Observation partielle des systèmes temporisés

Patricia Bouyer* — **Fabrice Chevalier***
Moez Krichen† — **Stavros Tripakis**†

* *LSV – ENS Cachan & CNRS*
61, avenue du Président Wilson
94235 CACHAN *Cedex, France*
e-mail : `{Patricia.Bouyer,Fabrice.Chevalier}@lsv.ens-cachan.fr`

† *VERIMAG – Centre Équation*
2, avenue de Vignates
38610 GIÈRES, *France*
e-mail : `{Moez.Krichen,Stavros.Tripakis}@imag.fr`

Article rédigé dans le cadre du projet CORTOS de l'ACI « Sécurité Informatique ».
Site web : `http://www.lsv.ens-cachan.fr/aci-cortos/`

RÉSUMÉ. *Dans cet article, nous présentons la contrainte d'observation partielle, contrainte qui apparaît naturellement lorsqu'il s'agit de modéliser des systèmes réels. Nous avons sélectionné trois problèmes où cette contrainte est fondamentale mais pose des problèmes spécifiques : le contrôle des systèmes temporisés, la détection d'erreurs et le test de conformité. Nous expliquons quelles méthodes peuvent être employées pour aider à résoudre ces problèmes.*

ABSTRACT. *In this paper, we present the partial observability constraint, which naturally appears when modeling real-time systems. We have selected three problems in which this hypothesis is fundamental but leads to more difficult problems: control of timed systems, fault diagnosis, and conformance testing. We describe methods which can be used for solving such problems.*

MOTS-CLÉS : *Systèmes temporisés, contrôle, test, observation partielle.*

KEYWORDS: *Timed systems, control, test, partial observation.*

1. Introduction

Analyse des systèmes temporisés. Ces 30 dernières années, vérifier formellement les systèmes informatiques s'est révélé être une nécessité et des problématiques telles que le model-checking, le test et la preuve formelle sont devenues des domaines de recherche très actifs. Au début des années 1990, plusieurs modèles ont été proposés, qui prennent en compte des contraintes quantitatives sur les délais séparant les événements, l'un des plus utilisés encore aujourd'hui étant le modèle des automates temporisés [ALU 94]. Depuis, plusieurs outils ont été développés (parmi lesquels UPPAAL [LAR 97] ou KRONOS [BOZ 98]) et ont permis de mener à bien de nombreuses études de cas.

Observation partielle : pourquoi et comment ? La notion d'observation partielle intervient naturellement dans de nombreux problèmes de vérification : un système réel est souvent plongé dans un environnement hostile, et celui-ci ne peut pas être complètement connu et maîtrisé. On suppose alors que le système ne connaît pas totalement son environnement et reçoit des informations de l'extérieur *via* des capteurs. On dit alors que le système *n'observe que partiellement* l'environnement qui l'entoure. D'autre part, le non-déterminisme qui intervient assez naturellement dans des modélisations d'applications peut être vu comme une observation partielle du système : lorsqu'une action est effectuée, il y a incertitude sur l'état dans lequel se trouve le système, et il faut être capable de tenir compte de cette incertitude pour vérifier le système. Dans le cadre des systèmes à événements discrets, l'observation partielle se traduit souvent par l'ajout d'actions silencieuses (ou ε-transitions si l'on reprend la terminologie de théorie des automates finis), et les techniques utilisées reposent souvent sur la déterminisation et la possibilité de supprimer les actions silencieuses dans les automates finis [REI 84, SAM 95, SAM 96, KUP 97b]. Pour les systèmes temps-réel, la contrainte d'observation partielle va être une vraie difficulté car toutes les techniques que l'on vient de citer ne sont pas utilisables dans ce cadre [BÉR 98, TRI 03].

Quelques problèmes où l'observation partielle joue un rôle. On vient d'expliquer brièvement pourquoi rajouter une hypothèse d'observation partielle pouvait être raisonnable pour modéliser des systèmes réels. Dans cet article, nous allons décrire plusieurs problèmes où intervient cette contrainte et présenterons brièvement les algorithmes et résultats existants.

– nous commencerons par considérer le problème du *contrôle* des systèmes temporisés sous observation partielle. Celui-ci consiste à guider un système de telle sorte qu'il vérifie une certaine propriété, et ce, quel que soit le comportement (observable et non observable) de l'environnement ;

– nous nous focaliserons ensuite sur un problème un peu plus simple que celui du contrôle, la *détection d'erreurs*, qui concentre cependant tous les problèmes liés à l'observation partielle. Ce problème consiste, étant donné un système partiellement observable, à détecter les erreurs qui ont lieu (les erreurs sont supposées ne pas être observables) ;

– nous terminerons par présenter le *test de conformité* où un système réel type « boîte noire » est mis en test par rapport à sa specification formelle. Le test est un problème assez similaire au problème du contrôle, dans le sens où le testeur joue un jeu contre le système sous test : le testeur « essaie » de prouver que le système sous test est non-conforme, alors que le système sous test « essaie » de prouver le contraire.

2. Contrôle sous observation partielle

Principe du contrôle sous observation partielle. Comme on l'a dit dans l'introduction, la notion d'observation partielle intervient naturellement dans les problèmes de contrôle : le système que l'on cherche à contrôler est plongé dans un environnement hostile et reçoit des informations sur celui-ci *via* des capteurs. Dans le cadre des systèmes à événements discrets, le contrôle a été largement étudié dans les cadres d'observation totale [GRä 02, THO 02] et d'observation partielle [REI 84, KUP 97a, KUP 99, ARN 03]. Dans le cadre temporisé, le contrôle a tout d'abord été étudié sous une hypothèse d'observation globale [ASA 98, ASA 99, ALT 99, CAS 02, ALF 03] : dans tous ces travaux, le contrôleur sait à n'importe quel moment dans quel état se trouve le système. L'ajout de la contrainte d'observation partielle dans les travaux sur le contrôle des systèmes temporisés n'est que récent.

L'approche du contrôle des systèmes temporisés sous observation partielle que nous allons présenter est celle qui a été proposée dans [BOU 03]. On se donne un système temporisé $\mathcal{S}$, et trois alphabets disjoints Σ_c, Σ_e^o et Σ_e^u. Les actions de Σ_c sont contrôlables (et observables), celles de Σ_e^o sont incontrôlables mais observables, enfin celles de Σ_e^u ne sont pas observables (et donc incontrôlables). Nous allons nous restreindre à des objectifs de contrôle simples (contrôle pour des propriétés de sûreté). Le problème est de déterminer s'il existe un contrôleur non restrictif et non bloquant (hypothèses courantes pour les contrôleurs) qui permet d'éviter les mauvais états d'un système quel que soit le comportement de l'environnement, et ce en n'ayant qu'une vision partielle des actions de l'environnement.

Exemple. Dans l'exemple qui suit, on prend $\Sigma_c = \{c_1, c_2\}$, $\Sigma_e^o = \emptyset$ et $\Sigma_e^u = \{u\}$. Sous l'hypothèse d'observation partielle, le contrôleur a accès à la suite des actions observables qui se sont produites : par exemple, si l'exécution réelle du sytème est $(c_1, 1.2)(u, 1.9)(c_2, 2.5)$, alors le contrôleur ne verra que la suite d'actions $(c_1, 1.2)(c_2, 2.5)$, il ne peut pas savoir si une action non-observable s'est produite ou non, et il devra prendre ses décisions uniquement à partir de cette observation.

Le système représenté sur la figure 1 est contrôlable pendant 2 unités de temps (en autorisant c_1 mais pas c_2) puis devient incontrôlable, car il devient impossible de savoir si le système se trouve dans l'état 1 ou 2, le contrôleur ne peut donc pas savoir s'il doit autoriser c_1 ou c_2. Notons que si l'action u était observable, le système serait contrôlable (tant que l'action u n'a pas eu lieu, le contrôleur autorise c_1, puis une fois que l'action u a eu lieu, il autorise c_2).

Indécidabilité du problème de contrôle sous observation partielle. Dans le cadre des systèmes à événements discrets, les problèmes de contrôle, que ce soit sous observation totale et sous observation partielle, peuvent être résolus algorithmiquement. Ce n'est pas le cas dans le cadre des systèmes temporisés où l'observation partielle entraîne un saut de complexité très important. En effet le contrôle temporisé sous observation totale est décidable [HEN 99], alors que sous observation partielle il devient indécidable [BOU 03], et ce même pour des objectifs de contrôle très simples comme de la sûreté ou de l'accessibilité. Ceci repose sur le fait que l'universalité des automates temporisés (c'est-à-dire savoir si un automate temporisé accepte tous les comportements temporisés) est un problème indécidable : en effet l'universalité peut être vue comme un cas particulier de contrôle temporisé sous observation partielle où le contrôleur correspond à un mot non reconnu par le système.

Restriction des ressources. Le problème général de contrôle temporisé sous observation partielle étant indécidable, certaines sous-classes de ce problème ont alors été considérées. Une restriction naturelle est de réduire les ressources dont dispose le contrôleur : on suppose que le contrôleur est un automate temporisé, et on ne lui autorise qu'un nombre fixé d'horloges et certaines constantes ; par exemple on se demande s'il est possible de contrôler le système en utilisant 2 horloges et les constantes $\{0, 1, 2, 3\}$. Fixer les ressources du contrôleur semble être une hypothèse raisonnable : en pratique on recherche des contrôleurs implémentables, il est donc intéressant de chercher des contrôleurs n'étant pas trop complexes ou requiérant une précision nettement supérieure à celle du système contrôlé.

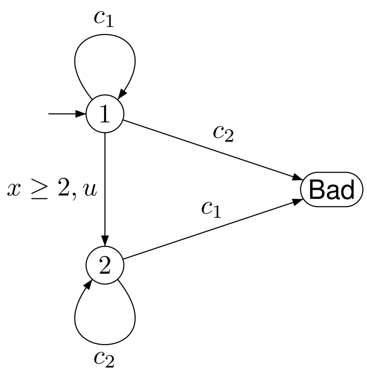

Figure 1. *Un système non contrôlable*

A ressources fixées, le problème de contrôle sous observation partielle devient décidable, mais reste plus complexe que le contrôle sous observation globale. La méthode de résolution consiste à utiliser une abstraction qui permet de se ramener à un jeu classique non temporisé à deux joueurs, un joueur correspondant à l'environnement et l'autre au contrôleur [BOU 03].

3. Détection d'erreurs

Principe de la détection d'erreurs. Un problème fortement lié à l'observation partielle est la détection d'erreurs : comme dans le cas du contrôle sous observation partielle, un système peut être observé uniquement *via* un ensemble de capteurs, de sorte que certaines actions ont lieu mais ne sont pas observables. En outre, le système peut avoir certains dysfonctionnements qu'il faut repérer le plus rapidement possible.

Dans le cadre des systèmes à événements discrets, ce problème a été largement étudié [SAM 95, SAM 96] et se réduit essentiellement à la déterminisation des automates finis avec des ε-transitions. Dans le cadre des systèmes temps-réel la déterminisation n'est pas possible en général. Par conséquent, d'autres techniques doivent être mises en place.

Le problème de la détection d'erreurs s'écrit comme suit : « Etant donné un système, est-il possible de détecter toute erreur au plus Δ unités de temps après qu'elle s'est produite ? ».[1] Plus particulièrement, on se donne le modèle du système sous la forme d'un automate temporisé et de deux alphabets Σ_o et Σ_u. Les actions de Σ_o sont observables et celles de Σ_u ne sont pas observables ; une action particulière de Σ_u notée f représente les erreurs qui ont lieu dans le système. Le but est alors, étant donnée une observation (une suite d'actions de Σ_o), de déterminer si une erreur a eu lieu dans le système ou non.

Exemples. Pour les deux exemples qui suivent, on considère $\Sigma_o = \{a\}$ et $\Sigma_u = \{u, f\}$. Dans le système représenté à la Figure 2, la détection d'erreurs n'est pas tou-

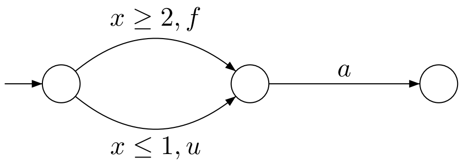

Figure 2. *Un système où l'erreur ne peut pas être détectée*

jours possible : sur l'observation $(a, 5)$, il n'est pas possible de savoir si une erreur a eu lieu ou pas, car cette observation peut provenir à la fois de l'exécution $(u, 1)(a, 5)$ (qui ne contient pas d'erreur) et de l'exécution $(f, 2)(a, 5)$ (qui contient une erreur).

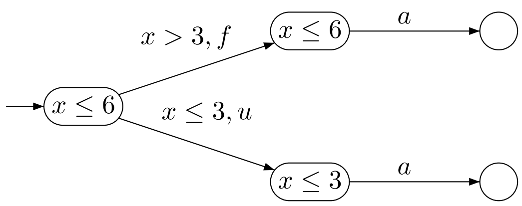

Figure 3. *Un système où on peut détecter les erreurs avec délai 3*

Dans le système représenté sur la Figure 3, il est possible de détecter les erreurs, avec un délai de détection qui vaut 3 unités de temps : pour une observation de la forme (a, t), il faudra annoncer qu'une erreur s'est produite si $t > 3$. Avec cette méthode, on

1. Dans le cas des systèmes non-temporisés, le passage du temps est modélisé par les actions discrètes, selon le principe «une action vaut une unité de temps».

peut toujours répondre avant $t = 6$ unités de temps et l'erreur se produit dans le pire cas un peu après $t = 3$, ce qui fait un délai de 3 unités de temps entre l'erreur et le moment où elle est annoncée.

Test de « détectabilité » [TRI 02]. Il est clair qu'il n'est pas toujours possible de détecter une erreur : si deux comportements du système sont identiques du point de vue de l'observateur, mais seul un des deux contient une erreur, alors il est impossible que cette erreur puisse être détectée. Par conséquent, avant de synthétiser un observateur (ou détecteur d'erreurs), il faut s'assurer que des situations ambiguës telles que celle décrite ci-dessus ne se produisent pas.

Dans ce but, nous employons un test de « détectabilité » qui consiste à identifier une paire de comportements ρ et ρ', tels que ρ est « erroné » (contient une erreur), ρ' ne l'est pas, et leurs projections à l'ensemble d'actions observable sont identiques. ρ et ρ' doivent être des comportements infinis et *non-zeno* (où le temps diverge), ce qui garantit que l'observateur ne pourra jamais les distinguer. Si deux tels comportements existent, le système n'est pas « détectable » (ou « observable »), sinon, il l'est. Pour tester l'existence d'une telle paire, il suffit de construire un produit spécial du modèle du système avec lui-même, où dans une des deux copies les erreurs sont supprimées. La synchronisation des deux copies se fait sur les actions observables uniquement (et le temps). Trouver dans l'automate-produit un comportement erroné, infini et non-zeno (qui correspond à une paire (ρ, ρ')) se réduit à un problème de test de langage vide d'automates temporisés de Büchi, pour lequel plusieurs solutions existent [ALU 94, TRI 05].

Une fois la « détectabilité » du système assurée, on peut aussi trouver par le moyen d'une recherche binaire le délai minimum Δ nécessaire pour la détection des erreurs (au pire cas). Ceci est fait en combinant l'automate-produit ci-dessus par un automate-moniteur qui vérifie que le délai Δ n'est pas dépassé avant que les deux comportements produisent des observations différentes.

Algorithmique, estimation des états accessibles [TRI 02]. Si le test de détectabilité réussit, on peut synthétiser automatiquement un observateur qui détecte les erreurs. Cet observateur n'est pas en général un automate temporisé (en effet, il ne peut pas toujours l'être) mais est une machine de Turing, qui effectue « à la volée » une sorte de déterminisation du système. Plus particulièrement, l'observateur calcule, étant donnée une observation, l'ensemble des états possibles dans lequel peut se trouver le système. On définit un opérateur $\mathsf{Post}_{(\delta,a)}(R)$ qui renvoie l'ensemble des états possibles du système après une attente de δ unités de temps et une observation a sachant que le système pouvait se trouver initialement dans l'ensemble R. En appliquant itérativement cet opérateur à chaque fois qu'un événement est observé, on réalise à la volée la détection d'erreurs.

Détection d'erreurs avec automates temporisés observateurs [BOU 05]. L'algorithmique par estimation d'état est la méthode la plus complète pour détecter les erreurs, elle est néanmoins assez coûteuse : calculer l'opérateur Post est coûteux et prend du temps. Cela est acceptable si on réalise une détection d'erreur *offline* (par exemple lorsque l'on recherche un problème dans un fichier de *logs*), mais peut être problématique quand on fait la détection d'erreur *online*, en temps réel. Une solution consiste à construire préalablement un détecteur d'erreurs sous la forme d'un automate temporisé déterministe. L'avantage de tels détecteurs est leur rapidité d'exécution : comme ils sont construits déterministes, ils ont juste à changer d'état à chaque nouvelle information reçue. Par contre, comme nous l'avons déjà mentionné, ces détecteurs ne sont pas exhaustifs : dans certains systèmes la détection d'erreurs peut être réalisée par estimation d'états tout en étant impossible par automate déterministe.

4. Test de conformité de systèmes temporisés

Le but de cette section est d'évoquer un autre domaine d'application où apparaissent les contraintes d'observabilité partielle, à savoir, le *test des systèmes temporisés*.

Introduction. Le test correspond à une étape importante dans la fabrication d'un grand nombre de systèmes aussi bien matériels que logiciels. Cette étape correspond à une étape de validation par laquelle l'utilisateur gagne plus de confiance en son système. Différents types de tests existent aujourd'hui.

Ici, on s'intéresse au test de conformité en boîte-noire pour les systèmes temporisés. Il s'agit de tester la « conformité » d'un système donné (*système sous test* ou SST) par rapport à son modèle (sa *spécification*). Dans notre cas, ce modèle est un automate temporisé. Nous considérons un type particulier d'automate temporisé : les *automates temporisés avec entrées et sorties* (ATES) [KRI 04a]. L'ensemble des actions d'un ATES donné est formé de deux sous-ensembles disjoints d'entrées (Σ_I) et de sorties (Σ_O). Les entrées sont les *stimuli* que le système reçoit de l'extérieur et les sorties les actions qu'il envoie à l'extérieur. Un exemple d'ATES est donné par la Figure 4. Cet ATES possède une seule entrée ($?a$) et deux sorties ($!b$ et $!c$). [2] Cet ATES spécifie le comportement suivant *"Après avoir reçu $a?$, le système doit produire soit $b!$ au bout de 3 à 5 unités de temps, soit $c!$ au bout de 4 à 6 unités de temps"*.

Comme pour les deux problèmes présentés dans les sections précédentes, l'observabilité partielle est une difficulté que l'on rencontre lorsque l'on fait du test. De nouveau, ceci est dû au fait que le SST utilise certaines actions internes invisibles depuis l'extérieur ou aussi au fait que le testeur présente certaines limitations au niveau de ses capacités d'observation (capteurs non suffisamment sophistiqués, par exemple). Ceci mène à enrichir le modèle d'ATES considéré. En plus des ensembles Σ_I et Σ_O, un ATES donné va posséder désormais un ensemble d'actions non observables, noté Σ_u (pour plus de simplicité, on prendra $\Sigma_u = \{\tau\}$). La Figure 5 donne l'exemple d'un ATES possédant des actions non observables. Il est à noter que, vu de l'extérieur, cet ATES est équivalent à celui de la Figure 4.

2. Les symboles « ? » et « ! » sont utilisés pour distinguer les entrées des sorties.

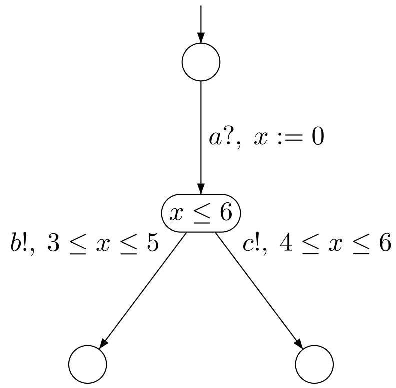

Figure 4. *Un exemple d'un ATES*

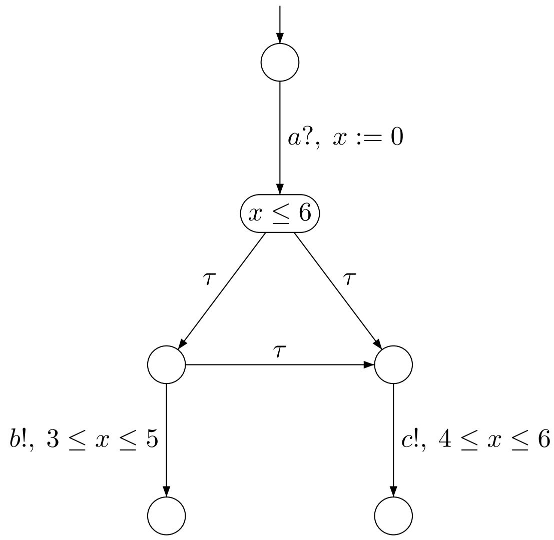

Figure 5. *Un ATES avec actions non observables*

La relation de conformité. Pour le cas des systèmes non temporisés, l'une des relations de conformité les plus utilisées est la relation $ioco$ [TRE 99]. La conformité par rapport à cette relation est vérifiée si le SST ne produit jamais plus de sorties que la spécification. La relation de conformité qu'on présente ici, appelée $tioco$ [KRI 04a], est une extension de $ioco$ pour le cas des systèmes temporisés. L'idée de base consiste à considérer les délais d'attente sans réponse comme de nouvelles sorties. Ainsi, le SST est considéré comme non conforme à son modèle s'il présente des délais plus importants que ceux permis par la spécification.

Par exemple, les ATES donnés par la Figure 6 sont deux implémentations non conformes à la spécification de la Figure 4. L'ATES se trouvant à gauche est non conforme parce qu'il produit la sortie $d!$ qui est non accéptée par la spécification. L'ATES se trouvant à droite est non conforme parce qu'il produit la sortie $c!$ dans des délais non permis par la spécification (pour $2 \leq x < 4$).

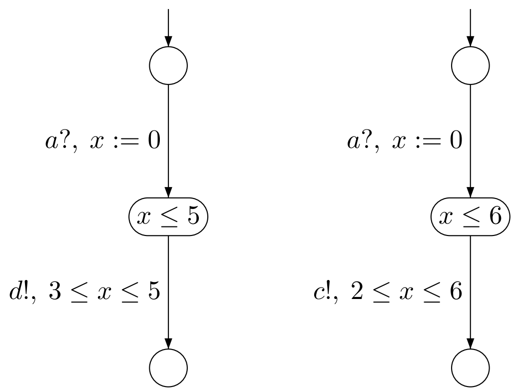

Figure 6. *Des implémentations possibles de l'ATES de la Figure 4*

Qu'est-ce qu'un test temporisé ? Un test temporisé correspond à une stratégie d'interaction entre le testeur et le SST. Suivant cette stratégie, le testeur sait, rigoureusement et à tout moment, quelle décision il doit prendre. Cette décision correspond à l'un des choix suivants :

- envoyer une entrée au SST ;
- laisser le temps passer ;
- émettre le verdict « Pass » (succès) ou « Fail » (échec).

Les tests considérés sont adaptatifs, dans le sens où ils sont en mesure d'évoluer en fonction des différentes réponses que le SST peut générer.

Comment représenter un test temporisé ? Il n'est pas possible de le faire sous forme d'arbres finis, comme pour le cas des tests non-temporisés. Ceci est dû au fait que le nombre des sorties que le SST peut produire est infini, car l'ensemble de délais possibles est infini (et même dense). Parfois, il est possible de représenter un test sous forme d'automate temporisé. Dans ce cas, l'exécution du test devient relativement plus simple (pour les mêmes raisons que celles évoquées dans la section précédente). L'ATES donné par la Figure 7 est un test possible pour la spécification donnée par la Figure 4. Cet ATES possède les nœuds particuliers Pass et Fail correspondant aux verdicts que le testeur pourrait émettre lors de l'exécution de ce test. Il est également à noter que les entrées et sorties au niveau du test sont inversées par rapport à la spécification (une sortie du testeur est une entrée de la spécification et vice versa).

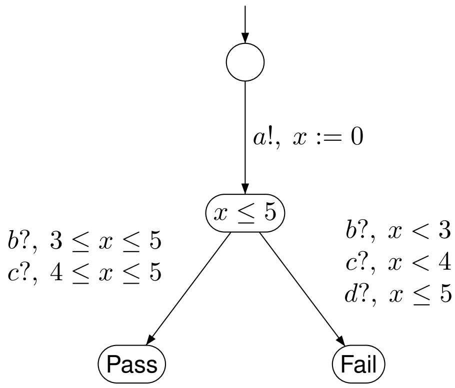

Figure 7. *Un test possible de l'ATES de la Figure 4*

Comment construire un test ? Comme dans le cas des détecteurs d'erreurs, il n'est pas toujours possible de représenter un test temporisé sous forme d'automate temporisé (rappelons que le test doit être déterministe, donc l'automate aussi). Par conséquent, dans le cas général, nous emploierons comme technique de base de génération de tests la technique d'estimation d'états. Ainsi, il est nécessaire de faire de nouveau appel à l'opérateur Post défini ci-dessus. Quand ce dernier renvoie l'ensemble vide comme résultat, alors il est à déduire que le chemin considéré correspond à un comportement non accepté par la spécification et qu'il va falloir déclarer Fail. Sinon si cet ensemble est non vide, alors il est possible de continuer le test ou de l'arrêter (tout en émettant Pass).

Ce qui est nouveau avec les techniques de génération de tests considérées ici correspond au fait que de temps à autre le générateur des tests aura à faire le choix entre sélectionner une entrée à exécuter ou lire les sorties possibles de la spécification. [3]

En fixant les ressources, comme cela a été fait pour le contrôle et la détection d'erreurs dans les sections précédentes, il est possible de synthétiser des automates temporisés déterministes pour le test également. Une technique pour faire cela de façon plus « symbolique », mais également moins précise, que la méthode basée sur le graphe des régions, est développée dans [KRI 04b].

L'observabilité partielle par rapport au temps. Il est clair, qu'en pratique, il n'y a pas de testeur qui soit capable d'observer le temps avec une précision infinie. Autrement dit, distinguer entre les durées $1s$ et $0.999s$ risque de ne pas être une tâche facile pour certains testeurs. Pour remédier à cette difficulté, une solution partielle consiste à discrétiser le temps au niveau de la spécification, autrement dit, prendre un modèle d'automates temporisés à temps discret. Cette solution présente un d'inconvénient majeur :

3. Ce sont ces différents choix possibles qui permettront de considérer une famille de tests et non pas un test unique.

la spécification s'en trouve modifiée par rapport aux contraintes d'implémentation du testeur. Or, ces deux aspects sont, et doivent le rester, clairement séparés.

Une meilleure façon pour contourner cette difficulté est proposée dans [KRI 05]. Elle consiste à garder une spécification temps-dense, mais inclure en même temps dans le modèle considéré une modélisation des imprécisions du testeur dans son observation du temps. Plus précisément, cette solution consiste à supposer que le testeur perd toute capacité de mesurer les délais et qu'il observe le temps seulement à travers l'exécution d'une action discrète particulière appelée $tick$. Cette action est supposée être produite par une horloge discrète dont on connaît le modèle. Ainsi, les traces considérées par le testeur sont de la forme "$tick \cdot a? \cdot tick \cdot b! \cdot c! \cdot tick \cdot d!''$ au lieu de "$(a?, 0.1) \cdot (b!, 1.1) \cdot (c!, 1.2) \cdot (d!, 2.2)''$. La technique de génération de tests est similaire à celle que pour les testeurs « idéaux » ci-dessus, sauf que l'opérateur $\mathsf{Post}_{(\delta,a)}(R)$ est remplacé par un opérateur $\mathsf{Post}_{(a)}(R)$, où a peut être l'action $tick$.

5. Conclusion

Dans cet article, nous avons expliqué pourquoi la contrainte d'observation partielle est une hypothèse importante. Nous avons notamment présenté trois problèmes importants où cette contrainte apparaît naturellement (le contrôle des systèmes temporisés, la détection d'erreurs, et le test de conformité) et expliqué quelles méthodes peuvent être employées pour résoudre ces problèmes. Une des méthodes couramment utilisées est l'*estimation d'états* où l'on calcule à la volée, étant donnée une séquence d'événements, l'ensemble des états dans lesquels le système peut se trouver. Ce calcul peut s'avérer être assez coûteux, des méthodes de construction d'observateurs déterministes offrent alors parfois des solutions un peu moins coûteuses.

Nous avons présenté trois domaines d'application de l'observation partielle, il en existe plusieurs autres, comme le domaine de l'apprentissage [GRI 04], ou le domaine du *runtime* model-checking [KRI 03].

6. Bibliographie

[ALF 03] ALFARO L. D., 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*, Springer, 2003, p. 142–156.

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

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

[ARN 03] ARNOLD A., VINCENT A., WALUKIEWICZ I., « Games for Synthesis of Controllers with Partial Observation », *Theoretical Computer Science*, vol. 1, n° 303, 2003,

p. 7–34.

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

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

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

[BOU 03] BOUYER P., D'SOUZA D., MADHUSUDAN P., PETIT A., « Timed Control with Partial Observability », *Proc. 15th International Conference on Computer Aided Verification (CAV'03)*, vol. 2725 de *Lecture Notes in Computer Science*, Springer, 2003, p. 180–192.

[BOU 05] BOUYER P., CHEVALIER F., D'SOUZA D., « Fault Diagnosis using Timed Automata », *Proc. 8th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'05)*, vol. 3441 de *Lecture Notes in Computer Science*, Springer, 2005, p. 219–233.

[BOZ 98] BOZGA M., DAWS C., MALER O., OLIVERO A., TRIPAKIS S., YOVINE S., « KRONOS : a Model-Checking Tool for Real-Time Systems », *Proc. 10th International Conference on Computer Aided Verification (CAV'98)*, vol. 1427 de *Lecture Notes in Computer Science*, Springer, 1998, p. 546–550.

[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 *LNCS*, Springer, 2002, p. 134–148.

[GRI 04] GRINCHTEIN O., JONSSON B., LEUCKER M., « Learning of Event-Recording Automata », *Proc. Joint Conference on Formal Modelling and Analysis of Timed Systems and Formal Techniques in Real-Time and Fault Tolerant System (FORMATS+FTRTFT'04)*, vol. 3253 de *Lecture Notes in Computer Science*, Springer, 2004, p. 379–395.

[GRä 02] GRÄDEL E., THOMAS W., WILKE T., Eds., *Automata, Logics, and Infinite Games : A Guide to Current Research*, vol. 2500 de *Lecture Notes in Computer Science*, Springer, 2002.

[HEN 99] 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*, Springer, 1999, p. 320–335.

[KRI 03] KRISTOFFERSEN K. J., PEDERSEN C., ANDERSEN H. R., « Runtime Verification of Timed LTL using Disjunctive Normalized Equation Systems », *Proc. 3rd International Workshop on Runtime Verification*, Electronic Notes in Computer Science, Elsevier, 2003.

[KRI 04a] KRICHEN M., TRIPAKIS S., « Black-Box Conformance Testing for Real-Time Systems », *Proc. 11th International SPIN Workshop (SPIN'04)*, vol. 2989 de *Lecture Notes in Computer Science*, Springer, 2004, p. 109–126.

[KRI 04b] KRICHEN M., TRIPAKIS S., « Real-Time Testing with Timed Automata Testers and Coverage Criteria », *Proc. Joint Conference on Formal Modelling and Analysis of*

Timed Systems and Formal Techniques in Real-Time and Fault Tolerant System (FORMATS+FTRTFT'04), vol. 3253 de *Lecture Notes in Computer Science*, Springer, 2004, p. 134–151.

[KRI 05] KRICHEN M., TRIPAKIS S., « An Expressive and Implementable Formal Framework for Testing Real-Time Systems », *Proc. 17th IFIP International Conference on Testing of Communicating Systems (TESTCOM'05)*, vol. 3502 de *Lecture Notes in Computer Science*, Springer, 2005, p. 209–225.

[KUP 97a] KUPFERMAN O., VARDI M. Y., « Synthesis with Incomplete Information », *Proc. 2nd International Conference on Temporal Logic (ICTL'97)*, Kluwer, 1997, p. 91–106.

[KUP 97b] KUPFERMAN O., VARDI M. Y., « Weak Alternating Automata Are Not That Weak », *5th Israel Symposium on Theory of Computing and Systems (ISTCS'97)*, IEEE Computer Society Press, 1997, p. 147–158.

[KUP 99] KUPFERMAN O., VARDI M., « Church's Problem Revisited », *The Bulletin of Symbolic Logic*, vol. 5, n° 2, 1999, p. 245–263.

[LAR 97] LARSEN K. G., PETTERSSON P., YI W., « UPPAAL in a Nutshell », *Journal of Software Tools for Technology Transfer (STTT)*, vol. 1, n° 1–2, 1997, p. 134-152, Springer.

[REI 84] REIF J. H., « The Complexity of Two-Player Games of Incomplete Information », *Journal of Computer and System Sciences (JCSS)*, vol. 29, n° 2, 1984, p. 274–301.

[SAM 95] SAMPATH M., SENGUPTA R., LAFORTUNE S., SINNAMOHIDEEN K., TENEKETZIS D. C., « Diagnosability of Discrete Event Systems », *IEEE Transactions on Automatic Control*, vol. 40, n° 9, 1995, p. 1555–1575.

[SAM 96] SAMPATH M., SENGUPTA R., LAFORTUNE S., SINNAMOHIDEEN K., TENEKETZIS D. C., « Failure Diagnosis using Discrete Event Systems », *IEEE Transactions on Control Systems Technology*, vol. 4, n° 2, 1996, p. 105–124.

[THO 02] THOMAS W., « Infinite Games and Verification », *Proc. 14th International Conference on Computer Aided Verification (CAV'02)*, vol. 2404 de *Lecture Notes in Computer Science*, Springer, 2002, p. 58–64, Invited Tutorial.

[TRE 99] TRETMANS J., « Testing Concurrent Systems : A Formal Approach », *Proc. 10th International Conference on Concurrency Theory (CONCUR'99)*, vol. 1664 de *Lecture Notes in Computer Science*, Springer, 1999, p. 46–65.

[TRI 02] TRIPAKIS S., « Fault Diagnosis for Timed Automata », *Proc. 7th International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT'02)*, vol. 2469 de *Lecture Notes in Computer Science*, Springer, 2002, p. 205–224.

[TRI 03] TRIPAKIS S., « Folk Theorems on the Determinization and Minimization of Timed Automata », *Proc. 1st International Workshop on Formal Modeling and Analysis of Timed Systems (FORMATS'03)*, vol. 2791 de *Lecture Notes in Computer Science*, Springer, 2003, p. 182–188.

[TRI 05] TRIPAKIS S., YOVINE S., BOUAJJANI A., « Checking Timed Büchi Automata Emptiness Efficiently », *Formal Methods in System Design*, , 2005, À paraître.