Contrôle des systèmes temporisés

Franck Cassez, Nicolas Markey

Chapitre 5

Contrôle des systèmes temporisés

Ce chapitre traite de la synthèse de contrôleurs pour les systèmes temporisés. Par systèmes temporisés nous entendons les systèmes dans lesquels des contraintes de temps *quantitatives* sont spécifiées. Les notions de base que nous utilisons dans ce chapitre s'appuient sur les concepts introduits dans les chapitre 3 et chapitre 4 ; la lecture de ces chapitres est donc recommandée au lecteur qui ne connaitrait pas les fondements des automates temporisés.

5.1. Introduction

On ne peut pas toujours utiliser du temps *discret* pour analyser des systèmes comportant des contraintes temporelles. C'est pourquoi les modèles à temps *dense* comme les automates temporisés [ALU 94] sont nécessaires. L'objet de ce chapitre est de donner des algorithmes de synthèse de contrôleurs pour de tels systèmes et d'identifier les propriétés particulières liées au temps dense.

5.1.1. *Vérification des systèmes temporisés*

Une approche classique de vérification, le *model checking* [SCH 99], consiste à construire un modèle complet S du comportement du système étudié (par exemple un automate fini), à formaliser la propriété de correction attendue par une formule ψ de logique (temporelle), puis à utiliser un algorithme (de *model checking*) pour vérifier que S satisfait (ou non) ψ, ce qui est noté $S \models \psi$ (resp. $S \not\models \psi$). Ces techniques

Chapitre rédigé par Franck CASSEZ et Nicolas MARKEY.

sont maintenant bien connues et couramment utilisées dans le domaine industriel. Cependant, pour certains types d'applications, il est nécessaire de prendre en compte des caractéristiques temporelles autres que le temps *logique* capturé par un modèle du type automate fini. Par exemple, pour des problèmes d'ordonnancement, les durées des tâches doivent être prises en compte explicitement. Il est parfois possible de se ramener à un modèle (en temps) discret mais cela peut s'avérer un facteur limitant : si un système a des *constantes* de temps variant de 1 à 10 000, considérer une horloge discrète de granularité 1 engendre un nombre d'états prohibitif pour les algorithmes de *model checking* ; d'autre part, l'utilisation du temps discret suppose que l'on connaisse exactement les durées des diverses opérations à réaliser. Pour prendre en compte une *incertitude* sur ces durées (durée des tâches, des communications), on utilise donc des modèles plus fins que les automates finis, par exemple les *automates temporisés* [ALU 94] (*Timed Automata*, TA), où l'on peut utiliser des *horloges* pour spécifier le comportement temporel du système. Comme cela est décrit dans le chapitre 4, les algorithmes de *model checking* utilisés sur les structures discrètes (automates finis, logique temporelle) ont été étendus aux modèles temporisés et ont conduit au développement d'outils de vérification dédiés tels que UPPAAL [AMN 01], KRONOS [YOV 97], CMC [LAR 98, LAR 05] ou encore HYTECH [HEN 97].

5.1.2. *Le problème de la synthèse de contrôleur*

Dans le cas du model checking , on dit que les systèmes considérés sont *fermés* : on travaille sur un modèle complet du système incluant l'environnement à contrôler, les éventuels actionneurs et le contrôleur ; ce système évolue sans influence extérieure. Dans le cadre de la synthèse de contrôleurs, on part d'un système *ouvert*, le but étant de le fermer : si S est un modèle du système ouvert à contrôler, on lui ajoute un contrôleur C, et $C(S)$ représente le modèle complet ou *fermé* du système, ou encore « S contrôlé par C ».

Si la propriété de correction à satisfaire est ϕ, les techniques de *model checking* permettent de répondre à la question « est-ce que S contrôlé par C satisfait ϕ ? ». Le problème de *model checking* s'écrit alors formellement :

étant donnés S, C et ϕ, est-ce que $C(S) \models \phi$? [MCP]

Cette approche nécessite donc d'abord de construire (éventuellement à la main) un contrôleur C, de manière à définir complètement le système, et ensuite à vérifier le système contrôlé $C(S)$. Ceci peut s'avérer difficile dans le cas de systèmes complexes ou/et avec des propriétés difficiles à mettre en œuvre (comme par exemple des propriétés dépendant du temps). De plus, si la propriété attendue n'est pas vérifiée, on doit modifier le contrôleur et vérifier de nouveau le système de façon itérative jusqu'à

obtenir un contrôleur correct[1]. Idéalement, on souhaiterait ne décrire que le système à contrôler et *calculer* (ou *synthétiser*) un contrôleur (s'il en existe un), de manière à ce que la spécification ϕ soit satisfaite. C'est la problématique du *contrôle*, plus générale que celle du *model checking*. Le problème du *contrôle* (*Control Problem*, CP) pour les systèmes ouverts est formellement le suivant :

étant donnés S et ϕ, existe-t-il C tel que $C(S) \models \phi$? [CP]

Pour répondre au problème [CP] il est souvent nécessaire de restreindre la classe de modèles dans laquelle on va chercher un contrôleur. Par exemple, on peut chercher des contrôleurs qui sont des automates finis (donc à mémoire bornée), ou bien des automates temporisés. Le problème naturel qui se pose après [CP] est celui de la *synthèse de contrôleur* (*Controller Synthesis Problem*, CSP) :

Si la réponse à [CP] est « oui », peut-on construire un tel contrôleur C ? [CSP]

Ces problèmes ont été formalisés et étudiés pour les systèmes finis dans les travaux pionniers de Ramadge et Wonhan [RAM 87, RAM 89]. On peut retrouver ces résultats en abordant le problème à l'aide de *la théorie des jeux*.

5.1.3. *Du contrôle aux jeux*

Un problème de contrôle peut être formulé simplement dans le cadre de la *théorie des jeux* discrets [ARN 03, MCN 93, RIE 03, THO 95] : on modélise dans ce cas le système par un système de transitions dont l'évolution résulte de l'interaction de plusieurs agents (ou joueurs). Dans ce cadre, le contrôleur recherché est en fait une *stratégie* pour un agent (ou une coalition d'agents) leur permettant d'assurer que leur objectif sera atteint, quel que soit le comportement des autres agents.

Un exemple de structure de jeu est donné à la figure 5.1. Dans ce jeu, les états ronds sont « contrôlés » par le joueur 1 (qui est alors seul responsable du choix de la transition à suivre), et les états carrés sont contrôlés par le joueur 2.

Considérons la stratégie suivante pour le joueur 1 : lorsque le jeu est dans l'état ℓ_0, il prend la transition vers ℓ_2, et si le jeu est dans l'état ℓ_3, il prend la transition vers l'état FINAL. Si le joueur 1 suit cette stratégie, alors quelle que soit la stratégie adoptée par le joueur 2, le jeu finira nécessairement par atteindre l'état FINAL. Cette stratégie est donc *gagnante* pour le joueur 1 si son objectif est d'atteindre l'état FINAL (on parle d'objectif d'*accessibilité*).

1. Ce qui peut être très long si un tel contrôleur n'existe pas ...

Si au contraire son objectif est de ne jamais atteindre cet état (objectif de *sûreté*), il est facile de voir qu'il a également une stratégie gagnante, qui consiste à toujours aller vers l'état ℓ_1.

Enfin, si son objectif est de visiter infiniment souvent l'état ℓ_3 (objectif d'*accessibilité répétée*, ou *de Büchi*), le joueur 1 n'a pas de stratégie gagnante : en effet, si dans ℓ_0 il choisit d'aller dans ℓ_2, le jouer 2 amène le système dans FINAL ; s'il choisit d'aller dans ℓ_1, le joueur 2 ramène le système dans ℓ_0.

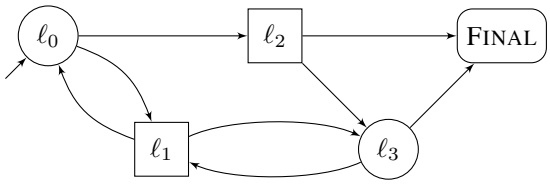

Figure 5.1. *Un exemple de jeu*

Pour formuler un problème de contrôle où le contrôleur doit contrôler un environnement, il suffit de voir le contrôleur comme le joueur 1 par exemple, et l'environnement comme le joueur 2. Le problème de contrôle devient donc : existe-t-il une *stratégie* gagnante pour le joueur 1 ? Une stratégie gagnante correspond alors à un contrôleur permettant de contrôler l'environnement.

5.1.4. *Les différents objectifs de jeux*

Étant donné un jeu G et un objectif de contrôle ϕ, on considère le couple (G, ϕ) comme la spécification du problème suivant : « existe-t-il une stratégie C pour le joueur 1 dans G telle que $C(G) \models \phi$? ». Si ϕ est un objectif de sûreté, en général donné par un ensemble d'états à éviter ou à préserver, on dit que (G, ϕ) est un jeu de sûreté et on parle de *Safety Control Problem (SafCP)*. Si ϕ est un objectif d'atteignabilité, que l'on peut toujours supposer comme étant un état désigné BUT, alors (G, BUT) est un jeu d'atteignabilité et on parle de *Reachability Control Problem (RCP)*. On peut définir des objectifs plus compliqués comme on l'a vu précédemment (accessibilité répétée ou encore Büchi, Co-Büchi, etc.) et donner ces objectifs sous forme de formule de logique temporelle (temporisée ou pas).

5.1.5. *Les différents types de jeux non temporisés*

Un jeu tel que celui de la figure 5.1 est un jeu « à tour ». Comme aux échecs ou aux dames, chacun des joueurs joue alternativement. De façon plus générale, on peut considérer des jeux à tour où les états sont de types carré ou rond : dans les états ronds

c'est le joueur 1 qui joue, et dans les carrés, le joueur 2. Dans ce cas, le numéro du joueur qui joue est déterminé par le type de l'état, mais il ne change pas forcément à chaque coup. Si un jeu est tel que, dans un état donné, les deux joueurs peuvent jouer en même temps, on a un jeu « concurrent ». Le jeu « pierre, feuille, ciseaux », par exemple, est un jeu concurrent. Nous parlerons brièvement, à la section 5.10 de ce chapitre, de l'extension temporisée de ces jeux.

Les jeux à tour ont de bonnes propriétés. Par exemple, ils sont *déterminés* pour une grande classe $\mathcal{B}$ d'objectifs[2] [MAR 75], ce qui signifie que si le joueur 1 n'a pas de stratégie pour gagner (G, ϕ) avec ϕ un l'objectif de $\mathcal{B}$, alors le joueur 2 a une stratégie pour gagner $(G, \neg\phi)$ où $\neg\phi$ est le complément de ϕ. Les jeux concurrents n'ont pas, en général, cette propriété.

De plus, pour les conditions de gain que nous avons mentionnées ci-dessus (sûreté, accessibilité, accessibilité répétée, etc.), les jeux (à tour ou concurrents) que nous avons décrits ici satisfont les propriétés suivantes [THO 95] :

– on peut décider en temps polynomial si le joueur 1 a une stratégie gagnante dans le jeu (G, ϕ) ;

– si le joueur 1 peut gagner (G, ϕ), alors il peut le faire avec une stratégie *sans mémoire* ou *positionnelle* (*memoryless* ou *positional* en anglais) ;

– si le joueur 1 peut gagner (G, ϕ), il existe une stratégie « la plus permissive », qui est également sans mémoire.

Dans la section 5.2 on introduit les concepts de base des jeux temporisés avec les définitions formelles des jeux, des stratégies, systèmes contrôlés. Dans la section 5.3, on donne les algorithmes de base permettant de résoudre le problème de contrôle pour les automates temporisés. Les sections suivantes abordent des notions plus avancées concernant le contrôle temporisé. La section 5.4 montre les limites des algorithmes précédents et dans la section 5.5 on s'intéresse à la notion d'*implémentabilté* des contrôleurs. Dans la section suivante 5.6, on aborde brièvement les aspects concernant la spécification des objectifs de contrôle. La section 5.7 traite du problème de contrôle optimal pour les jeux d'accessibilité. L'implémentation efficace des algorithmes de synthèse de contrôleurs fait l'objet de la section 5.8 ; un complément à cette section peut être trouvé à la fin du chapitre 8. Enfin, la section 5.10 est consacrée à une extension du modèle de jeu temporisé introduit dans la section 5.2.

5.2. Les jeux temporisés

Cette première section a pour but de définir le cadre des jeux temporisés, ainsi que les notions connexes. La notion de *jeu temporisé* que nous étudions ici a été proposée

2. Qui inclut les objectifs de Büchi.

dans [ASA 98, MAL 95] : c'est une variante des automates temporisés dans laquelle l'ensemble des actions est partitionné entre les deux joueurs. Outre ces actions, les joueurs peuvent également décider de ne rien faire : dans ce cas, le temps s'écoulera jusqu'à ce qu'un des joueurs souhaite faire une action.

5.2.1. *Jeux temporisés*

DÉFINITION 5.2.1 (AUTOMATE TEMPORISÉ DE JEU) *Un* automate temporisé de jeu *(ou* jeu temporisé*) est un* 7*-uplet* $\mathcal{G} = (L, \ell_0, X, \Sigma_1, \Sigma_2, E, Inv)$ *remplissant les conditions suivantes :*

– Σ_1 *et* Σ_2 *sont deux alphabets disjoints ;*

– $(L, \ell_0, X, \Sigma_1 \cup \Sigma_2, E, Inv)$ *est un automate temporisé (suivant la définition 4.2.2 du chapitre 4).*

Un jeu temporisé est donc un automate temporisé dont l'ensemble des actions est partagé entre les deux joueurs : les transitions qui peuvent être activées par le contrôleur (que nous appellerons *contrôlables*) et celles qui peuvent être activées par l'environnement (dites *incontrôlables*).

De manière informelle, une partie d'un jeu temporisé se joue de la façon suivante : elle commence dans l'état ℓ_0 avec toutes les horloges à zéro, et est composée d'une succession de transitions et de délais d'attente. Les transitions de temps durent aussi longtemps qu'aucun des deux joueurs ne souhaite faire une action. Dans le cas particulier des jeux à tours, le délai d'attente dans un état est choisi par le seul joueur qui contrôle cet état.

EXEMPLE 5.2.1 *Un exemple de jeu temporisé est représenté à la figure 5.2 [CAS 05]. Les traits pointillés sont les actions incontrôlables (alphabet* Σ_2*), et les traits pleins, les actions contrôlables (alphabet* Σ_1*). Nous utiliserons cette convention dans toute la suite de ce chapitre.*

Dans cet exemple, tous les invariants sont vrais et donc on peut toujours laisser passer du temps. Imaginons que le contrôleur souhaite jouer l'action c_1 *lorsque l'horloge* x *vaut* 1*. Si l'environnement n'effectue pas l'action* u_2 *avant cette date, c'est effectivement l'action* c_1 *qui sera appliquée, et le jeu arrivera dans l'état* ℓ_1 *avec* $x = 1$*. Le contrôleur pourra alors à nouveau attendre durant un délai d'une unité de temps et proposer l'action* c_2*, qui le mène dans l'état* BUT*. Il est à noter que, dans l'état* ℓ_1 *avec* $x \geq 1$*, l'environnement ne peut pas proposer d'action. Il est contraint d'attendre que le contrôleur fasse une action.*

Bien entendu, avec l'action « jouer le coup c_1 lorsque $x = 1$ *» choisie par le contrôleur dans l'état initial, il aurait également été possible que l'environnement*

choisisse un délai plus court, et joue l'action u_2 lorsque $x = 0{,}5$ (l'action u_1 n'est pas disponible lorsque $x \leq 1$). Le jeu aurait alors atteint l'état $(\ell_2, x = 0)$ (puisque l'horloge doit être mise à zéro sur la transition allant de ℓ_0 à ℓ_2).

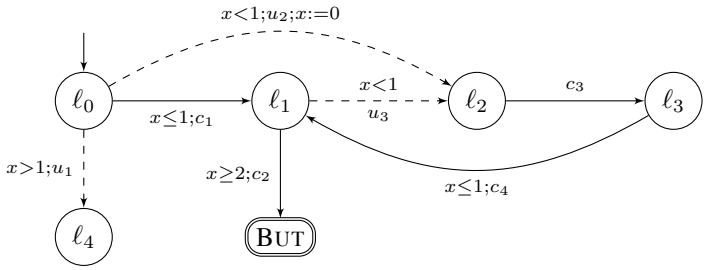

Figure 5.2. *Un exemple de jeu temporisé*

Ces quelques exemples illustrent de manière informelle l'intuition que l'on peut avoir du déroulement d'un jeu temporisé. Nous allons maintenant y associer un cadre mathématique, qui nous permettra par la suite de vérifier formellement des propriétés de ces jeux.

Dans la suite on considère implicitement le jeu $\mathcal{G} = (L, \ell_0, X, \Sigma_1, \Sigma_2, E, Inv)$.

5.2.2. *Déroulement du jeu et stratégies*

Plusieurs sémantiques différentes ont été proposées pour les jeux temporisés : l'ajout de la composante temporelle nécessite en effet quelques précisions dans les « règles du jeu ». Nous commençons par étudier une sémantique assez naturelle des jeux temporisés, proposée dans [MAL 95]. Nous verrons dans la section 5.10 les défauts de cette sémantique (notamment le fait qu'elle accepte les stratégies qui consistent à « *bloquer le temps* ») et une façon d'y remédier, proposée dans [ALF 03].

5.2.2.1. *Exécutions d'un jeu temporisé*

Notre définition des exécutions d'un jeu temporisé reprend celle des automates temporisés ; d'après la définition 5.2.1 un jeu temporisé est un automate temporisé dans lequel on a partitionné l'ensemble d'actions en actions contrôlables et incontrôlables. Étant donné l'automate de jeu temporisé $\mathcal{G} = (L, \ell_0, X, \Sigma_1, \Sigma_2, E, Inv)$, on note $\mathcal{T}_\mathcal{G}$ le système de transitions temporisé qui donne sa sémantique (voir le chapitre 4). Une exécution d'un jeu temporisé $\mathcal{G}$ est une exécution du système de transitions temporisé $\mathcal{T}_\mathcal{G}$: c'est une suite

$$\rho = s_0 \xrightarrow{d_1} s'_0 \xrightarrow{a_1} s_1 \xrightarrow{d_2} s'_1 \xrightarrow{a_2} \cdots \xrightarrow{d_{n-1}} s'_n \xrightarrow{a_n} s_n \cdots$$

avec $a_i \in \Sigma_1 \cup \Sigma_2$, $d_i \in \mathbb{R}_{\geq 0}$, et où les configurations s_i et s'_i sont des paires (ℓ, v), $\ell \in L$ et v une valuation des horloges de X. Les transitions continues alternent donc avec les transitions discrètes (deux transitions de temps consécutives peuvent toujours être regroupées, et on autorise en particulier la dernière transition d'une exécution finie à être une transition de temps de durée infinie). L'ensemble des exécutions d'un jeu temporisé $\mathcal{G}$ est noté $\text{Exec}_{\mathcal{G}}$. Nous définissons également le sous-ensemble des *exécutions maximales* comme étant l'ensemble des exécutions qui ne peuvent pas être « prolongées ». Cela inclut plusieurs types d'exécutions :

– les exécutions comportant un nombre infini de transitions discrètes ;

– les exécutions finies de durée infinie ;

– les exécutions finies se terminant dans un état depuis lequel aucune transition (de temps ou d'action) n'est possible.

Nous notons $\text{Exec}^m_{\mathcal{G}}$ l'ensemble des exécutions maximales de $\mathcal{G}$ et $\text{Exec}^f_{\mathcal{G}}$ l'ensemble des exécutions finies. La longueur d'une exécution finie est le nombre de transitions (tous types de transitions confondus) qui ont lieu le long de cette exécution.

5.2.2.2. *Stratégies*

Les stratégies sont l'élément central de la théorie des jeux. Une stratégie dit à un joueur ce qu'il doit jouer, en fonction de l'historique de la partie. Dans le cas des jeux temporisés, les joueurs sont autorisés à attendre sans rien faire (c'est la transition λ dans la définition ci-dessous). Comme nous le verrons par la suite, il faut que les deux joueurs jouent cette action spéciale pour que le temps s'écoule. Dès que l'un des deux joueurs décide de jouer une transition qui lui est permise, alors cette transition sera franchie.

DÉFINITION 5.2.2 (STRATÉGIE) *Soit $j \in \{1, 2\}$ un joueur pour $\mathcal{G}$. Une* stratégie *pour le joueur j est une fonction s: $\text{Exec}^f_{\mathcal{G}} \to \Sigma_j \cup \{\lambda\}$ remplissant les conditions suivantes : soit ρ est une exécution finie se terminant dans la configuration $q = (\ell, v)$, alors :*

– si $s(\rho) = a$ avec $a \in \Sigma_1$, alors il existe une transition $q \xrightarrow{a} q'$ dans $\mathcal{T}_{\mathcal{G}}$;

– si $s(\rho) = \lambda$, alors il existe $d > 0$ tel que $q \xrightarrow{d} q'$ dans $\mathcal{T}_{\mathcal{G}}$.

Ces conditions expriment que le coup proposé par la stratégie doit être autorisé dans les actions permises par le jeu $\mathcal{G}$. L'ensemble des stratégies du joueur j dans $\mathcal{G}$ sera noté $\text{Strat}_j(\mathcal{G})$.

Lorsqu'un joueur joue selon une stratégie, il restreint l'ensemble des exécutions possibles du système aux exécutions *conformes* à cette stratégie.

DÉFINITION 5.2.3 (JEU CONTRÔLÉ) *Soit $q = (\ell, v)$ un état de $\mathcal{G}$, et s une stratégie pour le joueur* 1 *(le cas du joueur* 2 *est symétrique). L'ensemble des* exécutions finies conformes à la stratégie s depuis q, *noté Out$^f(q, s)$, est défini inductivement de la façon suivante :*

– q est la seule exécution finie de longueur nulle appartenant à Out$^f(q, s)$;

– une exécution finie de $\mathcal{G}$ de la forme $\rho = \rho' \xrightarrow{\sigma} q'$ (donc de longueur non nulle) appartient à Out$^f(q, s)$ si, et seulement si, ρ appartient à Out$^f(q, s)$ et l'une des trois conditions suivante est satisfaite :

- σ appartient à Σ_2 ;

- σ appartient à Σ_1 et $\sigma = s(\rho)$;

- $\sigma \in \mathbb{R}_{\geq 0}$ et, pour tout d tel que $0 \leq d < \sigma$, on a $s(\rho \xrightarrow{d} q'') = \lambda$.

L'ensemble des exécutions maximales conformes à s, notées Out$^m(q, s)$, est l'union des deux ensembles d'exécutions suivants :

– l'ensemble des exécutions maximales finies conformes à s ;

– l'ensemble des exécutions infinies dont tous les préfixes sont conformes à s.

Comme le sous-entend notre définition, un joueur peut se souvenir de l'intégralité de la partie pour déterminer la stratégie à appliquer. Certaines stratégies ont cependant la particularité d'être indépendantes de l'historique de la partie, et de ne dépendre que de l'état courant. Ces stratégies sont dites *positionnelles* ou *sans mémoire*. Dans ces cas-là, pour simplifier les notations, on considérera que les stratégies sont des applications de l'ensemble des états (et non plus de l'ensemble des exécutions finies) dans l'ensemble des actions du joueur. L'exemple ci-dessous décrit une stratégie positionnelle.

EXEMPLE 5.2.2 *Reprenons l'exemple de la figure 5.2, et considérons la stratégie positionnelle $\tilde{s}$ suivante :*

– dans les états (ℓ_0, v) avec $v(x) < 1$, le joueur 1 (contrôleur) joue λ. Lorsque $v(x) = 1$ il joue c_1. Si $v(x) > 1$, il propose de nouveau λ puisque c'est le seul coup qui lui est autorisé. On peut définir formellement $\tilde{s}(\ell_0, x < 1 \vee x > 1) = \lambda$ et $\tilde{s}(\ell_0, x = 1) = c_1$;

– dans les états (ℓ_1, v), il joue λ pour $v(x) < 2$ et l'action c_2 quand $v(x) \geq 2$ c'est-à-dire $\tilde{s}(\ell_1, x < 2) = \lambda$ et $\tilde{s}(\ell_1, x \geq 2) = c_2$;

– dans les états (ℓ_2, v), il joue c_3, soit $\tilde{s}(\ell_3, -) = c_3$;

– dans les états (ℓ_3, v), si $v(x) < 1$, il joue λ et quand $v(x) = 1$ il joue c_4 ; quand $v(x) > 1$ il doit jouer λ : $\tilde{s}(\ell_3, x < 1 \vee x > 1) = \lambda$ et $\tilde{s}(\ell_3, x = 1) = c_4$;

– dans les états ℓ_4 et BUT, *le contrôleur a gagné et joue λ, c'est-à-dire $\tilde{s}(\ell_4, -) = \lambda$ et $\tilde{s}(\text{BUT}, -) = \lambda$.*

Parmi les exécutions issues de l'état $(\ell_0, 0)$ *et conformes à cette stratégie, certaines passent directement par l'état* ℓ_1*, d'autres passent par les états* ℓ_2 *et* ℓ_3 *(si le joueur 2 joue* u_2*), mais on se convainc facilement qu'aucune n'atteint l'état* ℓ_4 *et que toutes atteignent en fait l'état* BUT.

Soit Ω un ensemble d'exécutions (dites *gagnantes*) qui définissent l'objectif de contrôle. On suppose que cet ensemble est clos par suffixe[3]. On écrit dans la suite $(\mathcal{G}, \Omega)$ pour indiquer que l'on considère le jeu $\mathcal{G}$ avec l'objectif de contrôle Ω.

DÉFINITION 5.2.4 (ÉTAT GAGNANT, STRATÉGIE GAGNANTE) *Soit* $s \in Strat_j(\mathcal{G})$ *une stratégie pour le joueur* j*. La stratégie* s *du joueur* j *est* gagnante *à partir de* q *pour l'objectif* Ω *si* $Out^m(q, s) \subseteq \Omega$*, c'est-à-dire si toutes les exécutions maximales conformes à la stratégie sont des exécutions gagnantes.*

Un état q *est* gagnant *pour le joueur* j *si ce joueur a une stratégie gagnante à partir de* q*. On note* $\mathsf{WinState}_j(\mathcal{G})$ *l'ensemble des états gagnants de* $\mathcal{G}$ *pour le joueur* j*.*

Une stratégie s est gagnante pour $(\mathcal{G}, \Omega)$ si elle est gagnante depuis l'état initial : $\text{Out}^m((\ell_0, v_0), s) \subseteq \Omega$. On note $\mathsf{WinStrat}_j(\mathcal{G})$ l'ensemble des stratégies gagnantes du joueur j. Dans la suite, nous nous intéresserons aux stratégies gagnantes pour le joueur 1 (contrôleur). Ainsi, on ne mentionnera pas le nom du joueur et on écrira $\mathsf{WinStrat}(\mathcal{G})$, lorsque l'on parlera du joueur 1.

EXEMPLE 5.2.3 *Dans notre exemple de la figure 5.2, l'état* BUT *est l'objectif que le contrôleur veut atteindre. L'ensemble* Ω *est donc l'ensemble des exécutions qui contiennent l'état* BUT*(cet ensemble est bien clos par préfixe). La stratégie* $\tilde{s}$ *que nous avons décrite ci-dessus est gagnante depuis l'état initial.*

5.3. Calcul des états gagnants et des stratégies

Des algorithmes pour la synthèse de contrôleur pour les automates temporisés de jeu sont donnés dans [ALF 01, ALF 03, ASA 98, MAL 95]. Ils sont basés sur des calculs de points fixes (comme dans le cas des jeux non temporisés) : on calcule récursivement l'ensemble des états gagnants pour des parties de longueurs au plus n, pour tout n. A la limite, on aura calculé l'ensemble des états gagnants. La convergence de

3. C'est-à-dire que si une exécution finie ρ appartient à Ω, alors toutes les exécutions qui prolongent ρ appartiennent aussi à Ω.

cet algorithme repose sur le fait que la relation d'équivalence sur les valuations d'horloges, introduite dans le chapitre 4 pour définir les *régions*, est toujours correcte pour les jeux temporisés.

Nous illustrons cette section avec l'exemple de la figure 5.3, où l'objectif est d'éviter l'état BAD. Le résultat de l'algorithme de synthèse de contrôleur sur cet exemple est donné à la fin de la section.

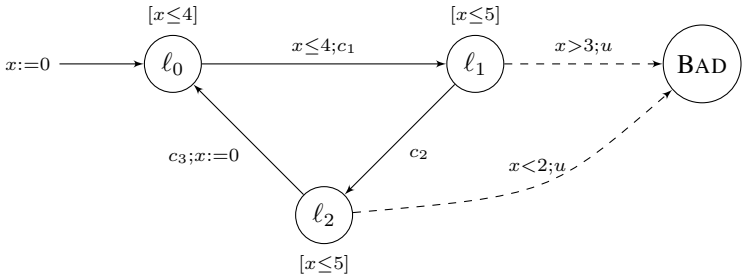

Figure 5.3. *Un exemple de jeu temporisé – le système ouvert S.*

5.3.1. *Prédécesseurs contrôlables*

Afin de mettre en œuvre notre algorithme de calcul récursif de points fixes, nous définissons les *prédécesseurs contrôlables* d'un ensemble d'états : en effet, pour gagner une partie finie d'au plus $n+1$ étapes, il faut et il suffit que le contrôleur ait une stratégie pour que, en une transition, le jeu aille dans l'ensemble des états gagnants en au plus n étapes,

Soit $\mathcal{G}$ un jeu temporisé, et $\mathcal{T}_{\mathcal{G}}$ le système de transitions temporisé donnant sa sémantique. Pour un sous-ensemble X de l'ensemble d'états S de $\mathcal{T}_{\mathcal{G}}$, on définit les ensembles $\mathit{CPre}(X)$ et $\mathit{UPre}(X)$ par :

$$\mathit{CPre}(X) = \{q \in S \mid \exists c \in \Sigma_c \text{ tel que } q \xrightarrow{c} q' \text{ et } q' \in X\};$$
$$\mathit{UPre}(X) = \{q \in S \mid \exists u \in \Sigma_u \text{ tel que } q \xrightarrow{u} q' \text{ et } q' \in X\}.$$

Ces deux ensembles correspondent aux états à partir desquels il existe une action contrôlable (respectivement incontrôlable) qui permet au contrôleur (respectivement à l'environnement) de forcer à atteindre l'ensemble X après la transition correspondante. Il nous reste à prendre en compte les transitions de temps, au cours desquelles les joueurs jouent l'action λ. La figure 5.4 décrit les conditions pour qu'un état s soit un *prédécesseur contrôlable* d'un ensemble d'états X : il faut que le contrôleur puisse

forcer le système à aller dans un état de X, en laissant éventuellement passer du temps puis en faisant une action contrôlable c ; sur tous les états rencontrés lors du passage du temps il ne doit pas se trouver dans un état où l'environnement peut l'amener à l'extérieur de X (noté $\overline{X}$). Sur la figure 5.4, l'état q' n'est pas un prédécesseur contrôlable de X, puisque lors de la transition de délai, le jeu passe par des états ($UPre(\overline{X})$ d'où l'environnement peut amener le jeu dans $\overline{X}$. L'état q est lui un prédécesseur contrôlable de X.

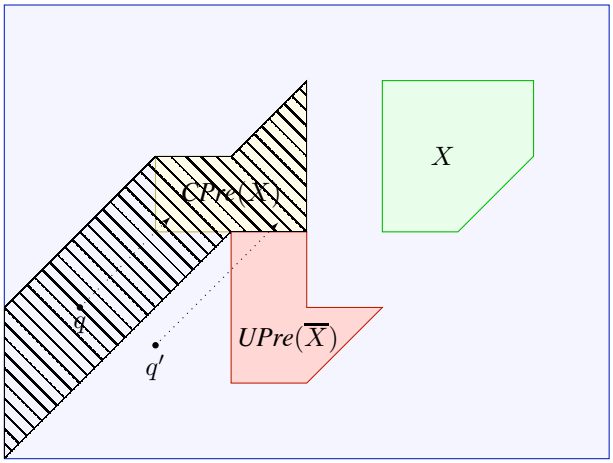

Figure 5.4. *Prédécesseurs contrôlables*

DÉFINITION 5.3.1 (PRÉDÉCESSEUR CONTRÔLABLE) *Un état q est un* prédécesseur contrôlable *de X si les deux conditions suivantes sont remplies :*

1) il existe $\delta \geq 0$ tel que $q \xrightarrow{\delta} q'$ et $q' \in CPre(X)$;

2) pour tout $0 \leq t \leq \delta$ tel que $q \xrightarrow{t} q_t$, on a $q_t \notin UPre(\overline{X})$.

L'ensemble des prédécesseurs contrôlables de X est noté $\pi(X)$.

Sur le schéma de la figure 5.4, la zone hachurée correspond aux prédécesseurs contrôlables de l'ensemble X. Dans l'exemple de la figure 5.3, l'état $(\ell_1, x = 1)$ est un prédécesseur contrôlable de $(\ell_2, x = 2)$: on peut laisser passer du temps de $(\ell_1, x = 1)$ jusqu'à $(\ell_1, x = 2)$ puis tirer c_2 de ℓ_1 à ℓ_2. Lors du passage du temps aucune transition incontrôlable n'est franchissable.

REMARQUE 5.3.1 *On peut remarquer que notre définition des prédécesseurs contrôlables ne tient pas compte de certaines particularités des modèles temporisés. Par*

exemple, si l'on a un automate de jeu, avec objectif BUT*, et une transition incontrôlable* $(0, x \leq 1, u, \emptyset, \text{BUT})$ *avec l'invariant Inv*$(0) \equiv x \leq 1$*, le joueur 2 (environnement) est obligé de jouer* u *au plus tard à la date* 1*. Ainsi toutes les exécutions maximales satisfont l'objectif. Il est possible de prendre en compte ces états où l'environnement est forcé de jouer, au prix de quelques modifications dans nos définitions. Les résultats mentionnés dans cette section restent valides dans ce cas. Nous laissons cette étude au lecteur.*

5.3.2. *Opérateurs symboliques*

Tout comme dans le cas du *model checking* d'automates temporisés, l'ensemble des états d'un jeu temporisé est infini, et nous devons développer des techniques symboliques, regroupant les états par « *familles* » ayant les mêmes propriétés, afin de les analyser. Nous allons réutiliser la relation d'équivalence de la section 4.4.2 du chapitre 4 ici. Afin de montrer que cette approche est correcte, nous devons montrer d'une part que deux états d'une même région ont les mêmes « propriétés » (gagnants) vis-à-vis du jeu, et que nous pouvons calculer les prédécesseurs contrôlables pour les régions. Le lemme suivant énonce ces deux résultats.

LEMME 5.3.1 *Si* X *est une union de régions (une* zone*), alors :*
$\mathbf{P_1}$ *:* $CPre(X)$*,* $UPre(X)$ *et* $\pi(X)$ *sont des unions de régions (donc des zones) ;*
$\mathbf{P_2}$ *:* $CPre(X)$*,* $UPre(X)$ *et* $\pi(X)$ *sont effectivement calculables.*

La preuve de ces résultats nécessite l'introduction d'un opérateur symbolique permettant de calculer les prédécesseurs temporels de $CPre(X)$ ne passant pas dans l'ensemble $UPre(\overline{X})$. Cet opérateur est défini par :

$$Pre_t(A, B) = \{s \in S \mid \exists \delta \geq 0.\ s \xrightarrow{\delta} s' \wedge s' \in A \wedge \forall 0 \leq t \leq \delta.\ [(s \xrightarrow{t} s'') \Rightarrow (s'' \notin B)]\}.$$

EXEMPLE 5.3.1 *Pour l'automate temporisé de la figure 5.3, on a* $CPre(\ell_1, x \leq 3) = (\ell_0, x \leq 3)$ *et si* $Z = (\ell_0, x \leq 4) \cup (\ell_1, x \geq 0) \cup (\ell_2, x \geq 0)$ *alors* $\pi(Z) = Z'$ *avec* $Z' = (\ell_0, x \leq 3) \cup (\ell_1, 0 \leq x \leq 3) \cup (\ell_2, x \geq 2)$.

5.3.3. *Calcul symbolique des états gagnants*

Nous sommes maintenant en mesure de calculer symboliquement l'ensemble des états gagnants d'un jeu temporisé. Si l'on considère un objectif de sûreté, défini par

X_i	ℓ_0	ℓ_1	ℓ_2
0	$0 \leq x \leq 4$	$0 \leq x \leq 5$	$0 \leq x \leq 5$
1	$0 \leq x \leq 4$	$0 \leq x \leq 3$	$2 \leq x \leq 5$
2	$0 \leq x \leq 3$	–	–

(a) Etats gagnants

	ℓ_0	ℓ_1	ℓ_2
λ	$x < 3$	$x < 3$	$x < 5$
c	$x \leq 3$	$2 \leq x$	$x \leq 5$

(b) Stratégie la plus permissive f^*

Tableau 5.1. *Résultats des algorithmes pour l'exemple de la figure 5.3*

la donnée d'une union de régions BAD que le contrôleur doit éviter, alors l'ensemble des états gagnants pour ce jeu est le plus grand point fixe de la fonction $h\colon X \mapsto \overline{\text{BAD}} \cap \pi(X)$. Ce point fixe est bien défini, puisque la fonction h est décroissante. Pour l'obtenir, on commence par calculer $h(S)$ (où S est l'ensemble de tous les états du jeu), puis $h(h(S))$, *etc*. On obtient une suite décroissante d'unions de régions, qui converge en un nombre fini d'étapes puisqu'il n'y a qu'un nombre fini de régions. L'ensemble obtenu à la limite est le plus grand point fixe de h : c'est donc un sous-ensemble de $\overline{\text{BAD}}$, et le contrôleur a une stratégie lui permettant de rester dans cet ensemble à la prochaine transition d'action. Ce point fixe est aussi le plus grand sous-ensemble $\overline{\text{BAD}}$ que le contrôleur peut forcer.

EXEMPLE 5.3.2 *Dans le cas de notre exemple de la figure 5.3, on obtient itérativement les ensembles*[4] *du tableau 5.1. Pour les propriétés plus compliquées que des propriétés de sûreté, le calcul des états gagnants est plus complexe [ALF 01, ALF 03, ASA 98, MAL 95].*

REMARQUE 5.3.2 *Le cas des objectifs d'accessibilité se traite de manière similaire : si le but est d'atteindre une union de régions* BUT*, l'ensemble des états gagnants est le plus petit point fixe de la fonction* $h\colon X \mapsto \text{BUT} \cup \pi(X)$*. Là encore, il est possible de calculer ce point fixe itérativement, en calculant* $h(\varnothing)$*,* $h(h(\varnothing))$*, etc. La terminaison et la correction de cet procédure se montrent de la même façon que ci-dessus. Pour des objectifs plus compliqués (accessibilité répétée, etc.) on doit calculer des imbrications de point fixes (voir [MAL 95]).*

Comme on peut décider si l'état initial d'un automate appartient à une union de régions, le calcul des états gagnants permet de résoudre notre problème de départ :

THÉORÈME 5.3.1 ([ASA 98, HEN 99B]) *Les problèmes de contrôle SafCP et RCP sont décidables pour les jeux temporisés ; ils sont EXPTIME-complets.*

4. Le domaine gagnant est donné pour chaque localité ℓ_i dans la colonne correspondante.

5.3.4. *Synthèse de stratégies gagnantes*

Pour ce qui est du problème de synthèse de stratégie, dans le cas d'un objectif de sûreté, on peut définir la stratégie *maximale* ou *la plus permissive*. En toute rigueur, cette stratégie la plus permissive n'est pas une stratégie au sens de la partie 5.2.2.2 car il faudrait qu'elle associe une action de $\Sigma_c \cup \{\lambda\}$ à une exécution. La stratégie la plus permissive f^* associe à chaque exécution ρ un sous-ensemble non vide de $\Sigma_c \cup \{\lambda\}$. Toute (sous-)stratégie f (au sens de la partie 5.2.2.2) non bloquante telle que $f(\rho) \in f^*(\rho)$ est une stratégie gagnante, et f^* est maximale pour cette propriété (c'est pourquoi on la qualifie de stratégie la plus permissive). Concernant la synthèse de stratégies on a le théorème suivant :

THÉORÈME 5.3.2 ([ASA 98]) *Si $\mathcal{G}$ est un automate temporisé de jeu tel que l'état initial de $\mathcal{G}$ est gagnant pour un objectif de sûreté, alors il existe une stratégie (la plus permissive) f^* gagnante et sans mémoire.*

Pour calculer cette stratégie on commence par renforcer les gardes des transitions de telle manière qu'en partant d'un état gagnant s, si l'on franchit une transition contrôlable dans l'automate de jeu avec les gardes renforcées, alors on atteint un état gagnant. Les renforcements des gardes sont donc les plus grandes *préconditions* permettant d'atteindre des états gagnants. Dans notre cas on doit renforcer la garde de c_1 en ajoutant la contrainte $x \leq 3$, celle de c_2 en ajoutant la contrainte $x \geq 2$ (qui indique que l'on ne doit pas tirer trop tôt c_2). La stratégie la plus permissive est ensuite donnée par : attendre pour tous les états gagnants q tels qu'il existe $\delta > 0$ et $q \xrightarrow{\delta} q'$ et q' est gagnant ; faire une transition contrôlable c_i si la garde renforcée le permet. Dans le cas de l'exemple de la figure 5.3 la stratégie la plus permissive est donnée dans le tableau 5.1. Il faut bien noter que pour obtenir une stratégie *non Zénon*, il ne faut pas choisir indéfiniment λ dans ℓ_0 (ou ℓ_1, ℓ_2), mais bien tirer c_1 à un certain point. De cette manière, on construit un automate temporisé C (donné à la figure 5.5) qui représente la stratégie la plus permissive f^*, et tel que[5] $G \times C$ satisfait la propriété de sûreté ϕ. Dans le cas de jeux d'atteignabilité, le calcul d'une stratégie gagnante peut être plus compliqué. Un algorithme correct pour cela est décrit dans [BOU 04b].

5.4. Stratégies « de Zénon »

Les algorithmes, basés sur le calcul de points fixes, que nous avons présentés dans la section 5.3 permettent de calculer l'ensemble des états gagnants et de synthétiser des stratégies gagnantes associées. Cependant, nous n'avons pas pris en compte l'effet

5. $G \times C$ représente le produit synchronisé de G et C.

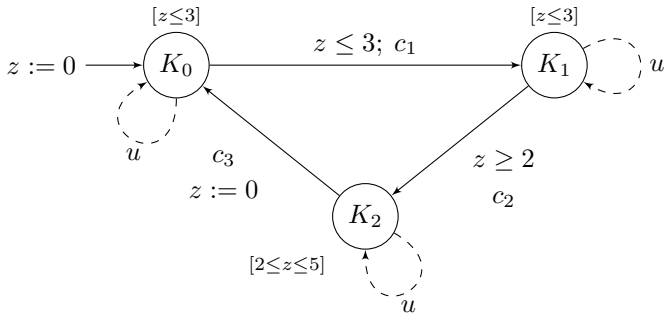

Figure 5.5. *Le contrôleur C le plus permissif*

« Zénon » : il se peut, et c'est le cas de l'exemple de la figure 5.6 où l'objectif est de ne jamais atteindre l'état BAD, qu'un jeu temporisé soit gagnant uniquement par des stratégies non réalistes. Dans cet exemple, la stratégie du contrôleur consiste à prendre

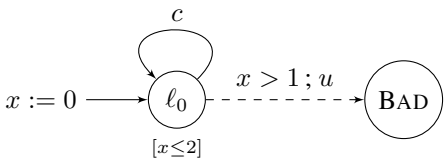

Figure 5.6. *Un jeu (avec objectif de sûreté) gagnant avec une stratégie de Zénon*

la transition c infiniment souvent avant que l'horloge x n'atteigne la valeur 1.

De manière symétrique, ce jeu n'admet pas de stratégie gagnante pour l'environnement pour forcer l'état BAD, puisque, quelle que soit la stratégie de l'environnement, celle-ci ne peut que proposer « attendre » tant que $x \leq 1$, et le contrôleur peut donc prendre la transition c aux dates $1 - 1/n$, produisant ainsi une exécution infinie n'atteignant jamais l'état BAD.

Nous verrons dans la section 5.10 une solution (assez technique) permettant de ne pas prendre en compte ces stratégies irréalistes dans les calculs de stratégies.

Ce genre d'anomalies justifie que l'on étudie les problèmes de contrôle, en particulier pour les objectifs de sûreté, de façon à construire des contrôleurs physiquement implémentables (et donc non Zénon). La section suivante est consacrée à cette notion.

5.5. Implémentabilité

Le SafCP a été formulé de manière différente dans plusieurs articles [ASA 98, HEN 99a, HEN 99b, MAL 95]. Dans l'article [CAS 02] on propose une classification de ces problèmes de contrôle suivant les critères suivants :

– contrôle en temps discret : peut-on contrôler le système avec un contrôleur qui effectue des actions à intervalles réguliers, tous les β unités de temps ? On parle alors de *Known Sampling Rate* (KSR) *control problem*. Une question plus générale consiste à trouver un pas d'échantillonage β tel que le système soit contrôlable. On parle alors de *Unknown Sampling Rate* (USR) *control problem* ;

– dans le cas du contrôle en temps dense on peut formuler des problèmes équivalents : si le contrôleur ne peut contrôler le temps qui passe mais juste choisir de faire une action ou d'attendre[6], on parle de *Known Switch Condition* (KSC) *control problem*. Dans le cas contraire (ce qui est le contexte des articles fondateurs [ASA 98, MAL 95]), on parlera de *Unknown Switch Condition* (USC) *control problem*.

La décidabilité du SafCP dépend de l'expressivité du modèle utilisé pour spécifier le système. Avant de donner les résultats de décidabilité, on introduit quelques classes d'automates qui étendent les TA.

5.5.1. *Classes d'automates hybrides*

La classe la plus générale considérée pour le SafCP est celle des *automates hybrides linéaires* (*Linear Hybrid Automata*, LHA) [HEN 96]. Le lecteur peut se référer au chapitre 9 pour une présentation détaillée de ces LHA. Cette classe étend les TA de la façon suivante : (i) les dérivées des variables[7] sont dans des intervalles du type $1 \leq \dot{x} \leq 3$; (ii) les gardes et les remises à jour sont des fonctions linéaires du type $2x + 3y \leq 4$ et $x := 3y - 7z$.

Une sous-classe intéressante des LHA est celle des *automates rectangulaires* (*Rectangular Automata*, RA) [HEN 99b]. Dans les RA les dérivées par rapport au temps des variables sont toujours contraintes par des prédicats du type $1 \leq \dot{x} \leq 3$ mais les remises à jour et les gardes ne permettent pas de comparer des variables entre elles : elles sont du type $x \geq 1 \wedge y < 2$ et les remises à jour du type $x :=]2, 3]$ pour affecter à x une valeur arbitraire entre 2 et 3. Cette classe a des propriétés de décidabilité intéressantes [HEN 99b] (décidabilité du problème de l'accessibilité d'une localité).

6. Et dans ce cas c'est l'environnement qui choisit la durée qui s'écoule.

7. On note $\dot{x}$ la dérivée de la variable x.

Une autre sous-classe des automates rectangulaires est celles des automates rectangulaires *initialisés* (*Initialized Rectangular Automata*, IRA) : si la dérivée d'une variable change entre deux localités ℓ et ℓ', alors cette variable est réinitialisée sur toute transition allant de ℓ et ℓ'.

Le problème de l'accessibilité est décidable pour la classe des RA mais pour toute surclasse il devient indécidable [HEN 98]. Enfin une dernière classe ayant des propriétés de décidabilité intéressante est celle des *automates o-minimaux* (*O-minimal Automata*, OminA) [LAF 00] : les dynamiques peuvent être riches (exponentielles par exemple ou une *théorie o-minimale*) mais toutes les variables doivent être remises à jour quand on franchit une transition discrète.

Les résultats concernant la décidabilité du problème de contrôle (safety) sont résumés dans le tableau 5.2. Les abréviations sont données par : KSC = *Known Switch Conditions controller*, USC = *Unknown Switch Conditions controller*, KSR = *Known Sampling Rate controller*, and USR = *Unknown Sampling Rate controller*. $\surd$ = décidable, $\times$ = indécidable.

Un résultat récent de [BOU 06b] est la décidabilité du RCP dans le cas USC, pour les automates o-minimaux (définis avec une théorie o-minimale décidable).

Description du système	KSC	USC	KSR	USR
TA	$\surd$ [MAL 95]	$\surd$ [MAL 95]	$\surd$ [HOF 92]	$\times$ [CAS 02]
IRA	$\surd$ [HEN 99a]	$\times$ [HEN 98]	$\surd$ [HEN 99b]	$\times$ [CAS 02]
RA	$\times$ [HEN 99a]	$\times$ [HEN 99a]	$\surd$ [HEN 99b]	$\times$ [CAS 02]

Tableau 5.2. *Existence de* safety *contrôleurs : résultats de décidabilité*

5.5.2. *Existence de contrôleurs continus non implémentables*

Dans [CAS 02] on montre que le problème USR-CP est non décidable même pour la classe des automates temporisés. Un autre résultat intéressant est qu'il existe des systèmes qui sont contrôlables par un contrôleur en temps dense (USC) mais tel qu'aucun pas d'échantillonage ne permet de les contrôler (USR). Nous avons donné un exemple d'automate temporisé (figure 5.7), pouvant être contrôlé avec un contrôleur non Zénon C. Dans cet exemple, toutes les transitions sont contrôlables. Pour contrôler le système (USC), il suffit de ne jamais tirer les transitions étiquetées d et pour que cela soit possible sans bloquer le système, on ajoute la contrainte (invariant) $[x < 1]$ dans la localité ℓ_2. Le contrôleur C doit faire les actions $(a.b.c)^\omega$ mais la durée entre le $n+1$-ième a et le $n+1$-ième b doit être strictement plus petite que la durée qui s'est écoulée entre le n-ième a et le n-ième b. Plus précisément, si la durée entre le

n-ième a et le n-ième b est δ_n, le contrôleur doit faire en sorte que $\sum_{i=1}^{+\infty} \delta_i \leq K$ où K est une constante inférieure à 1. Il est non Zénon car la n-ième action a est effectuée à la date n. Ceci montre que la notion de contrôle en temps dense n'implique pas que l'on puisse trouver un contrôleur en temps discret. En d'autres termes si l'on obtient comme réponse au USC-CP « oui », on n'a pas forcément un contrôleur *implémentable* sur une machine digitale.

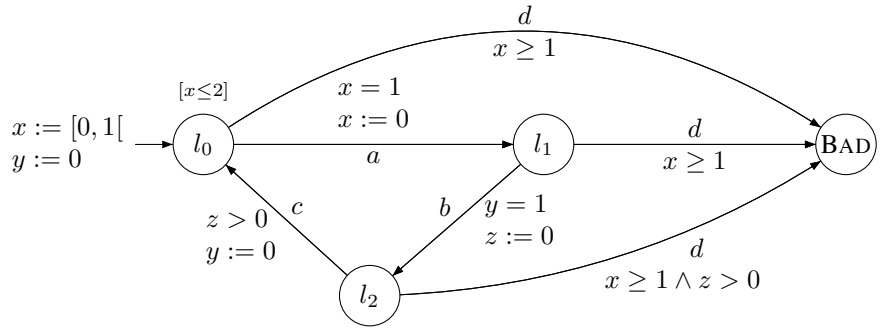

Figure 5.7. *Un système non contrôlable de manière échantillonée*

5.5.3. *Résultats récents et problèmes ouverts*

Le résultat principal d'indécidabilité de [CAS 02] peut être étendu aux intervalles de la forme $[\alpha, \beta]$ c'est-à-dire on impose qu'une action de contrôle ait lieu entre $[\alpha, \beta]$ unités de temps après la précédente. La preuve est décrite dans [DOY 07].

Ces résultats justifient donc que l'on s'intéresse à la notion de *contrôleurs implémentables* et ces problèmes ont été étudiés par Jean-François Raskin et son équipe dans les récents articles [DEW 04a, DEW 04b, DEW 05a, DEW 05b]. Une synthèse est proposée dans [ALT 05].

Un problème ouvert est la synthèse (directe) de contrôleurs implémentables ou robustes qui étendraient les résultats sur le *model checking* robuste [BOU 06f, BOU 08b].

5.6. Spécification des objectifs de contrôle

Dans les sections précédentes, on a considéré des propriétés de contrôle de type *safety* (ou *reachability*). Dans cette section, on va s'intéresser à faire varier les propriétés de contrôle que l'on souhaite forcer.

Dans [ALF 01], les auteurs donnent un algorithme pour la synthèse de contrôleurs avec des propriétés ω-régulières. Dans [ALF 03], ils considèrent des propriétés

ω-régulières sur les automates temporisés dans une sémantique avec surprise (voir section 5.10 de ce chapitre). Ils montrent que le CP est EXPTIME-complet dans ce cas. Ces propriétés sont appelées *internes* car elles font référence à des séquences d'états du système et donc à des séquences de localités qui n'incluent pas de notion de temps. On peut donc spécifier une propriété du type « à partir d'un état satisfaisant p on doit forcément passer par un état satisfaisant q », mais pas de propriétés du type « à partir d'un état satisfaisant p on doit forcément passer par un état satisfaisant q en moins de d unités de temps ». Les propriétés de *safety* et *reachability* sont des propriétés internes. Dans [FAE 02a], les auteurs montrent que le CP est 2EXPTIME-complet pour des spécifications données par des formules de LTL[8].

Une façon d'exprimer des propriétés de contrôle temporisées consistent à spécifier ces propriétés par des automates temporisés. Le langage[9] $\mathcal{L}(A)$ accepté par un automate temporisé décrit un ensemble de comportements qui peuvent être vus comme une spécification, les mauvais ou les bons comportements : on parlera alors de propriétés *externes*. On peut ainsi définir un automate temporisé A_d qui donnent les comportements permis du système, et un automate temporisés A_u qui donnent les comportements non permis. Si le système à contrôler est S, le problème de contrôle pour des spécifications externes (*External Specification Control Problem*, ESCP) est :

existe-t-il C tel que (1) $\mathcal{L}(C(S)) \cap \mathcal{L}(A_u) = \emptyset$ *et* (2) $\mathcal{L}(C(S)) \subseteq \mathcal{L}(A_d)$? [ESCP]

Dans l'article [DSO 02], les auteurs montrent que :

1) les sous-problèmes ESCP.(1) et ESCP.(2) ne sont pas décidables dans le cas où A_d et A_u sont des automates temporisés non déterministes ;

2) ces problèmes sont décidables pour A_d et A_u déterministes ; les automates temporisés déterministes sont complémentables et on peut en fait réduire ces problèmes à des problèmes de contrôle de *safety* (propriété interne) ;

3) le problème du contrôle ESCP.(2) est décidable (2EXPTIME-complet) si l'on fixe les ressources maximales du contrôleur (nombre d'horloges, granularité, constantes utilisées) et pour A_u un automate temporisé non déterministe. ESCP.(1) reste indécidable même dans le cas de ressources fixées.

Les propriétés exprimables par des automates temporisés constituent un apport dans l'expressivité des objectifs de contrôle. Cependant, on est passé des propriétés internes simples (*safety*, *reachability*) à des propriétés externes ω-régulières. Dans le cas non temporisé, on peut trouver un intermédiaire : les logiques temporelles, par exemple LTL qui forment une sous-classe des propriétés ω-régulières intéressantes.

---

8. Ce résultat n'est pas en contradiction avec [ALF 03] car la spécification est ici une formule de LTL et pas l'automate correspondant à cette formule.

9. Il s'agit de langages de mots infinis ou encore ω-réguliers.

Dans [BOU 06a], on considère une extension temporisée de la logique temporelle LTL qui est MTL [KOY 90]. Le CP est indécidable dans le cas général pour des propriétés de MTL mais devient décidable si l'on se restreint les ressources (nombre d'horloges, constantes utilisées dans les gardes) du contrôleur : dans ce cas le CP est non primitif récursif. Dans [FAE 02b], les auteurs utilisent TCTL sans égalité pour spécifier des objectifs de contrôle. Ils prouvent que l'existence d'un contrôleur pour une propriété de ce type peut être décidée en temps exponentiel pour les automates temporisés.

Dans [BOU 05b], on utilise la logique modale temporisée L_ν (en fait un sous-ensemble L_ν^{det} de L_ν), qui est un fragment du μ-calcul temporisé permettant de spécifier des propriétés de sûreté temporisée et donc aussi de *bounded safety*. On peut réduire le CP pour L_ν^{det} à un problème de *model checking* et montrer que ce problème de contrôle est EXPTIME-complet.

Enfin, il existe une logique temporelle spécialement conçue pour spécifier des propriétés sur des systèmes ouverts : *Alternating-time Temporal Logic* (ATL) [ALU 02]. Cette logique est une extension de CTL, dans laquelle les quantificateurs de chemin sont remplacés par des « quantificateurs de stratégie ». On peut ainsi spécifier des propriétés telles que « le joueur 1 a une stratégie lui permettant d'atteindre un état à partir duquel le joueur 2 n'aura plus la possibilité d'atteindre un état gagnant ».

Plusieurs travaux récents ont étendu la définition et l'étude de cette logique aux systèmes temporisés. On peut ainsi préciser qu'un joueur peut atteindre un état gagnant *en moins de cinq unités de temps*. Cette logique a été étudiée d'une part pour des systèmes « à temps discret » [LAR 06], où des algorithmes efficaces (en temps polynomial) sont proposés, et d'autre part pour les jeux « temps réel » [BRI 07, HEN 06a] (mais avec une sémantique différente de celle que nous avons étudiée jusqu'à maintenant ; voir section 5.10).

5.7. Contrôle optimal

Si l'on considère le *Reachability Control Problem* (RCP), on veut un contrôleur qui force le système à aller vers un état (ou une localité) particulier. On peut ensuite se poser la question d'y aller le plus vite possible. En ce sens on veut un contrôleur *optimal* qui va amener le système dans l'état voulu le plus vite possible. Ce problème de *time-optimal control* a été résolu dans [ASA 99].

5.7.1. *Automates temporisés à coûts.*

Quelques années plus tard en 2001, les automates temporisés ont été étendus avec des notions de *coûts* : cette extension a été nommée *priced timed automata* (PTA)

dans [BEH 01a, BEH 01b] et *weighted timed automata* (WTA) dans [ALU 01] mais il s'agit des mêmes modèles. Ces notions de coûts généralisent la notion de temps écoulé : le coût est une fonction linéaire du temps écoulé dans chaque localité, et on peut associer un coût entier à une transition discrète. Dans l'exemple du PTA $\mathcal{A}$ de la figure 5.8, les coûts sont indiqués avec le mot-clé $\mathbf{cost}$. Lorsque le coût est associé à une localité, il indique le *coût par unité de temps* quand on laisse passer le temps dans cette localité. Lorsqu'il est associé à une transition, il indique le *coût de franchissement* de cette transition. Supposons que l'on passe δ unités de temps à attendre dans ℓ_0, et que l'on franchisse c_1. L'invariant de ℓ_1 impose que le temps ne puisse pas s'écouler dans ℓ_1 et donc on va soit vers ℓ_2 où ℓ_3. Supposons que l'on aille vers ℓ_2 et que l'on attende δ' unités de temps avant de tirer c_2. Dans ce cas le coût de cette exécution est $5 \cdot \delta + 10 \cdot \delta' + 1$. Dans les PTA toutes les transitions sont contrôlables.

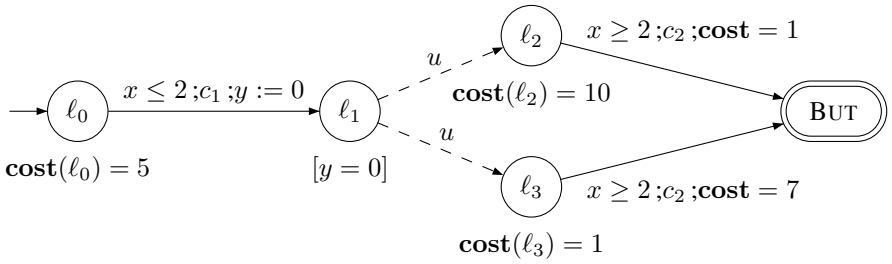

Figure 5.8. *Le PTGA $\mathcal{A}$*

La solution au problème du calcul du coût optimal pour un PTA, pour atteindre une localité particulière a été donnée en même temps dans deux articles [ALU 01, BEH 01b]. Elle a été implémentée dans une extension de l'outil UPPAAL qui s'appelle UPPAAL-CORA [BEH 07b]. Il faut bien remarquer que dans les PTA, il n'y a qu'un seul joueur et on n'a pas encore un problème de contrôle optimal.

On peut considérer les PTA (un seul joueur) et vouloir optimiser le coût sur les exécutions infinies. Ce problème implique que l'on définisse des *coûts* et des *récompenses*, et le but est de minimiser la limite du ratio coût/récompense sur les exécutions infinies. Une solution à ce problème a été résolu dans [BOU 04a, BOU 08a].

Enfin, on peut aussi définir des logiques intégrant cette notion de coût qui généralisent TCTL : il s'agit de *Weighted*-CTL (WCTL) introduite dans [BRI 04]. Avec cette logique on peut écrire des formules du type « est-il possible d'atteindre un état q avec un coût inférieur à K ». Le problème du *model checking* de WCTL pour les PTA est indécidable (même dans le cas du temps discret). Cependant pour une restriction de WCTL on peut identifier des classes décidables de PTA. Plus récemment dans [BOU 07] les auteurs montrent que WCTL est décidable pour les PTA

o-minimaux. Formellement un jeu temporisé avec coût est défini de la manière suivante :

DÉFINITION 5.7.1 (JEU TEMPORISÉ À COÛT) *Un* jeu temporisé à coût *(PTGA) est un couple* $(\mathcal{G}, \mathbf{cost})$ *où* $\mathcal{G} = (L, \ell_0, X, \Sigma_1, \Sigma_2, E, Inv)$ *est un automate temporisé de jeu et* $\mathbf{cost}$ *est une une* fonction de coût *telle que :* $\mathbf{cost} : L \cup E \to \mathbb{N}$*, c'est-à-dire* $\mathbf{cost}$ *associe à chaque localité un coût par unité de temps et à chaque transition un coût de franchissement.*

Soit $\mathcal{T}_\mathcal{G} = (Q, Q_0, \Sigma_1, \Sigma_2, \Gamma, \delta)$ la sémantique de $\mathcal{G}$. Si ρ définie par :

$$\rho = (\ell_0, v_0) \xrightarrow{(\delta_1, e_1)} (\ell_1, v_1) \xrightarrow{(\delta_2, e_2)} \dots \xrightarrow{(\delta_n, e_n)} (\ell_n, v_n)$$

est une exécution de $\mathcal{T}_\mathcal{G}$ on peut définir le coût de ρ par :

$$\mathbf{Cost}(\rho) = \sum_{i=0}^{n-1} \big(\mathbf{cost}(\ell_i) \times \delta_{i+1} + \mathbf{cost}(\ell_i \xrightarrow{e_i} \ell_{i+1})\big).$$

Le coût d'une exécution est donc la somme des coûts des transitions discrètes augmentée de la somme des coûts des transitions continues.

5.7.2. *Coût optimal dans les jeux temporisés*

Maintenant on peut poser le problème du calcul du coût optimal pour les *jeux* temporisés avec coût. On considère ici des jeux d'atteignabilité (RCP). Le but est de calculer le meilleur coût que l'on peut assurer quel que soit le comportement de l'adversaire. Un exemple de PTGA est donné par l'automate $\mathcal{A}$ de la figure. 5.8.

Les seules transitions incontrôlables sont les transitions d'étiquettes u. Il est aussi évident que le RCP a une solution dans $\mathcal{A}$: on tire c_1 puis c_2 et on est sûr de pouvoir forcer BUT. Dans la suite on suppose que l'on a déjà prouvé que le RCP avait une réponse positive et donc on peut s'intéresser au calcul du coût optimal.

Ce coût est défini comme étant la « valeur minimale » du coût que le contrôleur peut garantir. Par exemple, sur l'automate $\mathcal{A}$ de la figure 5.8, il y a deux exécutions possibles et l'unique choix du contrôleur est la date δ à laquelle il va tirer c_1 (car une fois dans ℓ_2 ou ℓ_3 il doit forcément tirer de suite c_2 pour minimiser le coût). Les valeurs des coûts engendrées par les exécutions passant par $\ell_i, i = 2, 3$ sont respectivement $\alpha_2(\delta) = 5{\cdot}\delta{+}10{\cdot}(2{-}\delta){+}1$ et $\alpha_3(\delta) = 5{\cdot}\delta{+}1{\cdot}(2{-}\delta){+}7$. Lorsque le contrôleur a tiré c_1 à la date δ, l'environnement agit comme un adversaire et va essayer de maximiser le coût : il va donc choisir la localité $\ell_i, i = 2, 3$ pour laquelle $\alpha_i(\delta)$ est maximal.

Le but du contrôleur est de choisir δ de façon à minimiser le maximum sur ces deux chemins.

On obtient donc comme définition du coût optimal pour $\mathcal{A}$:

$$\mathbf{OptCost} = \inf_{0 \leq \delta \leq 2} \max(5 \cdot \delta + 10 \cdot (2 - \delta) + 1, 5 \cdot \delta + 1 \cdot (2 - \delta) + 7).$$

Dans le cas de $\mathcal{A}$ cette valeur est $\frac{43}{3}$ et il faut tirer c_1 à la date $\frac{4}{3}$. Le problème qui consiste à déterminer le coût optimal pour un PTGA pour un objectif d'atteignabilité est le *Cost Optimal Reachability Control Problem* (CO-RCP).

On peut généraliser la définition du coût optimal $\mathbf{OptCost}$ donnée pour l'automate $\mathcal{A}$ aux PTGA. On retrouve le problème du *time optimal control* en prenant un coût unitaire par unité de temps dans chaque localité et un coût nul pour le franchissement d'une transition.

Soit $\mathsf{WinStrat}(q, \mathcal{G})$ l'ensemble des stratégies gagnantes à partir de q dans $\mathcal{G}$. Etant donnée une stratégie s pour le contrôleur, on peut définir le coût de s à partir d'un état q de $\mathcal{T}_{\mathcal{G}}$ par :

$$\mathbf{Cost}(q, s) = \sup \{\mathbf{Cost}(\rho) \mid \rho \in \mathrm{Out}^m(q, f)\}.$$

DÉFINITION 5.7.2 (COÛT OPTIMAL) *Soit $(\mathcal{G}, \mathbf{cost})$ un PTGA. On considère le RCP pour $\mathcal{G}$ avec la localité à forcer qui est* BUT. *L'*ensemble des coûts possibles, $\mathbf{Cost}(q)$, *à partir de q dans $\mathcal{G}$ est défini par :*

$$\mathbf{Cost}(q) = \{\mathbf{Cost}(q, s) \mid s \in \mathsf{WinStrat}(q, \mathcal{G})\}.$$

Le coût optimal à partir de q *est* $\mathbf{OptCost}(q) = \inf \mathbf{Cost}(q)$. *Le* coût optimal pour $\mathcal{G}$, $\mathbf{OptCost}(\mathcal{G})$, *est* $\sup_{q \in Q_0} \mathbf{OptCost}(q)$.

REMARQUE 5.7.1 *On considère le supremum à partir des états initiaux de $\mathcal{G}$ ce qui indique que l'on suppose que le choix de l'état initial est incontrôlable. S'il n'y qu'un seul état initial (comme dans les exemples qui suivent)* $\mathbf{OptCost}(\mathcal{G}) = \mathbf{OptCost}(q_0)$.

Le problème du coût optimal pour les jeux temporisé d'atteignabilité (*Cost Optimal Reachability Control Problem*, CO-RCP) est donc le suivant :

étant donnés $(\mathcal{G}, \mathbf{cost})$, calculer $\mathbf{OptCost}(\mathcal{G})$. [CO-RCP]

5.7.3. *Calcul du coût optimal*

Une première solution au problème du calcul du coût optimal, dans les jeux temporisés avec coût, a été donnée pour les jeux acycliques [LAT 02]. Comme le système ne comporte pas de cycles on peut réduire le problème à un problème d'optimisation linéaire.

Si l'on s'intéresse au calcul du coût optimal pour les PTGA cycliques, le problème est plus complexe. Une solution au problème général du calcul du coût optimal pour les PTGA a été donnée simultanément par Alur *et al*. [ALU 04] et Bouyer *et al*. [BOU 04c, BOU 05a]. Dans [ALU 04], on trouve des résultats de complexité (borne inférieure) et dans [BOU 04c], un certain nombre de résultats de décidabilité pour les PTGA ainsi que des propriétés structurelles des stratégies qui permettent de gagner ces jeux. Une implémentation de [BOU 04c] à l'aide de HYTECH [HEN 97] est décrite dans l'article [BOU 05a].

On donne ici l'algorithme de [BOU 04c] pour calculer le coût optimal d'un PTGA. L'idée de la solution[10] est la suivante : le CO-RCP peut être reformulé en ces termes : « quelle est la quantité minimale de ressources (temps, carburant, etc) avec laquelle je dois commencer le jeu pour pouvoir forcer la localité BUT et ne pas avoir consommé toutes mes ressources ? ». Pour résoudre le CO-RCP, on peut donc procéder de la façon suivante :

1) on part d'un jeu $\mathcal{A}$ du type de celui de la figure 5.8 ;

2) on définit un jeu $H_{\mathcal{A}}$ qui est un jeu temporisé avec une variable particulière *rsrc* : on obtient ainsi un jeu temporisé qui est dans une classe particulière d'automates hybrides (LHA). Cette variable correspond à la quantité de ressources dont on dispose. Elle va donc décroître dans la localité ℓ de $H_{\mathcal{A}}$ avec un taux opposé à celui de la localité correspondant dans $\mathcal{A}$.

L'automate $H_{\mathcal{A}}$ associé à $\mathcal{A}$ est donnée sur la figure 5.9 (la notation $\dot{rsrc}$ dénote la dérivée de *rsrc* par rapport au temps). On peut ainsi poser le CO-RCP comme un problème du type RCP mais sur un *automate hybride linéaire*[11] (LHA) : la localité que l'on veut atteindre est BUT et on demande aussi d'y arriver avec une valeur de la variable *rsrc* positive ou nulle. Il suffit de résoudre le RCP pour le LHA $H_{\mathcal{A}}$ avec l'objectif de contrôle $(\text{BUT}, rsrc \geq 0)$. Si l'on sait calculer les états gagnants de ce jeu (hybride linéaire) on obtient en particulier les états gagnants initiaux de la forme (ℓ_0, Z) où Z est un polyèdre donnant les contraintes à respecter au départ pour gagner.

10. On suppose que l'on a auparavant prouvé qu'il y avait une stratégie gagnante permettant de forcer BUT dans le jeu et ainsi le coût optimal existe.

11. voir paragraphe 5.5.1

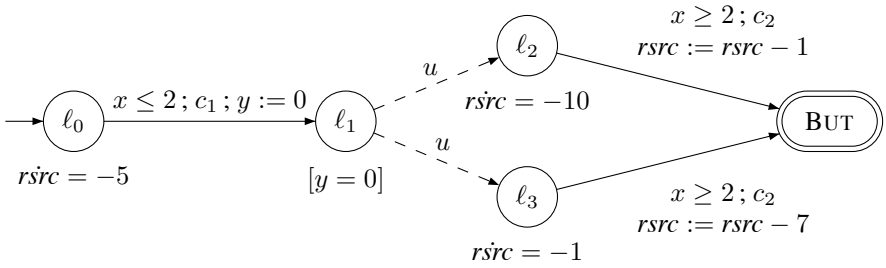

Figure 5.9. *Le LHA $H_{\mathcal{A}}$ associé à $\mathcal{A}$*

Par définition, Z est la zone maximale gagnante dans le sens où l'on ne peut gagner que si l'on part de Z. Si l'on projette Z sur la variable *rsrc* on obtient l'intervalle dans lequel doit être cette variable au départ pour pouvoir gagner c'est-à-dire atteindre BUT et ne pas avoir consommé toute la ressource *rsrc*. On peut montrer que cet intervalle est de la forme $rsrc \bowtie k$ avec $\bowtie \{>, \geq\}$ et donc par définition le coût optimal est k.

On peut ensuite étudier la décidabilité du problème de contrôle optimal en termes de décidabilité du RCP sur les automates hybrides linéaires du type de ceux engendrés par notre traduction. Les résultats obtenus par cette démarche sont les suivants :

– si l'on prend des PTGA bornés (toutes les horloges sont bornées) et que sur tout cycle de l'automate le coût d'un cycle est au moins $\beta > 0$ (on parle de *cost non-Zeno PTGA*), alors on peut calculer les états gagnants de l'automate hybride linéaire associé au PTGA de départ. Il suffit de prendre l'algorithme calculant itérativement les prédécesseurs contrôlables (en utilisant des polyèdres pour représenter les états symboliques) et cet algorithme termine ;

– on obtient le coût optimal en calculant la projection sur *rsrc* de la zone gagnante dans la localité initiale. On obtient toujours un intervalle du type $rsrc \bowtie k$ avec $\bowtie \{>, \geq\}$ et k est le coût optimal. On peut aussi décider s'il existe une stratégie permettant d'atteindre cette valeur optimale : si l'intervalle est de la forme $rsrc \geq k$ c'est « oui » ; si l'intervalle est de la forme $rsrc > k$ alors il existe une famille de stratégies permettant de réaliser des valeurs arbitrairement proches de k mais pas k. Il n'y a donc pas de stratégie optimale ;

– les stratégies optimales quand elles existent, peuvent nécessiter l'information correspondant au coût cumulé depuis que le jeu a commencé. En ce sens, les valeurs des horloges ne suffisent pas toujours car on peut avoir besoin d'une information supplémentaire de coût.

5.7.4. *Résultats récents et problèmes ouverts*

Depuis ces travaux il y a eu un certains nombres de résultats nouveaux :

– dans l'article [BRI 05], les auteurs montrent que sans l'hypothèse de *cost non-Zeno* le problème du calcul du coût optimal dans les PTGA devient indécidable. Ceci montre que cette hypothèse est nécessaire pour la terminaison de l'algorithme présenté plus haut. Cette preuve a été raffinée dans [BOU 06c] et le résultat le plus récent est que pour un PTGA à trois horloges le problème du calcul du coût optimal est indécidable ;

– dans [BOU 06e], les auteurs montrent que, pour les jeux temporisés à tour, on peut calculer le coût optimal pour les PTGA à une horloge (3EXPTIME) ;

– dans [BOU 07] il est montré que le coût optimal est aussi calculable pour les automates o-minimaux.

5.8. Algorithmes efficaces à la volée

Des outils efficaces ont été développés pour analyser les automates temporisés UPPAAL [AMN 01, AMN 07] ou KRONOS [YOV 97] mais jusqu'à récemment aucun d'eux ne permettait de faire de la synthèse de contrôleur. Ceci malgré le fait que des algorithmes sont connus depuis le milieu des années 1990.

Le succès des outils tels que UPPAAL repose sur une exploration *on-the-fly* de l'espace d'états (en plus bien entendu de structures de données adaptées et d'une implémentation efficace des algorithmes). Ces algorithmes *on-the-fly* ont aussi fait le succès de *model-checkers* de systèmes finis communicants comme SPIN [HOL 07].

L'algorithme de calcul du paragraphe 5.3.3 fonctionne en arrière en partant des états buts et en collectant les prédécesseurs contrôlables : on parle de *backward algorithm*. Un désavantage de cet algorithme est qu'il ne prend pas en compte l'espace d'états réellement atteignables. Il se peut donc que, lorsque l'on collecte en arrière les états gagnants, on obtienne beaucoup d'états (même symboliques) qui ne sont pas atteignables. Evidemment à chaque itération on calcule l'opérateur π en gardant ces états non atteignables.

Un problème plus gênant encore peut survenir avec des extensions simples de TGA : parfois on ne peut plus calculer l'opérateur π où le calculer devient très coûteux. Par exemple si les transitions contiennent des affectations de variables discrètes du type $i := 2j + k$ (ce qui est permis dans UPPAAL), à partir d'un état symbolique du type $(\ell, j \geq 0 \wedge k \geq 0)$, un calcul *backward* doit collecter toutes les valeurs i telles que $i = 2j + k$ et ce calcul peut s'avérer compliqué ou nécessiter des représentations complexes d'ensembles de valeurs entières. Or en pratique, si l'on exécute le système, on va certainement partir avec des valeurs particulières pour j et k et donc il n'y aura qu'un possibilité pour la valeur de i.

On peut dans un premier temps calculer l'espace d'états atteignables R. Ensuite à chaque itération de l'opérateur π dans le calcul du point fixe de l'ensemble des états

gagnants, on restreint les états gagnants collectés à cet ensemble R. Un désavantage est que l'on va calculer R, le stocker alors que ce n'est peut-être pas nécessaire pour décider si une stratégie gagnante existe.

5.8.1. *Algorithmes à la volée*

Une implémentation efficace dans l'esprit *on-the-fly* ne calculerait pas R mais devrait explorer l'espace d'états à la demande. Un algorithme *on-the-fly* a été proposé pour les automates temporisés par K. Altisen et S. Tripakis dans [ALT 99]. En fait cet algorithme consiste à calculer une abstraction du graphe des régions (qui peut être très grande) et à utiliser un algorithme *on-the-fly* sur l'automate fini ainsi obtenu. Il ne s'agit pas donc tout à fait d'un algorithme *on-the-fly* sur l'automate temporisé de départ car il nécessite de calculer une abstraction de l'espace de tous les états atteignables.

Pour éviter ce problème et obtenir un algorithme vraiment *on-the-fly* on peut se baser sur l'idée de Liu & Smolka [LIU 98] pour le *model checking on-the-fly* de LTL pour les automates finis.

Pour construire notre algorithme *on-the-fly* pour les TGA, on peut commencer par étendre l'algorithme de Liu & Smolka [LIU 98] au RCP dans le cas de systèmes finis (non temporisés).

L'idée d'un algorithme *on-the-fly* pour décider si l'on peut gagner un jeu (fini) d'atteignabilité $\mathcal{A}$ est la suivante : on suppose que l'on a deux ensembles `ToExplore` et `ToBackPropagate` dont les éléments sont des transitions de $\mathcal{A}$; on dispose aussi de l'ensemble R des états de $\mathcal{A}$ qui ont été explorés. Pour chaque état s de $\mathcal{A}$ qui a été exploré on maintient son *statut* qui est soit *gagnant* soit *indéterminé*.

Pour effectuer un pas dans l'algorithme *on-the-fly* on va prendre une transition (s, a, s') dans l'un de ces deux ensembles et faire soit (A1) si l'on a pris une transition de `ToExplore` et (A2) sinon :

– (A1) on explore (s, a, s') :

– si c'est la première fois que l'on rencontre s' on ajoute les transitions issues de s' à l'ensemble `ToExplore` ; si s' est un état de BUT on met son statut à *gagnant* sinon à *indéterminé* ;

– si le statut de s' est *gagnant* on ajoute ensuite (s, a, s') à `ToBackPropagate` ; sinon on retient que « le statut de s dépend de s' par (s, a, s') » ;

– (A2) on va propager en arrière de l'information pour la transition (s, a, s'). Si celle-ci est dans `ToBackPropagate` c'est que le statut de s' est devenu *gagnant*. On va propager cette information sur s :

– soit a est contrôlable et dans ce cas on incrémente un compteur $c(s)$ d'une unité. La signification de $c(s)$ est « $c(s)$ représente le nombre de transitions contrôlables qui peuvent amener dans un état gagnant à partir de s » ;

– soit a est incontrôlable : dans ce cas on incrémente un compteur $u(s)$. La signification de $u(s)$ est « $u(s)$ représente le nombre de transitions incontrôlables qui amènent dans un état gagnant » ;

– une fois mis à jour $c(s)$ ou $u(s)$, on met éventuellement à jour le statut de s. Soit $U(s)$ le nombre de transitions incontrôlables issus de s dans l'automate $\mathcal{A}$. Si $c(s) \geq 1$ et $u(s) = U(s)$ alors on peut déclarer s gagnant. Il faut dans ce cas planifier la mise à jour des *statut* des états qui dépendent de s. On ajoute donc à `ToBackPropagate` toutes les transitions (s'', a, s) telles que le *statut* de s'' dépend de s par (s'', a, s).

Un algorithme *on-the-fly* pour les jeux d'atteignabilité temporisés va réaliser dans un ordre aléatoire le traitement des transitions de `ToExplore` ou `ToBackPropagate`. Dès que le *statut* de l'état initial de $\mathcal{A}$ est gagnant on peut arrêter l'algorithme car on a la réponse au RCP (et on peut extraire une stratégie gagnante). On peut remarquer que cet algorithme fonctionne en temps linéaire en la taille du système. En effet, chaque transition est traitée au plus une fois quand elle est dans `ToExplore` et aussi au plus une fois quand elle est `ToBackPropagate`.

Pour adapter cet algorithme au cas temporisé on doit considérer des états *symboliques*, ou zones, qui sont des unions de régions. Etant donné un TGA et un état symbolique (ℓ, Z) on va explorer en avant une transition allant de la localité ℓ à ℓ' en calculant les successeurs discrets de Z par cette transition et en prenant le futur temporel (contraint par l'invariant de ℓ'). On maintient ensuite le *statut* des états symboliques.

La version de l'algorithme *on-the-fly* pour les TGA ainsi que la preuve de correction et de terminaison sont données dans [CAS 05]. La version temporisée de cet algorithme n'est pas linéaire en la taille du graphe des régions ! On utilise en effet des zones et une région peut être contenue dans plusieurs zones. Notre algorithme n'est donc pas optimal dans le sens où il existe un algorithme linéaire en la taille du graphe des régions [ALT 99]. Cependant il est *vraiment on-the-fly* et en pratique on obtient de bons résultats, comme le montre les expériences conduites avec la version « jeux » de UPPAAL qui s'appelle UPPAAL-TIGA [BEH 07a] et qui est présentée au chapitre 8.

5.8.2. *Résultats récents et problèmes ouverts*

Il est aussi possible de calculer des contrôleurs *time optimal* (les coûts associés aux localités sont tous 1 et les coûts des transitions discrètes sont 0) si l'on connaît une borne du temps nécessaire pour atteindre l'état BUT (voir [CAS 05]). L'algorithme

décrit dans cette section pour les jeux d'atteignabilité admet une version *duale* pour les jeux de sûreté.

Dans [CAS 07b], l'algorithme à la volée décrit ci-dessus a été adapté au cas de l'*observation partielle* (voir section 5.9 qui suit). Une synthèse concernant les algorithmes à la volée pour le cas temporisé est proposée dans [CAS 07a].

Les problèmes ouverts concernant cette section sont :

– étendre les algorithmes à la volée pour des objectifs autres que sûreté et accessibilité (par exemple accessibilité répétée). Ceci est très utile en pratique car de tels objectifs permettent de donner des contraintes de *non-zenoness* sur le contrôleur synthétisé ;

– étendre les algorithmes à la volée et structures de données pour obtenir des algorithmes efficaces pour le calcul du coût optimal (voir section 5.7).

5.9. Observation partielle

Dans les problèmes précédents, on a supposé que le contrôleur avait une vision parfaite de l'environnement : il peut observer tous les événements et l'état du système. Dans certains cas pratiques, le contrôleur ne voit pas tous les événements qui sont produits par l'environnement et il ne peut pas non plus observer l'état de l'environnement mais perçoit seulement un sous-ensemble des événements : les évènements *observables*. Malgré cela on voudrait pouvoir contrôler l'environnement : c'est le problème du contrôle sous observation partielle (*Control Under Partial Observation*, PO).

Pour ce problème on suppose toujours que les événements sont classés en deux classes, contrôlables (Σ_c) et incontrôlables (Σ_u) mais seulement un sous-ensemble Σ_u^o de Σ_u est observable. Le contrôleur va pouvoir observer une séquence d'événements temporisés $(a_0, t_0)(a_1, t_1) \cdots (a_n, t_n)$ de $((\Sigma_c \cup \Sigma_u^o) \times \mathbb{R}_{\geq 0})^*$ et doit déterminer à partir de cela les transitions contrôlables qu'il souhaite jouer.

En termes de stratégie, on dira qu'une stratégie est *trace-based* si pour deux exécutions ρ et ρ' ayant même trace (visible) temporisée[12] $f(\rho) = f(\rho')$.

Le problème de contrôle sous observation partielle (PO-CP) consiste à déterminer s'il existe une stratégie *trace-based* f qui permet de gagner. Un contrôleur est une stratégie et on peut donc parler de contrôleur *trace-based* (*Trace-Based Controller*, TBC) et définir le problème de contrôle sous observation partielle par :

étant donnés S et ϕ, existe-t-il un TBC C tel que $(S \parallel C) \models \phi$? [PO-CP]

12. La trace visible d'une exécution est un mot temporisé de $((\Sigma_c \cup \Sigma_u^o) \times \mathbb{R}_{\geq 0})^*$.

L'article [BOU 05c] propose une synthèse sur l'observation partielle dans les systèmes temporisés. Si la spécification est donnée par un automate temporisé déterministe, le PO-CP est indécidable [BOU 03]. Si l'on considère le problème plus contraint *safety* PO-CP (PO-SCP) et que l'on désire synthétiser un contrôleur qui ne génère que des comportements non-Zénon (Non-Zeno-PO-SCP), alors ce problème reste indécidable [BOU 06d].

En revanche, si l'on fixe les ressources du contrôleur, on peut étendre la preuve du résultat de [DSO 02] sur la synthèse de contrôleur pour des spécifications externes (voir section 5.6) au cas de l'observabilité partielle. On obtient ainsi : si l'on fixe les ressources du contrôleur et que l'objectif de contrôle, ϕ, est donnée par un automate fini déterministe, alors le PO-CP est EXPTIME-complet [BOU 03].

Des travaux plus récents [CHA 06, DEW 06] considèrent le problème de l'observation partielle des *états* d'un système pour les systèmes finis. L'extension au cas temporisé de ces travaux est proposée dans [CAS 07b].

5.10. Changeons les règles du jeu...

Comme nous l'avons mentionné lors de la définition des TGA, plusieurs sémantiques ont été définies pour les jeux temporisés. Nous allons présenter dans cette section une autre sémantique, issue de [ALF 03], dont le principal intérêt est qu'elle tient compte des stratégies de Zénon.

De manière informelle[13], la principale différence dans les règles du jeu est la suivante : au lieu d'avoir le choix entre attendre et jouer une transition d'action, les joueurs devront choisir simultanément la prochaine action qu'ils souhaitent jouer, ainsi que le délai qu'ils souhaitent laisser écouler avant de jouer cette action. Le reste du jeu est identique : le joueur qui souhaite faire son action le plus tôt est autorisé à l'appliquer, après avoir attendu le temps qu'il a choisi.

Bien que la différence soit mince, cette nouvelle sémantique a un avantage important : elle permet de savoir qui, du contrôleur ou de l'environnement, a choisi le délai le plus court et a ainsi pu appliquer son action. De cette façon, si le temps converge le long d'une exécution infinie, cette sémantique permet de détecter quel joueur (ça peut éventuellement être les deux) a fait converger le temps, en regardant quel joueur a pu infiniment souvent suivre la transition qu'il a choisie. De cette manière, pour chaque

13. Nous ne définirons pas la sémantique formelle de ces jeux dans ce chapitre, et renvoyons le lecteur intéressé aux articles [ALF 03, BRI 07, HEN 06b], qui introduisent et utilisent cette sémantique.

exécution de Zénon, on considère que le joueur qui a pu appliquer son choix infiniment souvent a triché ; il est disqualifié, et quel que soit l'objectif de jeu, l'autre joueur est déclaré vainqueur de la partie[14].

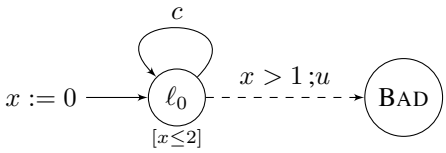

Figure 5.10. *Un jeu (avec objectif de sûreté) gagnant avec une stratégie de Zénon*

Reprenons l'exemple de la section 5.4, reproduit à la figure 5.10. Considérons la stratégie suivante : dans l'état (ℓ_0, x), avec $x < 1$, le contrôleur propose d'attendre un délai $\delta > 0$ tel que, après ce délai, l'horloge est de la forme $1 - 1/n$ (une telle valeur de δ existe toujours si $x < 1$), et de jouer l'action c après ce délai. Avec notre nouvelle sémantique, cette stratégie n'est pas gagnante : en effet, si l'environnement joue la stratégie « attendre 1 unité de temps et jouer l'action u », alors le contrôleur sera disqualifié et perdra la partie correspondante.

Dans ce cadre, l'existence de stratégies gagnantes sans mémoire n'est plus garantie : dans l'état initial $(\ell_0, 0)$ du jeu temporisé de la figure 5.10(a), le joueur 1 doit jouer à la date 0 l'action c : c'est la seule façon de remplir la condition d'accessibilité. En jouant de cette façon, l'exécution va revenir dans le même état $(\ell_0, 0)$. Si le joueur 1 applique sa stratégie sans mémoire, il va tout le temps proposer un délai nul, ce qui le disqualifiera.

Bien que les joueurs ne puissent plus gagner en faisant converger le temps, il leur est parfois nécessaire d'utiliser des stratégies « de Zénon » pour gagner. Ainsi, sur l'exemple de la figure 5.10(b), une stratégie gagnante (l'objectif étant d'atteindre ℓ_1) pour le joueur 1 consiste à jouer l'action c aux dates $1/2, 3/4, 7/8$, etc., c'est-à-dire d'attendre un délai $1/2^{n+1}$ entre le n-ième et le $n+1$-ième coup. Bien que cette stratégie puisse amener des exécutions de Zénon, elle est gagnante pour le joueur 1 : en effet, si la partie atteint ℓ_1, le joueur 1 jouera une stratégie différente dans ℓ_1, lui assurant de ne pas être disqualifié. Si le jeu n'arrive jamais dans ℓ_1, cela signifie que c'est toujours l'action du joueur 2 qui est prise, et ce joueur sera disqualifié.

Avec ces règles du jeu, la plupart des résultats que nous avons montré au cours de ce chapitre sont préservés. En particulier, on peut adapter la définition des prédécesseurs contrôlables, et montrer que les prédécesseurs contrôlables d'une zone forment

14. Il peut arriver que les deux joueurs soient disqualifiés, auquel cas il n'y a pas de gagnant.

(a) stratégie avec mémoire (b) stratégie avec « surprise »

Figure 5.11. *Exemples de jeux temporisés*

toujours une zone. De cette façon, les techniques de calcul de points fixes peuvent toujours s'appliquer, et il en résulte un algorithme en temps exponentiel.

Des travaux récents ont encore étendu ces modèles pour que, même si un joueur ne choisit pas le délai le plus court, il peut néanmoins influer sur le déroulement de la partie. Ces jeux sont des extensions des *concurrent game structures* [ALU 02]. Dans la version temporisée, les joueurs ne choisissent plus seulement un délai et une action, mais également les actions qu'ils souhaitent effectuer si leur adversaire choisit un délai plus court. La transition effectuée dépend alors de l'ensemble des actions proposées par les joueurs au moment où la transition doit être effectuée. Dans [BRI 07], les techniques que nous avons décrites précédemment sont adaptées à ce cadre.

5.11. Bibliographie

[ALF 01] DE ALFARO L., HENZINGER T.A., MAJUMDAR R., « Symbolic Algorithms for Infinite-State Games », *Proc. 12th International Conference on Concurrency Theory (CONCUR'01)*, vol. 2154 de *Lecture Notes in Computer Science*, p. 536-550, Springer, 2001.

[ALF 03] DE ALFARO L., FAËLLA M., HENZINGER T.A., MAJUMDAR R., STOELINGA M., « The Element of Surprise in Timed Games », *Proc. 14th International Conference on Concurrency Theory (CONCUR'03)*, vol. 2761 de *Lecture Notes in Computer Science*, p. 142-156, Springer, 2003.

[ALT 99] ALTISEN K., TRIPAKIS S., « On-the-Fly Controller Synthesis for Discrete and Dense-Time Systems », *Proc. World Congress on Formal Methods in the Development of Computing System (FM'99)*, vol. 1708 de *Lecture Notes in Computer Science*, p. 233-252, Springer, 1999.

[ALT 05] ALTISEN K., MARKEY N., REYNIER P.A., TRIPAKIS S., « Implémentabilité des automates temporisés », ALLA H., RUTTEN É. (dir.), *Actes du 5ème Colloque sur la Modélisation des Systèmes Réactifs (MSR'05)*, p. 395-406, Autrans, France, Hermès, octobre 2005.

[ALU 94] ALUR R., DILL D., « A Theory of Timed Automata », *Theoretical Computer Science*, vol. 126, n° 2, p. 183-235, Elsevier Science, 1994.

[ALU 01] ALUR R., LA TORRE S., PAPPAS G.J., « Optimal Paths in Weighted Timed Automata », *Proc. 4th International Workshop Hybrid Systems, Computation and Control (HSCC'01)*, vol. 2034 de *Lecture Notes in Computer Science*, p. 49-62, Springer, 2001.

[ALU 02] ALUR R., HENZINGER T.A., KUPFERMAN O., « Alternating-time temporal logic », *Journal of the ACM*, vol. 49, p. 672-713, ACM Press, 2002.

[ALU 04] ALUR R., BERNADSKY M., MADHUSUDAN P., « Optimal Reachability in Weighted Timed Games », *Proc. 31st International Colloquium on Automata, Languages and Programming (ICALP'04)*, vol. 3142 de *Lecture Notes in Computer Science*, p. 122-133, Springer, 2004.

[AMN 01] AMNELL T., BEHRMANN G., BENGTSSON J., D'ARGENIO P.R., DAVID A., FEHNKER A., HUNE T., JEANNET B., LARSEN K.G., MÖLLER O., PETTERSSON P., WEISE C., YI W., « UPPAAL – Now, Next, and Future », *Proc. Modelling and Verification of Parallel Processes (*MOVEP*2k)*, vol. 2067 de *Lecture Notes in Computer Science*, p. 99-124, Springer, 2001.

[AMN 07] AMNELL T., BEHRMANN G., BENGTSSON J., D'ARGENIO P.R., DAVID A., FEHNKER A., HUNE T., JEANNET B., LARSEN K.G., MÖLLER O., PETTERSSON P., WEISE C., YI W., « UPPAAL », 2007, http://www.uppaal.com.

[ARN 03] ARNOLD A., VINCENT A., WALUKIEWICZ I., « Games for synthesis of controllers with partial observation », *Theoretical Computer Science*, vol. 1, n° 303, p. 7-34, Elsevier Science, 2003.

[ASA 98] ASARIN E., MALER O., PNUELI A., SIFAKIS J., « Controller Synthesis for Timed Automata », *Proc. IFAC Symposium on System Structure and Control*, p. 469-474, Elsevier Science, 1998.

[ASA 99] ASARIN E., MALER O., « As Soon as Possible: Time Optimal Control for Timed Automata », *Proc. 2nd International Workshop Hybrid Systems, Computation and Control (HSCC'99)*, vol. 1569 de *Lecture Notes in Computer Science*, p. 19-30, Springer, 1999.

[BEH 01a] BEHRMANN G., FEHNKER A., HUNE T., LARSEN K.G., PETTERSSON P., ROMIJN J., VAANDRAGER F., « Efficient Guiding Towards Cost-Optimality in UPPAAL », *Proc. 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01)*, vol. 2031 de *Lecture Notes in Computer Science*, p. 174-188, Springer, 2001.

[BEH 01b] BEHRMANN G., FEHNKER A., HUNE T., LARSEN K.G., PETTERSSON P., ROMIJN J., VAANDRAGER F., « Minimum-Cost Reachability for Priced Timed Automata », *Proc. 4th International Workshop Hybrid Systems, Computation and Control (HSCC'01)*, vol. 2034 de *Lecture Notes in Computer Science*, p. 147-161, Springer, 2001.

[BEH 07a] BEHRMANN G., COUGNARD A., DAVID A., FLEURY E., LARSEN K.G., LIME D., « UPPAAL-TiGA », 2007, http://www.cs.aau.dk/~adavid/tiga/.

[BEH 07b] BEHRMANN G., LARSEN K.G., RASMUSSEN J.I., « UPPAAL-CORA », 2007, http://www.cs.aau.dk/~behrmann/cora/.

[BOU 03] BOUYER P., D'SOUZA D., MADHUSUDAN P., PETIT A., « Timed Control with Partial Observability », HUNT JR W.A., SOMENZI F. (dir.), *Proc. 15th International Conference on Computer Aided Verification (CAV'03)*, vol. 2725 de *Lecture Notes in Computer Science*, p. 180-192, Boulder, États-Unis, Springer, juillet 2003.

[BOU 04a] BOUYER P., BRINKSMA E., LARSEN K.G., « Staying Alive As Cheaply As Possible », ALUR R., PAPPAS G.J. (dir.), *Proc. 7th International Conference Hybrid Systems, Computation and Control (HSCC'04)*, vol. 2993 de *Lecture Notes in Computer Science*, p. 203-218, Philadelphie, États-Unis, Springer, mars 2004.

[BOU 04b] BOUYER P., CASSEZ F., FLEURY E., LARSEN K.G., Optimal Strategies in Priced Timed Game Automata, BRICS Reports Series n° RS-04-0, BRICS, Aalborg, Danemark, février 2004, ISSN 0909-0878.

[BOU 04c] BOUYER P., CASSEZ F., FLEURY E., LARSEN K.G., « Optimal Strategies in Priced Timed Game Automata », LODAYA K., MAHAJAN M. (dir.), *Proc. 24th Conference on Fundations of Software Technology and Theoretical Computer Science (FSTTCS'04)*, vol. 3328 de *Lecture Notes in Computer Science*, p. 148-160, Chennai, Inde, Springer, décembre 2004.

[BOU 05a] BOUYER P., CASSEZ F., FLEURY E., LARSEN K.G., « Synthesis of Optimal Strategies Using HyTech », DE ALFARO L. (dir.), *Proc. Workshop on Games in Design and Verification (GDV'04)*, vol. 119 de *Electronic Notes in Theoretical Computer Science*, p. 11-31, Boston, États-Unis, Elsevier Science, février 2005.

[BOU 05b] BOUYER P., CASSEZ F., LAROUSSINIE F., « Modal Logics for Timed Control », ABADI M., DE ALFARO L. (dir.), *Proc. 16th International Conference on Concurrency Theory (CONCUR'05)*, vol. 3653 de *Lecture Notes in Computer Science*, p. 81-94, San Francisco, États-Unis, Springer, août 2005.

[BOU 05c] BOUYER P., CHEVALIER F., KRICHEN M., TRIPAKIS S., « Observation partielle des systèmes temporisés », ALLA H., RUTTEN É. (dir.), *Actes du 5ème Colloque sur la Modélisation des Systèmes Réactifs (MSR'05)*, p. 381-393, Autrans, France, Hermès, octobre 2005.

[BOU 06a] BOUYER P., BOZZELLI L., CHEVALIER F., « Controller Synthesis for MTL Specifications », BAIER C., HERMANNS H. (dir.), *Proc. 17th International Conference on Concurrency Theory (CONCUR'06)*, vol. 4137 de *Lecture Notes in Computer Science*, p. 450-464, Bonn, Allemagne, Springer, août 2006.

[BOU 06b] BOUYER P., BRIHAYE T., CHEVALIER F., « Control in o-Minimal Hybrid Systems », *Proc. 21st IEEE Symposium on Logic in Computer Science (LICS'06)*, p. 367-378, Seattle, États-Unis, IEEE Computer Society Press, août 2006.

[BOU 06c] BOUYER P., BRIHAYE T., MARKEY N., « Improved Undecidability Results on Weighted Timed Automata », *Information Processing Letters*, vol. 98, n° 5, p. 188-194, Elsevier Science, juin 2006.

[BOU 06d] BOUYER P., CHEVALIER F., « On the Control of Timed and Hybrid Systems », *EATCS Bulletin*, vol. 89, p. 79-96, European Association for Theoretical Computer Science, juin 2006.

[BOU 06e] BOUYER P., LARSEN K.G., MARKEY N., RASMUSSEN J.I., « Almost Optimal Strategies in One-Clock Priced Timed Automata », GARG N., ARUN-KUMAR S. (dir.), *Proc. 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, vol. 4337 de *Lecture Notes in Computer Science*, p. 345-356, Kolkata, Inde, Springer, décembre 2006.

[BOU 06f] BOUYER P., MARKEY N., REYNIER P.A., « Robust Model-Checking of Linear-Time Properties in Timed Automata », CORREA J.R., HEVIA A., KIWI M. (dir.), *Proc. 7th Latin American Symposium on Theoretical Informatics (LATIN'06)*, vol. 3887 de *Lecture Notes in Computer Science*, p. 238-249, Valdivia, Chili, Springer, mars 2006.

[BOU 07] BOUYER P., BRIHAYE T., CHEVALIER F., « Weighted O-Minimal Hybrid Systems are more Decidable than Weighted Timed Automata ! », ARTEMOV S.N. (dir.), *Proc. Symposium on Logical Foundations of Computer Science (LFCS'07)*, Lecture Notes in Computer Science, New-York, États-Unis, Springer, juin 2007.

[BOU 08a] BOUYER P., BRINKSMA E., LARSEN K.G., « Optimal Infinite Scheduling for Multi-Priced Timed Automata », *Formal Methods in System Design*, vol. 32, n° 1, p. 2-23, Kluwer Academic Publishers, février 2008.

[BOU 08b] BOUYER P., MARKEY N., REYNIER P.A., « Robust Analysis of Timed Automata *via* Channel Machines », AMADIO R. (dir.), *Proc. 11th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'08)*, vol. 4962 de *Lecture Notes in Computer Science*, p. 157-171, Budapest, Hongrie, Springer, mars 2008.

[BRI 04] BRIHAYE T., BRUYÈRE V., RASKIN J.F., « Model-Checking for Weighted Timed Automata », *Proc. Joint International Conferences on Formal Modelling and Analysis of Timed Systems (FORMATS'04) and Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'04)*, vol. 3253 de *Lecture Notes in Computer Science*, p. 277-292, Springer, 2004.

[BRI 05] BRIHAYE T., BRUYÈRE V., RASKIN J.F., « On Optimal Timed Strategies », *Proc. 3th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'05)*, vol. 3829 de *Lecture Notes in Computer Science*, p. 49-64, Springer, 2005.

[BRI 07] BRIHAYE T., LAROUSSINIE F., MARKEY N., OREIBY G., « Timed Concurrent Game Structures », *Proc. 18th International Conference on Concurrency Theory (CONCUR'07)*, vol. 4703 de *Lecture Notes in Computer Science*, p. 445-459, Springer, septembre 2007.

[CAS 02] CASSEZ F., HENZINGER T.A., RASKIN J.F., « A Comparison of Control Problems for Timed and Hybrid Systems », *Proc. 5th International Workshop on Hybrid Systems, Computation and Control (HSCC'02)*, vol. 2289 de *Lecture Notes in Computer Science*, p. 134-148, Springer, 2002.

[CAS 05] CASSEZ F., DAVID A., FLEURY E., LARSEN K.G., LIME D., « Efficient On-The-Fly Algorithms for the Analysis of Timed Games », ABADI M., DE ALFARO L. (dir.), *Proc. 16th International Conference on Concurrency Theory (CONCUR'05)*, vol. 3653 de *Lecture Notes in Computer Science*, p. 66-80, San Francisco, États-Unis, Springer, août 2005.

[CAS 07a] CASSEZ F., « Efficient On-the-Fly Algorithms for Partially Observable Timed Games », *Proc. 5th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'07)*, vol. 4763 de *Lecture Notes in Computer Science*, p. 5-24, Springer, 2007.

[CAS 07b] CASSEZ F., DAVID A., LARSEN K.G., LIME D., RASKIN J.F., « Timed Control with Observation Based and Stuttering Invariant Strategies », *Proc. 5th International Symposium on Automated Technology for Verification and Analysis (ATVA'07)*, vol. 4762 de *Lecture Notes in Computer Science*, p. 307-321, Springer, 2007.

[CHA 06] CHATTERJEE K., DOYEN L., HENZINGER T.A., RASKIN J.F., « Algorithms for Omega-Regular Games with Imperfect Information », *Proc. 20th International Workshop on Computer Science Logic (CSL'06)*, p. 287-302, 2006.

[DEW 04a] DE WULF M., DOYEN L., MARKEY N., RASKIN J.F., « Robustness and Implementability of Timed Automata », *Proc. Joint International Conferences on Formal Modelling and Analysis of Timed Systems (FORMATS'04) and Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'04)*, vol. 3253 de *Lecture Notes in Computer Science*, p. 118-133, Springer, 2004.

[DEW 04b] DE WULF M., DOYEN L., RASKIN J.F., « Almost ASAP Semantics: From Timed Models to Timed Implementations », *Proc. 7th International Workshop Hybrid Systems, Computation and Control (HSCC'04)*, vol. 2993 de *Lecture Notes in Computer Science*, p. 296-310, Springer, 2004.

[DEW 05a] DE WULF M., DOYEN L., RASKIN J.F., « Almost ASAP Semantics: From Timed Models to Timed Implementations », *Formal Aspects of Computing*, vol. 17, n° 3, p. 319-341, Springer, 2005.

[DEW 05b] DE WULF M., DOYEN L., RASKIN J.F., « Systematic Implementation of Real-Time Models », *Proc. International Symposium of Formal Methods Europe (FM'05)*, vol. 3582 de *Lecture Notes in Computer Science*, p. 139-156, Springer, 2005.

[DEW 06] DE WULF M., DOYEN L., RASKIN J., « A Lattice Theory for Solving Games of Imperfect Information », *Proc. 9th International Workshop Hybrid Systems, Computation and Control (HSCC'06)*, p. 153-168, 2006.

[DOY 07] DOYEN L., « Robust Parametric Reachability for Timed Automata », *Information Processing Letters*, vol. 102, n° 5, p. 208-213, Elsevier Science, 2007.

[DSO 02] D'SOUZA D., MADHUSUDAN P., « Timed Control Synthesis for External Specifications », *Proc. 19th International Symposium on Theoretical Aspects of Computer Science (STACS'02)*, vol. 2285 de *Lecture Notes in Computer Science*, p. 571-582, Springer, 2002.

[FAE 02a] FAËLLA M., LA TORRE S., MURANO A., « Automata-Theoretic Decision of Timed Games », *Proc. 3rd International Workshop on Verification, Model Checking, and Abstract Interpretation (VMCAI'02)*, vol. 2294 de *Lecture Notes in Computer Science*, p. 240-254, Venise, Italie, janvier 2002.

[FAE 02b] FAËLLA M., LA TORRE S., MURANO A., « Dense Real-Time Games », *Proc. 17th IEEE Symposium on Logic in Computer Science (LICS'02)*, p. 167-176, IEEE Computer Society Press, 2002.

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

[HEN 97] HENZINGER T.A., HO P.H., WONG-TOI H., « HYTECH: A Model-Checker for Hybrid Systems », *Journal on Software Tools for Technology Transfer*, vol. 1, n° 1-2, p. 110-122, Springer, 1997.

[HEN 98] HENZINGER T.A., KOPKE P.W., PURI A., VARAIYA P., « What's decidable about hybrid automata ? », *Journal of Computer and System Sciences*, vol. 57, p. 94-124, Elsevier Science, 1998.

[HEN 99a] HENZINGER T.A., HOROWITZ B., MAJUMDAR R., « Rectangular Hybrid Games », *Proc. 10th International Conference on Concurrency Theory (CONCUR'99)*, vol. 1664 de *Lecture Notes in Computer Science*, p. 320-335, Springer, 1999.

[HEN 99b] HENZINGER T.A., KOPKE P.W., « Discrete-time control for rectangular hybrid automata », *Theoretical Computer Science*, vol. 221, p. 369-392, Elsevier Science, 1999.

[HEN 06a] HENZINGER T.A., PRABHU V.S., « Timed Alternating-Time Temporal Logic », ASARIN E., BOUYER P. (dir.), *Proc. 4th International Conferences on Formal Modelling and Analysis of Timed Systems, (FORMATS'06)*, vol. 4202 de *Lecture Notes in Computer Science*, p. 1-17, Springer, septembre 2006.

[HEN 06b] HENZINGER T.A., PRABHU V.S., « Timed Alternating-Time Temporal Logic », *Proc. 4th International Conferences on Formal Modelling and Analysis of Timed Systems, (FORMATS'06)*, vol. 4202 de *Lecture Notes in Computer Science*, p. 1-17, Springer-Verlag, septembre 2006.

[HOF 92] HOFFMANN G., WONG-TOI H., « The input-output control of real-time discrete-event systems », *Proc. 13th Annual Real-time Systems Symposium*, p. 256-265, IEEE Computer Society Press, 1992.

[HOL 07] HOLTZMANN G., « SPIN », 2007, http://spinroot.com.

[KOY 90] KOYMANS R., « Specifying Real-Time Properties with Metric Temporal Logic », *Real-Time Systems*, vol. 2, n° 4, p. 255-299, 1990.

[LAF 00] LAFFERRIERE G., PAPPAS G., SASTRY S., « O-minimal hybrid systems », *Mathematics of Control Signals and Systems*, vol. 13, n° 1, p. 1-21, 2000.

[LAR 98] LAROUSSINIE F., LARSEN K.G., « CMC: A Tool for Compositional Model-Checking of Real-Time Systems », *Proc. IFIP Joint International Conference on Formal Description Techniques & Protocol Specification, Testing, and Verification (FORTE-PSTV'98)*, p. 439-456, Kluwer Academic, 1998.

[LAR 05] LAROUSSINIE F., « CMC », 2005, http://www.lsv.ens-cachan.fr/~fl/cmcweb.html.

[LAR 06] LAROUSSINIE F., MARKEY N., OREIBY G., « Model Checking Timed ATL for Durational Concurrent Game Structures », ASARIN E., BOUYER P. (dir.), *Proc. 4th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'06)*, vol. 4202 de *Lecture Notes in Computer Science*, p. 245-259, Paris, France, Springer, septembre 2006.

[LAT 02] LA TORRE S., MUKHOPADHYAY S., MURANO A., « Optimal-Reachability and Control for Acyclic Weighted Timed Automata », *Proc. 2nd IFIP International Conference on Theoretical Computer Science (TCS'02)*, vol. 223 de *IFIP Conference Proceedings*, p. 485-497, Kluwer, 2002.

[LIU 98] LIU X., SMOLKA S.A., « Simple Linear-Time Algorithm for Minimal Fixed Points », *Proc. 26th International Conference on Automata, Languages and Programming (ICALP'98)*, vol. 1443 de *Lecture Notes in Computer Science*, p. 53-66, Aalborg, Danemark, Springer, 1998.

[MAL 95] MALER O., PNUELI A., SIFAKIS J., « On the Synthesis of Discrete Controllers for Timed Systems », *Proc. 12th Annual Symposium on Theoretical Aspects of Computer Science (STACS'95)*, vol. 900 de *Lecture Notes in Computer Science*, p. 229-242, Springer, 1995.

[MAR 75] MARTIN D.A., « Borel determinacy », *Annals of Mathematics*, vol. 102, n° 2, p. 363-371, 1975.

[MCN 93] MCNAUGHTON R., « Infinite Games Played on Finite Graphs », *Annals of Pure and Applied Logic*, vol. 65, n° 2, p. 149-184, Elsevier Science, 1993.

[RAM 87] RAMADGE P.J., WONHAM W.M., « Supervisory Control of a Class of Discrete Event Processes », *SIAM Journal of Control and Optimization*, vol. 25, n° 1, p. 1202-1218, 1987.

[RAM 89] RAMADGE P.J., WONHAM W.M., « The Control of Discrete Event Systems », *Proceedings of the IEEE*, vol. 77, n° 1, p. 81-98, 1989.

[RIE 03] RIEDWEG S., PINCHINAT S., « Quantified Mu-Calculus for Control Synthesis », *Proc. 28th International Symposium on Mathematical Foundations of Computer Science (MFCS'03)*, vol. 2747 de *Lecture Notes in Computer Science*, p. 642-651, Springer, 2003.

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

[THO 95] THOMAS W., « On the Synthesis of Strategies in Infinite Games », *Proc. 12th Annual Symposium on Theoretical Aspects of Computer Science (STACS'95)*, vol. 900 de *Lecture Notes in Computer Science*, p. 1-13, Springer, 1995.

[YOV 97] YOVINE S., « KRONOS: A Verification Tool for Real-Time Systems », *Journal of Software Tools for Technology Transfer*, vol. 1, n° 1-2, p. 123-133, Springer, 1997.