top$. Soit $\mathcal{C}$ une classe d'automates temporisés déterministes. On note ATD la classe de tous les automates temporisés déterministes.

DÉFINITION 6.5.1 ($\mathcal{C}$-DIAGNOSTIQUEUR) *Soit A un automate temporisé sur $\Sigma_{\tau,f}$ et $\Delta \in \mathbb{N}$. Un (Σ, Δ)-$\mathcal{C}$-diagnostiqueur pour A est un automate temporisé déterministe complet $\Theta = (N, n_0, C, \Sigma, E_\Theta, Inv_\top, F_\Theta, \emptyset)$ de la classe $\mathcal{C}$ tel que :*

– pour tout $\rho \in NonFaulty(A)$, $last(\boldsymbol{\pi}_{/\Sigma}(tr(\rho))) \notin F_\Theta$;

– pour tout $\rho \in Faulty_{\geq\Delta}(A)$, $last(\boldsymbol{\pi}_{/\Sigma}(tr(\rho))) \in F_\Theta$. □

Autrement dit, Θ accepte les mots Δ-fautifs, mais pas les mots non fautifs. A est (Σ, Δ)-$\mathcal{C}$-diagnosticable s'il existe un (Σ, Δ)-$\mathcal{C}$-diagnostiqueur pour A. A est Σ-$\mathcal{C}$-diagnosticable s'il existe un $\Delta \in \mathbb{N}$ tel que A est (Σ, Δ)-$\mathcal{C}$-diagnosticable.

On peut formuler le problème de diagnosticabilité par automate temporisé déterministe de la classe $\mathcal{C}$ de la manière suivante :

PROBLÈME 6.5.1 ($\mathcal{C}$-DIAGNOSTICABILITÉ)
ENTRÉES *: Un automate temporisé $A = (L, \ell_0, X, \Sigma_{\tau,f}, E, Inv)$.*
PROBLÈME *: Est-ce que A est Σ-$\mathcal{C}$-diagnosticable ?*

Dans ce problème on cherche un délai Δ et un automate temporisé déterministe qui accepte un langage correspondant aux mots Δ-fautifs.

EXEMPLE 6.5.2 *L'automate $\mathcal{A}(3)$ de la figure 6.1 est 3-diagnosticable et on peut trouver un automate temporisé qui va détecter les fautes en maximum 3 unités de temps. Un exemple de diagnostiqueur est l'automate de la figure 6.6. Le diagnostic avec cet automate temporisé déterministe est réalisé de la manière suivante : une localité particulière (ici 3) est la localité où l'on annonce les fautes. Cet automate est dans 3 si et seulement si une faute est survenue. L'automate effectue des transitions à chaque occurrence d'événements (ici a et b) de manière déterministe. Lorsque l'automate entre dans la localité 3 on annonce une faute. $\mathcal{A}(3)$ est donc 3-ATD-diagnosticable.*

Le problème 6.5.1 dans sa version générale est actuellement toujours ouvert. Une solution a été proposée dans [BOU 05] dans le cas où les *ressources* de l'automate temporisé diagnostiqueur sont fixées et où la borne Δ dans laquelle on doit annoncer les fautes est donnée. Les ressources d'un automate temporisé sont : le nombre

d'horloges, les constantes auxquelles peuvent être comparées les horloges ; ceci comprend en particulier la constante maximale à laquelle on peut comparer une horloge et aussi la finesse ou le grain d'observation. Par exemple, les ressources d'un automate temporisé peuvent être $(\{x\}, 2, \frac{1}{3})$ qui indiquent que :

– une seule horloge, x, est disponible pour mesurer le temps ;

– la constante maximale à laquelle on peut comparer x est 2 ;

– on ne peut comparer x qu'à des multiples de $\frac{1}{3}$.

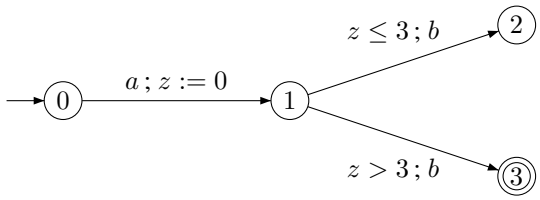


Figure 6.6. *Un diagnostiqueur ATD pour* $\mathcal{A}(3)$

EXEMPLE 6.5.3 *L'automate* $\mathcal{A}(3)$ *est diagnosticable avec la ressource* $(\{z\}, 3, 1)$.

Soit $\mu = (Y, \max, \frac{1}{m})$ une ressource, avec Y un ensemble fini d'horloges (distinctes de X), $\max \in \mathbb{N}$ et $m \in \mathbb{N}^*$. Un automate temporisé *de ressource* μ est un automate qui utilise des horloges de Y, comparées à des constantes dont la valeur maximale est au plus $\max$, et tel que toutes les valeurs utilisées dans les gardes et invariants sont des multiples de $\frac{1}{m}$. On note ATD_μ la classe des automates temporisés déterministes de ressource μ. Dans la suite on utilise le terme de « régions de μ » pour désigner à la fois les régions engendrées par μ et les contraintes sur les horloges qui les définissent. Le problème de diagnosticabilité par ATD avec ressource fixée μ est le suivant :

PROBLÈME 6.5.2 (ATD_μ-DIAGNOSTICABILITÉ)
ENTRÉES *: Un automate temporisé* $A = (L, \ell_0, X, \Sigma_{\tau,f}, E, \mathit{Inv})$ *et une ressource* μ.
PROBLÈME *: Est-ce que* A *est* Σ*-ATD*$_\mu$*-diagnosticable ?*

Le problème précédent est lui aussi ouvert car la borne Δ dans laquelle on doit annoncer les fautes n'est pas spécifiée. Un problème plus simple est :

PROBLÈME 6.5.3 (Δ-ATD_μ-DIAGNOSTICABILITÉ)
ENTRÉES *: Un automate* $A = (L, \ell_0, X, \Sigma_{\tau,f}, E, \mathit{Inv})$ *et* $\Delta \in \mathbb{N}$.
PROBLÈME *: Est-ce que* A *est* (Σ, Δ)*-ATD*$_\mu$*-diagnosticable ?*

Une solution à ce problème a été proposée dans [BOU 05]. Avant de donner les principes de cette solution, on s'intéresse au problème suivant :

PROBLÈME 6.5.4 (VÉRIFICATION DE LA ATD-DIAGNOSTICABILITÉ)
ENTRÉES *: Un automate temporisé* $A = (L, \ell_0, X, \Sigma_{\tau,f}, E, Inv)$ *et un ATD* $\Theta = (N, n_0, C, \Sigma, E_\Theta, Inv_\top, F_\Theta, \emptyset)$ *et* $\Delta \in \mathbb{N}$.
PROBLÈME *: Est-ce que* Θ *est un* (Σ, Δ)*- diagnostiqueur pour* A *?*

Pour résoudre le problème précédent, on considère les automates A'_1, $Obs(\Delta)$ du paragraphe 6.5.2. On note $A(\Delta) = A'_1 \times Obs(\Delta)$. Pour $A(\Delta)$ on peut distinguer les localités Δ-fautives, $L_{\Delta f}$, du type $((\ell, 1), \ell_2))$ et les localités non fautives, $L_{\neg f}$, du type $((\ell, 0), \ell')$. Une exécution de A est Δ-fautive si et seulement si dans $A(\Delta)$ elle conduit[6] dans une localité de $L_{\Delta f}$; inversement, elle est non fautive si et seulement si elle termine dans une localité de $L_{\neg f}$.

Vérifier que Θ est un (Σ, Δ)-diagnostiqueur pour A revient donc à vérifier que pour tout état atteignable $((q, \ell), v)$ de $A(\Delta) \times \Theta$:

1) $q \in L_{\Delta f} \implies \ell \in F_\Theta$;

2) $\ell \in F_\Theta \implies q \notin L_{\neg f}$.

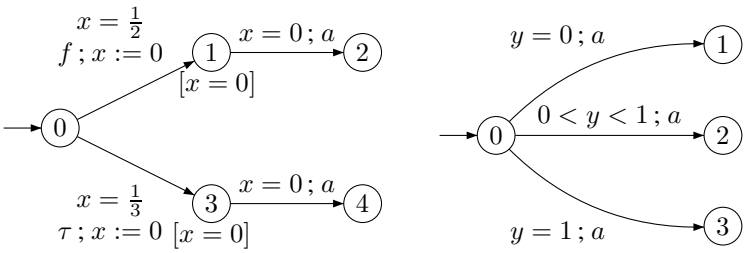
Ceci peut être fait à l'aide du graphe des régions de $A(\Delta) \times \Theta$ (et donc en espace polynomial).

On peut considérer une version un peu plus libre du problème 6.5.4 : soit un ATD $\Theta^- = (N, n_0, C, \Sigma, E_\Theta, Inv_\top)$ dans lequel on n'a pas fixé l'ensemble des localités acceptantes. Existe-t-il $F_\Theta \subseteq N$ tel que A est (Σ, Δ)-diagnosticable avec $\Theta = (N, n_0, C, \Sigma, E_\Theta, Inv_\top, F_\Theta, \emptyset)$?

EXEMPLE 6.5.4 *Comme on l'a vu précédemment, l'automate de la figure 6.6 peut être configuré en diagnostiqueur pour* $\mathcal{A}(3)$ *en désignant la localité* 3 *comme fautive. Dans l'exemple de la figure 6.7, on veut diagnostiquer les fautes produites par l'automate* J *de la figure 6.7(a). L'automate* O^- *de la figure 6.7(b) ne peut pas être configuré pour diagnostiquer* J*. Il ne peut pas faire la différence entre les exécutions* $\delta.a$ *avec* $0 < \delta < 1$ *et donc ne peut distinguer les exécutions fautives des non fautives dans* J*. Il faut remarquer que* J *est diagnosticable au sens de la définition 6.3.1 et même il existe un ATD qui diagnostique* J *: il suffit de prendre un automate de ressource* $(\{y\}, 1, \frac{1}{2})$.

6. D'après la définition de $Obs(\Delta)$, il se peut que $Obs(\Delta)$ soit dans ℓ_1 Δ unités de temps après une faute. Cependant, à cause de l'invariant on va forcément atteindre ℓ_2.

Comme le montre l'exemple précédent, pour savoir si l'on peut configurer un automate pour être un diagnostiqueur il faut vérifier qu'il n'est pas trop « grossier » : il doit accepter toutes les exécutions fautives mais cela ne doit pas impliquer qu'il accepte une exécution non fautive de même trace.



(a) Automate J (b) Automate O^-

Figure 6.7. *Les automates J et O^-*

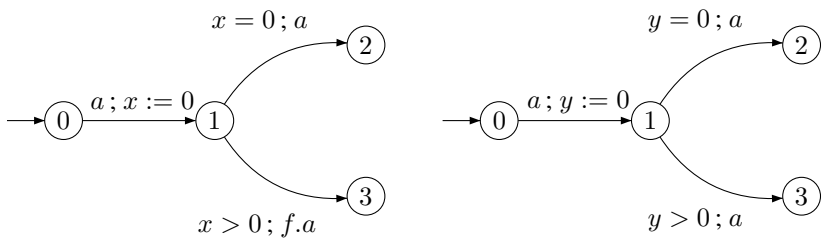
Etant donnée une exécution ρ d'un ATD H telle que :

$$\rho = (n_0, \mathbf{0}) \xrightarrow{\delta_0} (n_0, v'_0) \xrightarrow{a_1} (n_1, v_1) \cdots (n_{k_1}, v'_{k-1}) \xrightarrow{a_k} (n_k, v_k) \xrightarrow{\delta_k} (n_k, v'_k),$$

on note *Symbtr*(ρ) la *trace symbolique* associée à ρ : *Symbtr*(ρ) est la[7] séquence (g_i, a_i, R_i) d'étiquettes symboliques des transitions franchies lors de l'exécution ρ dans H. Soit *Det*(*RG*(H)) l'automate obtenu en déterminisant le graphe des régions *RG*(H) (c'est-à-dire où l'on a interprété les transitions τ d'écoulement du temps et les transition (g, τ, R) de *RG*(H) comme non visibles). Par définition du graphe des régions *RG*(H), si ρ est une exécution de H alors *Symbtr*(ρ) est une exécution possible de l'automate *Det*(*RG*(H)). Réciproquement, pour toute trace w de *Det*(*RG*(H)), il existe un mot temporisé v qui est accepté par H et tel que *Symbtr*$(v) = w$. En fait l'automate H ne peut pas faire la différence (en termes de localité atteignable) entre deux mots temporisés v_1 et v_2 qui ont même trace symbolique.

EXEMPLE 6.5.5 *On considère l'automate de la figure 6.8(a) tiré de [BOU 05] et le candidat automate diagnostiqueur Q de la figure 6.8(b). La transition $f.a$ de l'automate P indique que f est immédiatement suivie de a. L'automate Q peut être configuré pour diagnostiquer P : il suffit de déclarer la localité 3 comme finale. L'automate Q est de ressource $(\{y\}, 0, 1)$. Q est donc un 0-diagnostiqueur pour P.*

7. Il n'y a qu'une séquence possible puisque l'on considère des automates déterministes.



(a) L'automate P (b) L'automate Q

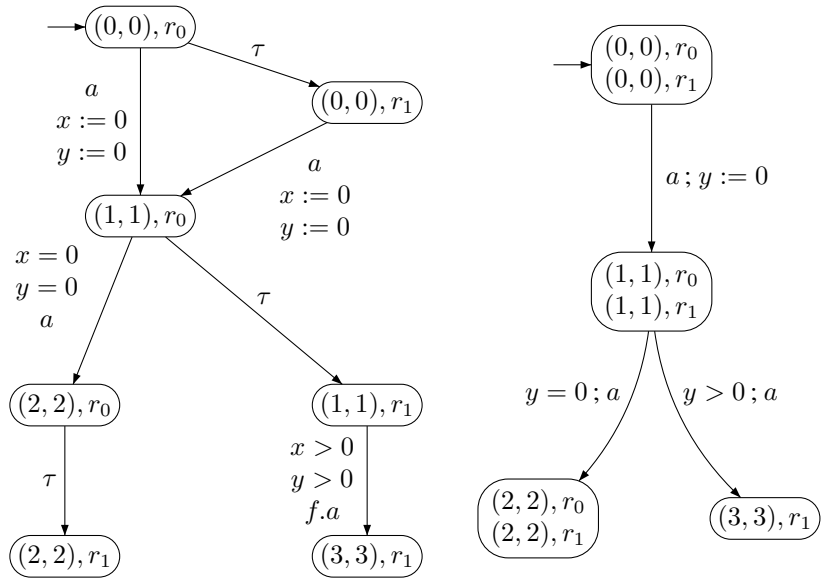
Figure 6.8. *Les automate P et Q*

Formellement, on pourra configurer une automate Θ^- pour diagnostiquer un automate temporisé A si et seulement si la condition suivante, (K), est satisfaite : il n'y a pas deux exécutions $\rho_1 \in \mathit{Faulty}_{\geq\Delta}(A)$, $\rho_2 \in \mathit{NonFaulty}(A)$, telle que $\boldsymbol{\pi}_{/\Sigma}(\mathit{tr}(\rho_1))$ et $\boldsymbol{\pi}_{/\Sigma}(\mathit{tr}(\rho_2))$ produisent les mêmes traces symboliques quand elles sont « lues » par Θ^-.

Pour vérifier (K), on procède de la manière suivante : soit $RG_1 = RG(A(\Delta) \times \Theta^-)$ le graphe des régions de $A(\Delta)\times\Theta^-$. On va « projeter » les étiquettes de ce graphe des régions sur Θ^- : chaque étiquette différente de τ et du type $(g_1 \wedge g_2, a, R_1 \cup R_2)$ où (g_2, a, R_2) est l'étiquette symbolique de Θ^-. La projection consiste à remplacer toutes les étiquettes $(g_1 \wedge g_2, a, R_1 \cup R_2)$ par (g_2, a, R_2). Soit RG_2 le graphe ainsi obtenu. La phase suivante consiste à déterminiser RG_2 pour obtenir $\mathit{Det}(RG_2)$ (les transitions τ et (g, τ, R) sont considérées comme non visibles). Dans $\mathit{Det}(RG_2)$ les états sont du type $\{((q_1, \ell), r_1), \cdots, ((q_n, \ell), r_n)\}$ car Θ^- est déterministe. On peut donc les mettre sous la forme équivalente (S, ℓ) avec $S = \{(q_1, r_1), \cdots, (q_n, r_n)\}$ et chaque q_i est une localité de L et chaque r_i une région de RG_1. On note $\mathit{Bad}(\ell)$ l'ensemble des états $(\{(q_1, r_1), \cdots, (q_n, r_n)\}, \ell)$ tels qu'il existe i, j avec $q_i \in L_{\Delta f}$ et $q_j \in L_{\neg f}$.

EXEMPLE 6.5.6 *Pour les automates P et Q de la figure 6.8, les graphes RG_1 et RG_2 sont données sur la figure 6.9. On considère que $\Delta = 0$ et donc $P(\Delta) = P$ et la localité Δ-fautive de P est* 3. *Dans les graphes des régions, r_0 est la région correspondant à $x = y = 0$ et r_1 à $x = y > 0$. Il n'y a pas de localité ℓ de Q tel $\mathit{Bad}(\ell) \neq \emptyset$ car seule $(3, 3)$ est Δ-fautive.*

Vérifier qu'un F_Θ existe pour configurer Θ^- est donc équivalent à : existe-t-il une localité $\ell \in N$ telle que $\mathit{Bad}(\ell) \neq \emptyset$? Si la réponse est « oui » alors on ne peut pas partitionner N pour que Θ^- soit un diagnostiqueur pour A. Si la réponse est « non » il suffit de mettre dans F_Θ les localités ℓ telles que, dans le graphe des régions, (S, ℓ)



(a) RG_1 (b) RG_2

Figure 6.9. *Les graphes des régions RG_1 et RG_2*

contient un q_i Δ-fautif. La complexité de cette procédure est doublement exponentielle en la taille de A (une exponentielle pour le graphe des régions et une pour la déterminisation).

EXEMPLE 6.5.7 *Pour l'exemple de Q, on désigne donc la localité 3 comme acceptante et on a configuré Q en diagnostiqueur pour P.*

Pour résoudre le problème 6.5.4 la solution proposée dans [BOU 05] repose sur une construction du type exposée ci-dessus. Le résultat est le théorème suivant :

THÉORÈME 6.5.4 ([BOU 05]) *Le problème 6.5.3 est 2EXPTIME-complet pour la classe ATD.*

PREUVE 6.5.2 *La preuve d'appartenance à la classe 2EXPTIME est faite en réduisant le problème 6.5.3 à un problème de jeu de sûreté à deux joueurs. La preuve de 2EXPTIME difficile est obtenue par réduction du problème de l'acceptation d'une*

machine de Turing alternante d'espace exponentiel au problème 6.5.3. On donne ici l'idée de la preuve d'appartenance à 2EXPTIME. Pour le détail et la preuve complète on peut se référer à [CHE 04].

Soit donc un automate temporisé $A = (L, \ell_0, X, \Sigma_{\tau,f}, E, Inv)$ *et les automates* A'_1, *et Obs*(Δ) *définis dans la section 6.5.2. On note encore* $A(\Delta)$ *le produit* $A'_1 \times Obs(\Delta)$. *Comme on ne connaît pas la structure discrète de l'automate temporisé qui permettrait (s'il existait) de diagnostiquer* A, *on va prendre un automate le plus puissant possible étant donné la ressource qu'on a fixée.*

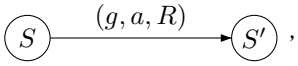
Etant donnée une ressource $\mu = (Y, \max, \frac{1}{m})$ *(*$X \cap Y = \emptyset$*), une garde* minimale *pour* μ *est une garde qui définit une région de* μ. *On définit l'*automate universel $\mathcal{U} = (\{0\}, \{0\}, Y, \Sigma, E_\mu, Inv_\mu)$ *par :*

– $Inv_\mu(0) = \top$,

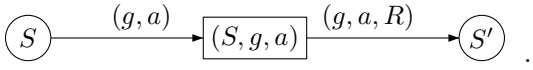
– $(0, g, a, R, 0) \in E_\mu$ *pour tout* (g, a, R) *tel que* $a \in \Sigma$, $R \subseteq Y$, *et* g *est une garde* minimale *de* μ.

$\mathcal{U}$ *est bien fini car l'ensemble* E_μ *est fini. Cependant* $\mathcal{U}$ *n'est pas déterministe car il peut choisir de remettre à zéro plusieurs sous-ensembles de* Y *pour un même couple* (g, a). *Pour diagnostiquer* A *il faut trouver quand remettre à zéro des horloges pour avoir assez d'information temporelle et ne pas interpréter de la même manière deux exécutions de* A, *une* Δ*-fautive et une non fautive.*

Pour cela on définit l'automate $A(\Delta) \times \mathcal{U}$. *On construit ensuite le graphe des régions RG*$(A(\Delta) \times \mathcal{U})$ *que l'on projette comme précédemment sur* $\mathcal{U}$. *Pour finir, on détermine pour obtenir l'automate* $H_{A,\Delta,\mu}$. *Les états mauvais de* $H_{A,\Delta,\mu}$ *sont les états dans lesquels il y a une localité* Δ*-fautive et une non fautive. On note Bad*$(H_{A,\Delta,\mu})$ *ces états. On va construire un jeu à partir de* $H_{A,\Delta,\mu}$ *en le transformant de la manière suivante : pour toute transition de* $H_{A,\Delta,\mu}$ *du type :*



on crée deux transitions consécutives :



On obtient ainsi le jeu $G_{A,\Delta,\mu}$ *en partitionnant les états en : les états du joueur* 1 *(ronds) du type* S *où c'est lui qui choisit une transition ; les états du joueur* 2 *(carrés), du type* (S, g, a) *où ce joueur choisit une transition. Les états Bad de* $G_{A,\Delta,\mu}$ *sont les états de Bad*$(H_{A,\Delta,\mu})$. *On a donc un jeu à tour, à deux joueurs. On fixe l'objectif*

du joueur 2 *à : « éviter les états Bad ». On peut prouver que : le joueur* 2 *a une* stratégie *pour gagner si et seulement si l'on peut diagnostiquer* A *avec un ATD de ATD*$_\mu$*. Comme le jeu* $G_{A,\Delta,\mu}$ *est un jeu fini à tour, si le joueur* 2 *peut gagner il peut le faire avec une stratégie* positionnelle *: le choix de ses actions ne dépend que de l'état (carré) dans lequel le jeu se trouve. Il y a même une stratégie* la plus permissive *positionnelle : cette stratégie donne l'ensemble des actions que l'on peut jouer pour gagner. La procédure ci-dessus est donc constructive et elle permet : (1) de décider s'il existe un automate de ressource* μ *pour diagnostiquer* A *et (2) dans le cas où la réponse est « oui », de calculer un tel automate.*

L'algorithme ci-dessus est dans 2EXPTIME car (i) *le jeu* $G_{A,\Delta,\mu}$ *est de taille doublement exponentielle en la taille de* A *et* (ii) *résoudre un jeu de ce type peut être fait en temps linéaire en la taille du jeu.* □

EXEMPLE 6.5.8 *La construction ci-dessus est illustrée dans par la figure 6.10. L'automate à diagnostiquer est* P *de la figure 6.8. On veut trouver un ATD de ressource* $\mu = (\{y\}, 0, 1)$ *qui diagnostique les fautes en* $\Delta = 0$ *unité de temps. Comme on l'a vu précédemment, pour diagnostiquer* A *il suffit de prendre l'automate* Q *qui va remettre* y *à* 0 *quand (le premier)* a *survient ; ensuite si le deuxième* a *intervient quand* $y > 0$ *on annonce une faute. Le jeu* $G_{P,0,\mu}$ *obtenu par la procédure décrite précédemment est donné sur la figure 6.10. Dans ce graphe,* r_0 *correspond à la région* $x = y = 0$*,* r_1 *à* $x = y > 0$*,* r_2 *à* $y = 0 \wedge x > 0$ *et* r_3 *à* $x = 0 \wedge y > 0$*.*

Les états à éviter pour le joueur carré sont ceux dans un double trait qui contiennent un état de faute 3 *et un état non fautif* 2*. Pour gagner, le joueur carré doit donc éviter d'aller dans* $(1, r_2)$ *et remettre* y *à zéro quand* a *survient à partir de* $\boxed{2}$*. Les choix gagnants du joueur carré sont en traits pleins et les perdants en pointillés. Les états terminaux où l'on annonce les fautes sont* $(3, r_2)$ *et* $(3, r_1)$ *: dans l'état du jeu* $\boxed{4}$ *on peut remettre* y *à zéro ou pas, dans tous les cas on annonce une faute.*

Une autre classe intéressante d'automates temporisés est celle des *Event Recording Automata* (ERA) introduite dans [ALU 94b]. La classe ERA impose que les horloges soient associées à des événements. Ainsi, à tout événement $a \in \Sigma$ correspond une horloges x_a qui est remise à zéro uniquement lorsque a survient. La classe ERA a de bonnes propriétés en particulier, tout automate A de ERA peut être déterminisé en *Det*(A) acceptant le même langage que A. Un autre résultat de [BOU 05] concerne la diagnosticabilité par automate ERA et montre que le problème de synthèse d'un diagnostiqueur du type ERA est moins coûteux :

THÉORÈME 6.5.5 ([BOU 05]) *Le problème 6.5.3 est PSPACE-complet pour la classe ERA.*

Comme le problème de diagnosticabilité est lui-même PSPACE-complet, la complexité précédente est *optimale* pour un problème de synthèse de diagnostiqueurs.

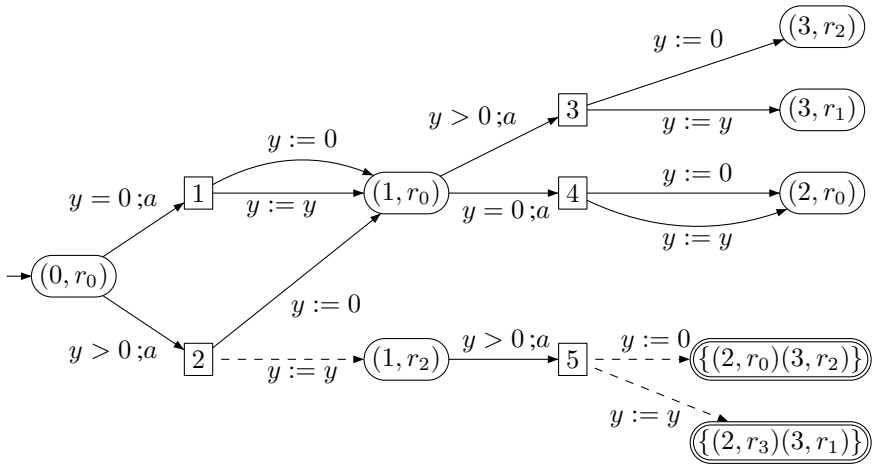


Figure 6.10. *Le jeu $G_{P,0,\mu}$*

6.6. Extensions et problèmes ouverts

Les résultats précédents s'étendent au cas de *fautes multiples*. Dans ce cas on désire diagnostiquer plusieurs fautes $f_1, f_2, \cdots, f_k$. Il suffit donc de considérer les problèmes de diagnostic de chaque faute f_i (et les autres fautes $f_j, j \neq i$ sont remplacées par τ).

Le diagnostic de fautes est un problème proche du test de conformité. Les relations entre ces deux problèmes ont fait l'objet des articles [KRI 04a, KRI 04b].

On peut aussi considérer le problème de diagnostic temporisé avec des *horloges digitales* comme dans [ALT 06, JIA 06]. Dans cette approche, on considère que l'on a un automate temporisé A à diagnostiquer, un autre automate temporisé (pas forcément déterministe) *Clock* sur l'alphabet $\{tick\}$ où *tick* n'est pas un événement de A. On peut donc poser le problème de diagnostic avec horloges digitales de la façon suivante : le système produit des mots temporisés ρ qui sont des entrelacements de mots temporisés de A et *Clock*, mais le diagnostiqueur ne peut observer que le mot $Unt(\rho)$: ses indications quant aux informations temporelles quantitatives proviennent uniquement des événements *tick*. Décider si $A \times Clock$ est diagnosticable est dans PSPACE [ALT 06].

Les problèmes ouverts concernant le diagnostic temporisé sont :

– la diagnosticabilité sans ressource fixée ou avec ressource fixée mais sans borne Δ sur le temps nécessaire pour diagnostiquer ; ceci correspond aux problèmes 6.5.1 et 6.5.2 ;

– dans le cas du diagnostic à l'aide d'horloges digitales [ALT 06], le problème de l'existence d'une horloge digitale, *Clock*, tel qu'un système soit diagnosticable avec cette horloge.

6.7. Bibliographie

[ALT 06] ALTISEN K., CASSEZ F., TRIPAKIS S., « Monitoring and Fault-Diagnosis with Digital Clocks », *Proceedings of the 6th International Conference on Application of Concurrency to System Design (ACSD'06)*, p. 101-110, IEEE Computer Society, 2006.

[ALU 94a] ALUR R., DILL D., « A theory of timed automata », *Theoretical Computer Science*, vol. 126, p. 183-235, 1994.

[ALU 94b] ALUR R., FIX L., HENZINGER T.A., « A Determinizable Class of Timed Automata », *Proceedings of the 6th International Conference on Computer Aided Verification (CAV'94)*, vol. 818 de *Lecture Notes in Computer Science*, p. 1-13, Springer Verlag, 1994.

[BER 98] BÉRARD B., DIEKERT V., GASTIN P., PETIT A., « Characterization of the Expressive Power of Silent Transitions in Timed Automata », *Fundamenta Informaticae*, vol. 36, n° 2-3, p. 145-182, IOS Press, 1998.

[BOU 05] BOUYER P., CHEVALIER F., D'SOUZA D., « Fault Diagnosis Using Timed Automata », SASSONE V. (dir.), *Proceedings of the 8th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'05)*, vol. 3441 de *Lecture Notes in Computer Science*, p. 219-233, Springer Verlag, avril 2005.

[CHE 04] CHEVALIER F., Détection d'erreurs dans les systèmes temporisés, Rapport de DEA, DEA Algorithmique, Paris, France, septembre 2004.

[FIN 05] FINKEL O., « On decision problems for timed automata », *Bulletin of the European Association for Theoretical Computer Science*, vol. 87, p. 185-190, 2005.

[HEN 96] HENZINGER T.A., « The Theory of Hybrid Automata. », *11th Annual IEEE Symposium on Logic in Computer Science (LICS'96)*, p. 278-292, New Brunswick, New Jersey, Etats-Unis, IEEE Computer Society, juillet 1996.

[HOL 96] HOLZMANN G., PELED D., YANNAKAKIS M., « On nested depth-first search », *Proceedings of the 2nd SPIN Workshop*, p. 23-32, American Mathematical Society, 1996.

[HOL 05] HOLZMANN G.J., « Software model checking with SPIN », *Advances in Computers*, vol. 65, p. 78-109, 2005.

[JIA 01] JIANG S., HUANG Z., CHANDRA V., KUMAR R., « A Polynomial Algorithm for Testing Diagnosability of Discrete Event Systems », *IEEE Transactions on Automatic Control*, vol. 46, n° 8, août 2001.

[JIA 06] JIANG S., KUMAR R., « Diagnosis of Dense-Time Systems Using Digital Clocks », *Proceedings of the American Control Conference (ACC'06)*, IEEE Computer Society, juin 2006.

[KRI 04a] KRICHEN M., TRIPAKIS S., « Black-box conformance testing for real-time systems », GRAF S., MOUNIER L. (dir.), *Proceedings of the 11th International SPIN Workshop on Model Checking of Software (SPIN'04)*, vol. 2989 de *Lecture Notes in Computer*

Science, p. 109-126, Springer Verlag, 2004.

[KRI 04b] KRICHEN M., TRIPAKIS S., « Real-time Testing with Timed Automata Testers and Coverage Criteria », LAKHNECH Y., YOVINE S. (dir.), *Proceedings of Formal Techniques, Modelling and Analysis of Timed and Fault Tolerant Systems (FORMATS-FTRTFT'04)*, vol. 3253 de *Lecture Notes in Computer Science*, p. 134-151, Springer Verlag, 2004.

[SAM 95] SAMPATH M., SENGUPTA R., LAFORTUNE S., SINNAMOHIDEEN K., TENEKETZIS D., « Diagnosability of Discrete Event Systems », *IEEE Transactions on Automatic Control*, vol. 40, n° 9, septembre 1995.

[SAM 96] SAMPATH M., SENGUPTA R., LAFORTUNE S., SINNAMOHIDEEN K., TENEKETZIS D., « Failure Diagnosis Using Discrete-Event Models », *IEEE Transactions on Control Systems technology*, vol. 4, n° 2, mars 1996.

[SCH 99] SCHNOEBELEN P., BÉRARD B., BIDOIT M., LAROUSSINIE F., PETIT A., *Vérification de logiciels : Techniques et outils de model-checking*, Vuibert, Paris, 1999.

[TRI 02] TRIPAKIS S., « Fault Diagnosis for Timed Automata », DAMM W., OLDEROG E.R. (dir.), *Proceedings of the International Conference on Formal Techniques in Real Time and Fault Tolerant Systems (FTRTFT'02)*, vol. 2469 de *Lecture Notes in Computer Science*, p. 205-224, Springer Verlag, 2002.

[TRI 06] TRIPAKIS S., « Folk theorems on the determinization and minimization of timed automata », *Information Processing Letters*, vol. 99, n° 6, p. 222-226, Elsevier, North-Holland, Inc., 2006.

[YOO 02] YOO T.S., LAFORTUNE S., « Polynomial-Time Verification of Diagnosability of Partially-Observed Discrete-Event Systems », *IEEE Transactions on Automatic Control*, vol. 47, n° 9, p. 1491-1495, septembre 2002.

[YOO 03] YOO T.S., GARCIA H., « Computation of Fault Detection Delay in Discrete-Event Systems », *Proceedings of the 14th International Workshop on Principles of Diagnosis, DX'03*, p. 207-212, Washington, D.C., Etats-Unis, juin 2003.