Test d'automates temporisés

Amélie Staïner

encadrée par Nathalie Bertrand et Thierry Jéron,

équipe VerTeCs, INRIA.

4 juin 2010

Master informatique spécialité Recherche en informatique à Rennes 1.
Promotion ENS.

Table des matières

1 Introduction

Les systèmes temps-réel sont répandus et souvent critiques, c'est-à-dire qu'une erreur au cours de leur fonctionnement peut avoir des conséquences graves. Pour rendre leur utilisation plus sûre, il existe plusieurs méthodes telles que la vérification ou le test que l'on dispose ou non du modèle du système, ou encore le contrôle pour empêcher les comportements fautifs. Dans ce rapport, nous nous consacrons à la modélisation par automates temporisés qui est tout à fait adaptée aux systèmes temps-réel. Plus précisément nous approfondirons la question de la déterminisation des automates temporisés pour la génération de test.

Nous nous intéressons particulièrement au cas du test de conformité. On souhaite vérifier qu'un système « boîte-noire », une implémentation dont on ne peut observer que les entrées et sorties, est conforme à une spécification donnée. Pour essayer de répondre à cette question, on interagit avec l'implémentation. On génère des entrées et on observe les sorties, on détecte alors d'éventuelles erreurs de conformité par rapport à la spécification. Il y a deux grandes approches pour faire du test, *on-line* et *off-line*. Le test *on-line* consiste en la génération du test durant son exécution. Il implique la déterminisation à la volée car on a besoin de savoir en permanence, quelles actions sont possibles dans la spécification. À chaque étape, on calcule donc l'ensemble des états accessibles par la séquence d'actions en cours. Le test *on-line* de systèmes temps-réel est difficilement réalisable pour les spécifications non-déterministes pour des raisons évidentes de temps de calcul. D'autre part, le test *off-line* consiste en la génération préalable de cas de test, dans notre cas sous forme d'automates temporisés, puis leur exécution. Contrairement au test *on-line*, cette approche permet d'appliquer la même série de cas de test à plusieurs implémentations. On peut alors fournir une certification, par exemple, on peut générer un ensemble de cas de test que l'implémentation devra passer pour être validée. Pour la généralité des cas de test générés, il est nécessaire de se baser sur une spécification déterministe. Sinon, on ne génèrerait que des cas de test arborescents de hauteur finie. C'est pour cela que nous nous intéressons à la déterminisation des automates temporisés et à l'approximation déterministe préservant la conformité, c'est-à-dire telle que toute implémentation conforme à la spécification non-déterministe soit conforme à l'approximation.

Les automates temporisés sont un modèle introduit par Alur et Dill en 1994 [AD94], obtenu en munissant les automates finis d'horloges continues. Ce modèle est couramment utilisé en vérification. La décidabilité de l'accessibilité dans un automate temporisé est une propriété primordiale dans l'utilisation de ce modèle pour la vérification et pour le test. Tout comme les automates finis, les automates temporisés ne sont pas nécessairement déterministes, mais ils ne sont pas non plus déterminisables en général [AD94]. Quelques classes d'automates déterminisables sont présentées dans [BBBB09]. Pour la problématique de la vérification, il est essentiel pour l'expressivité d'autoriser la modélisation des spécifications par des automates non-déterministes. Malheureusement, pour la génération des cas de tests, le déterminisme est primordial pour l'efficacité des calculs. Dans de nombreux travaux, on se restreint à des classes d'automates temporisés déterminisables, quand ce n'est pas à des automates déterministes. Krichen et Tripakis proposent une solution à ce problème [KT09]. Ils présentent une méthode de sur-approximation déterministe des automates temporisés sur laquelle on pourrait générer les cas de tests au risque de perdre en précision. La méthode nécessite une hypothèse contraignante sur la spécification pour préserver la relation de conformité, mais cela permet d'élargir le champ d'application des outils de modélisation à des automates non-déterministes, ce qui est considérable. D'un autre côté, l'article [BBBB09] présente une heuristique de déterminisation. Lorsqu'elle termine, elle produit un déterminisé de l'automate en argument et elle termine sur plusieurs classes d'automates temporisés déterminisables.

Le principal résultat de ce stage a été de développer une approche par le jeu qui re-

couvre les deux méthodes actuelles de déterminisation. Cette approche est inspirée d'un travail sur les automates temporisés pour un tout autre problème : la diagnosticabilité à ressources fixées [BCD05]. Notre méthode produit un déterminisé si possible, une sur-approximation déterministe sinon, et elle est à la fois plus générale que la procédure [BBBB09] et plus précise que la sur-approximation [KT09].

Au début de ce stage, nous avions élargi le domaine d'application de [BBBB09] grâce à une méthode d'analyse statique déjà existante présentée dans [DY96]. Cette procédure diminue le nombre d'horloges d'un automate temporisé en considérant l'ensemble des horloges actives dans chaque localité. Cette analyse permet de lisser le comportement des horloges et de supprimer des comportements inutiles qui empêchaient la terminaison de la méthode [BBBB09]. L'algorithme introduit des transferts d'horloges sur les transitions, mais la procédure de déterminisation [BBBB09] s'adapte naturellement aux automates avec transferts. De plus, il existe une construction pour supprimer les transferts sans ajouter d'horloges [Bou09]. Nous avons illustré ce travail par des exemples étendant strictement le domaine de [BBBB09]. Ce travail, bien qu'intéressant, n'est pas présenté dans ce rapport car le domaine d'application de cette procédure améliorée reste inclus dans l'ensemble des automates temporisés que notre approche par le jeu déterminise exactement. Par ailleurs, cette analyse ne permet pas d'étendre le domaine d'application de notre approche de déterminisation qui est robuste aux comportements parasites.

Enfin, nous avons appliqué notre méthode de déterminisation à la problématique du test de conformité avec spécification sous forme d'automate temporisé. Nous avons tout d'abord proposé une formalisation pour la génération de cas de test à partir de spécifications déterministes. Une adaptation de notre approche pour sous-approximer sur les entrées et sur-approximer sur les sorties nous permet de fournir une approximation préservant la conformité. Grâce à cela, nous pouvons générer des cas de test corrects (ne produisant pas de faux négatifs) à partir d'une spécification non-déterministe. Nous avons ensuite présenté un modèle d'automates temporisés ouverts, c'est-à-dire d'automates temporisés capables d'observer des horloges d'un environnement et en particulier d'un autre automate temporisé. Nous avons alors utilisé ce modèle pour formaliser la notion d'objectif de test. Tout cela nous a permis de formaliser la génération de cas de test correct avec spécification et objectif *a priori* non-déterministes.

La suite de ce document se décompose en trois parties. La première partie est consacrée à la présentation des automates temporisés et des principaux résultats utiles pour les autres parties de ce rapport, en particulier relatifs au problème de la déterminisation des automates temporisés. Dans la seconde partie, on présentera notre approche par le jeu pour la déterminisation ainsi que ses propriétés. La dernière partie présentera la problématique du test et notre travail de formalisation pour la génération de test à partir d'une spécification *a priori* non-déterministe avec, éventuellement, un objectif pas nécessairement déterministe.

2 Théorie des automates temporisés

2.1 Introduction aux automates temporisés

Dans cette partie, nous allons tout d'abord donner un cadre formel à notre discussion en définissant une syntaxe et une sémantique précises des automates temporisés. Ensuite nous approfondirons la notion d'automate des régions d'un automate temporisé qui fournit une abstraction finie utile à l'analyse comportementale. Enfin nous discuterons la décidabilité de plusieurs problèmes tels que l'accessibilité dans un automate temporisé. Pour plus de détails sur les énoncés de cette partie voir [AD94].

2.1.1 Modèle des automates temporisés

Les automates temporisés entrées/sorties. Pour définir les automates temporisés on munit un automate fini d'une ou plusieurs horloges avançant toutes à la même vitesse constante. Sur notre exemple de la Figure 1, l'horloge est nommée x. $\{x\}$ sur une transition signifie que l'horloge x est réinitialisée durant cette transition. Il existe plusieurs variantes des automates temporisés, nous avons choisit de présenter un modèle couramment utilisé dans le domaine du test. L'alphabet représente les actions d'un système. On a besoin pour la modélisation de systèmes temps-réel d'une notion d'entrée et de sortie. L'alphabet est donc partitionné en deux ensembles : les actions d'entrées $Act_?$, les actions de sorties $Act_!$. Pour plus de détails, voir les références [KT09] et [ST08] et pour des exemples voir la Figure 1.

On peut ainsi modéliser des systèmes temps-réel et en particulier des systèmes simples tels qu'une machine à café ou un double clic sur une souris.

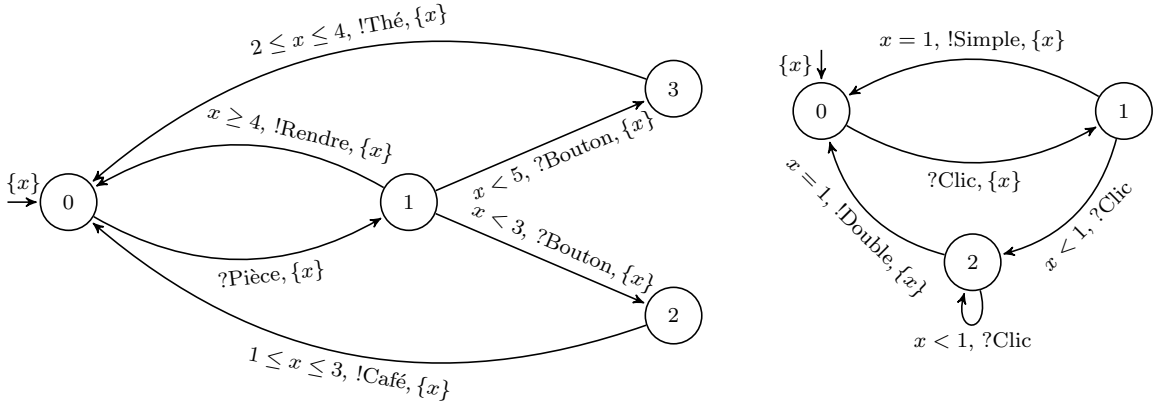


FIG. 1 – Exemples d'automates temporisés entrées/sorties.

Formalisons maintenant la notion d'automate temporisé :

Définition 1 (Syntaxe des automates temporisés entrées/sorties)**.** *Un* automate temporisé entrées/sorties *(TAIO pour* Timed Automaton with Inputs/Outputs*) est un tuple* $\mathcal{A} = (L, l_0, Act, X, M, E)$ *où :*

- L *est un ensemble fini de* localités,
- $l_0 \in L$ *est la* localité initiale,
- $Act = Act_? \cup Act_!$ *est un* alphabet fini *(entrées/sorties),*
- X *est un ensemble fini d'*horloges,
- M *est la constante maximale,*
- $E \subseteq L \times Guard_M^X \times Act \times 2^X \times L$ *est un ensemble d'*arêtes
  où $Guard_M^X = \{\bigwedge x \sim c \mid x \in X, c \in [0, M] \cap \mathbb{N}\}$ *avec* $\sim \in \{<, >, \leq, \geq, =\}$. *Pour* $(l, g, a, X', l') \in E$, *une* transition *de* $\mathcal{A}$ *:*
  - l *est la* source *et* l' *la* cible *de la transition,*
  - g *est une conjonction de contraintes (la* garde*),*
  - a *est* l'action.
  - X' *est l'ensemble des horloges réinitialisées lorsque l'on tire cette transition,*

Sémantique des automates temporisés entrées/sorties. Définissons tout d'abord les valuations d'horloges qui donnent la valeur de chaque horloge de l'automate considéré et les notations associées. Nous définirons ensuite le système de transitions qui donne la sémantique

des automates temporisés. Ce système est composé d'un ensemble d'états qui sont des couples constitués d'une localité et d'une valuation et de transitions entre états.

Étant donné X un ensemble fini d'horloges, une *valuation* v est un élément de $\mathbb{R}_+^X$: il associe à chaque horloge une valeur d'horloge. On note $\overline{0}$ la valuation qui associe 0 à toutes les horloges de X. Si v est une valuation sur X et $t \in \mathbb{R}_+$, alors $v+t$ est la valuation qui associe à chaque horloge $x \in X$ la valeur $v(x)+t$ et $\vec{v}$ est l'ensemble des valuations $v+t$ pour $t \in \mathbb{R}_+$. Si $X' \subseteq X$, on écrit $v_{[X' \leftarrow 0]}$ pour la valuation égale à v sur $X \setminus X'$ et $\overline{0}$ sur X' et on note $v_{|X'}$ la valuation v restreinte à X'. Étant donnée M un entier naturel et $g \in Guard_M^X$, on écrit $v \models g$ si v satisfait g et $[\![g]\!]$ pour l'ensemble des valuations satisfaisant g.

Définition 2. *La* sémantique *d'un TAIO* $\mathcal{A} = (L, l_0, Act, X, M, E)$ *est donnée par un* système de transitions $(Q,\ q_0,\ \Sigma,\ \rightarrow)$ *où :*

- $Q = L \times \mathbb{R}_+^X$ *est l'ensemble des* états *de* $\mathcal{A}$*,*
- $q_0 = (l_0, \overline{0})$ *est* l'état initial *de* $\mathcal{A}$*,*
- $\rightarrow \subseteq Q \times (\mathbb{R}_+ \cup Act) \times Q$ *est la* relation de transitions *associée. Elle est composée des* transitions discrètes $(l,v) \xrightarrow{a} (l',v')$ *où* $a \in Act$ *telles qu'il existe* $(l,g,a,X',l') \in E$ *avec* $v \models g$ *et* $v' = v_{[X' \leftarrow 0]}$ *et des* transitions de délai $(l,v) \xrightarrow{\theta} (l, v+\theta)$ *où* $\theta \in \mathbb{R}_+$.

On peut représenter par une seule transition $(l,v) \xrightarrow{\theta,a} (l',v')$ la concaténation d'une transition de délai θ et d'une transition discrète $(l, v+\theta) \xrightarrow{a} (l',v')$. Une *exécution* de $\mathcal{A}$ est une suite de transitions $q_0 \xrightarrow{\theta_1,a_1} (l_1,v_1) \xrightarrow{\theta_2,a_2} ... \xrightarrow{\theta_k,a_k} (l_k,v_k) \xrightarrow{\theta_{k+1}} (l_k, v_{k+1})$ partant de l'état initial q_0.

La notion d'état est plus précise que la notion de localité, elle prend en compte la valeur des horloges. Elle ne prend par contre pas en compte le temps absolu, on peut donc passer plusieurs fois par le même état.

Étant donné $tr \in (\Sigma \cup \mathbb{R}_+)^*$, on notera $time(tr)$ la somme des délais dans tr. On notera $Reach(\mathcal{A})$ l'ensemble des états accessibles d'un TA $\mathcal{A}$. On définit ci-dessous la notion de traces temporisées, c'est-à-dire l'ensemble des séquences d'actions observables sur $\mathcal{A}$.

Définition 3. *L'ensemble des* traces temporisées *d'un TAIO* $\mathcal{A}$ *noté* $traces(\mathcal{A})$*, est défini par :* $traces(\mathcal{A}) = \{tr \mid tr \in (\mathbb{R}_+.Act)^+.\mathbb{R}_+ \wedge\ q_0 \xrightarrow{tr}\}$.

$tr1 = 2.?Pièce.0.?Bouton.1.!Café.2$ et $tr2 = 5.?Clic.0, 3.?Clic.0, 2.?Clic.0, 5.!Double.3$ sont respectivement des exemples de traces des deux TAIO de la Fig. 1. On présente un autre exemple Fig. 2 où l'on donne le langage de traces de l'automate temporisé. On peut enrichir ce modèle par une notion d'urgence sur les transitions, on reviendra sur cette éventualité dans la section 4 sur le test. Pour cette partie théorique, nous considérerons le modèle simplifié (TA pour *timed automata*) où seul l'alphabet des sorties est utilisé : il est noté Σ. Les définitions et résultats donnés pour les TAs sont valables pour les TAIO.

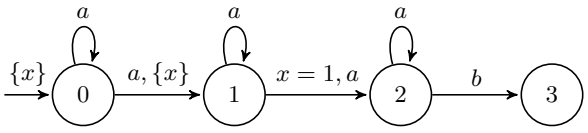


FIG. 2 – TA ayant pour traces $\{(t_i.a_i)_{1 \leq i \leq k}.t \in (\mathbb{R}_+.Act)^+.\mathbb{R}_+\} \cup \{(t_i.a_i)_{1 \leq i \leq k}.(t_b.b).t \in (\mathbb{R}_+.Act)^+.\mathbb{R}_+ |\ \exists i > 1, j > 1 \text{ tq } \Sigma_{i \leq k \leq j} t_k = 1\}$.

2.1.2 Régions

Une infinité de valuations distinctes peut vérifier exactement les mêmes gardes dans un TA. Cette observation va nous servir pour analyser les TA. On cherche à partitionner les valuations d'un ensemble fini d'horloges de manière à ce que sur deux valuations d'une même partie, exactement les mêmes transitions d'un TA soient tirables. Pour cela, définissons l'ensemble des régions standards d'un TA. Pour $t \in \mathbb{R}_+$, on note $\lfloor t \rfloor$ sa partie entière et $\{t\}$ sa partie fractionnaire.

Définition 4. *Étant donné $\mathcal{A} = (L, l_0, \Sigma, X, M, E)$ un TA,* l'ensemble des régions standards *de $\mathcal{A}$ est l'ensemble des classes de la relation d'équivalence $\equiv_M$ définie comme suit : $v \equiv_M v'$ si* $\forall (x, y) \in X$

- $v(x) > M \Leftrightarrow v'(x) > M$
- $v(x) \leq M \Rightarrow ((\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor)$ *et* $(\{v(x)\} = 0 \Leftrightarrow \{v'(x)\} = 0))$
- $(v(x) \leq M$ *et* $v(y) \leq M) \Rightarrow (\{v(x)\} \leq \{v(y)\} \Leftrightarrow \{v'(x)\} \leq \{v'(y)\})$

Un ensemble de régions standards dépend à la fois de la constante maximale M et du nombre d'horloges. On notera $\mathcal{R}_M$ cet ensemble en supposant le nombre d'horloges fixé.

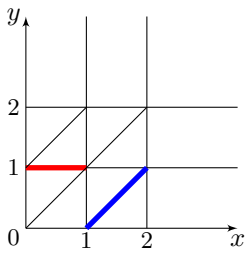


FIG. 3 – Ensemble des régions standards dans le cas $M = 2$ avec deux horloges.

Dans la Figure 3, les régions sont les intérieurs des polygones formés par les droites de la figure, les intersections des droites et les segments tels que le rouge et le bleu surlignés sur la figure. On peut remarquer que les régions sont nécessairement des parties connexes. Introduire la constante maximale en paramètre de ces régions standards nous permet de nous restreindre à un ensemble fini de régions.

Étant donné r un ensemble de valuations, on note $r_{[X' \leftarrow 0]} \subseteq \mathbb{R}_+^X$ l'ensemble des valuations $v_{[X' \leftarrow 0]}$ telles que $v \in r$. Voyons maintenant que la définition donnée a les propriétés souhaitées :

Proposition 1. *Soit $\mathcal{A} = (L, l_0, \Sigma, X, M, E)$ un TA. L'ensemble $\mathcal{R}_M$ de régions standards vérifie les propriétés suivantes :*

- $\forall g \in Guard_M^X$ *et* $\forall r \in \mathcal{R}_M$, $r \subseteq [\![g]\!]$ *ou* $[\![g]\!] \cap r = \emptyset$,
- $\forall (r, r') \in \mathcal{R}_M^2$ *si* $\exists v \in r$ *et* $t \in \mathbb{R}_+$ *avec* $v + t \in r'$ *alors* $\forall w \in r$, $\exists s \in \mathbb{R}_+$ *avec* $w + s \in r'$,
- $\forall (r, r') \in \mathcal{R}_M^2$, $\forall X' \subseteq X$ *si* $r_{[X' \leftarrow 0]} \cap r' \neq \emptyset$, *alors* $r_{[X' \leftarrow 0]} \subseteq r'$.

Autrement dit, elles vérifient ces trois propriétés :

- toutes les valuations d'une région vérifient exactement les mêmes gardes,
- toutes les valuations d'une région atteignent la même région en laissant s'écouler le temps (pas forcément autant),
- par toutes les projections $[X' \leftarrow 0]$, toutes les valuations d'une région se projettent sur la même région.

L'ensemble des régions standards d'un automate n'est pas la seule partition des valuations à avoir ces propriétés, mais les régions standards ont l'avantage d'êtres identiques pour tous les automates de même constante maximale.

Définition 5. *Étant données deux régions r et r', r' est un* successeur temporel *de r si $\exists (v,t) \in r \times \mathbb{R}_+$ tel que $v + t \in r'$.*

Automate des régions. Les automates temporisés possèdent une infinité d'états, ce qui rend leur analyse délicate. On montre ici que grâce à la notion de région, on peut construire une abstraction finie d'un automate temporisé, tout en préservant des propriétés intéressantes. En particulier, cette abstraction nous permettra de décider l'accessibilité d'une localité.

Définition 6. *Soit $\mathcal{A} = (L, l_0, \Sigma, X, M, E)$ un TA. L'*automate des régions *de $\mathcal{A}$, $\alpha(\mathcal{A})$, est un automate fini défini comme suit :*

- *l'ensemble des états est $L \times \mathcal{R}_M$,*
- *l'état initial est $(l_0, \overline{0})$,*
- *les transitions sont définies par : $(l,r) \xrightarrow{a} (l',r')$ si $\exists l \xrightarrow{g,a,X'} l'$ dans $\mathcal{A}$ et r'' successeur temporel de r avec $r'' \subseteq [\![g]\!]$ et $r' = r''_{[X' \leftarrow 0]}$,*
- *les états sont tous finals.*

La construction de l'automate des régions est illustrée sur la Fig. 4.

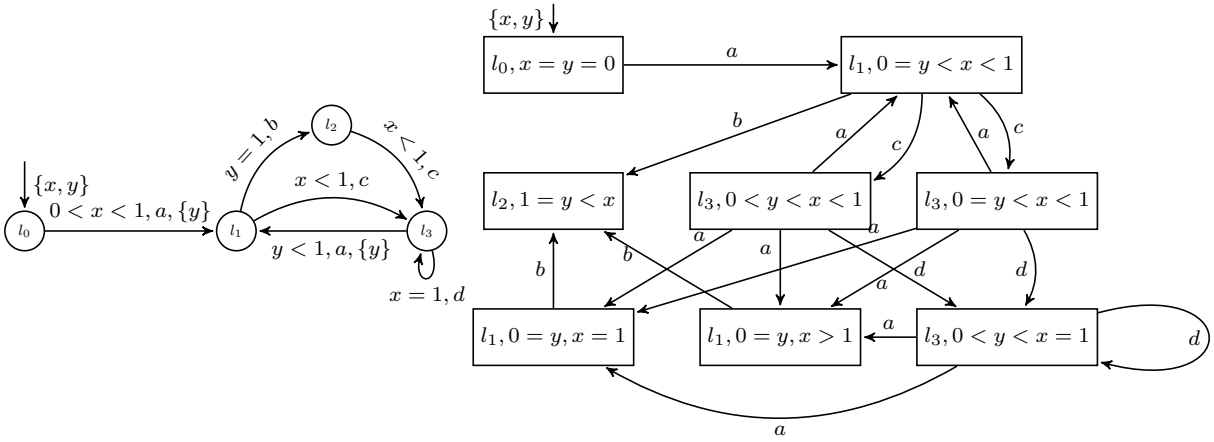


FIG. 4 – Exemple d'automate des régions associé à un TA.

La proposition suivante établit que le langage régulier reconnu par l'automate des régions d'un TA $\mathcal{A}$ est exactement l'ensemble des traces non temporisées reconnu par $\mathcal{A}$.

Proposition 2. *Soit $Untime(traces(\mathcal{A})) = \{\overline{\sigma} \mid (\overline{t}, \overline{\sigma}).t' \in traces(\mathcal{A})\} \subseteq \Sigma^*$ l'ensemble des traces non-temporisées de $\mathcal{A}$, alors $Untime(traces(\mathcal{A})) = \mathcal{L}(\alpha(\mathcal{A}))$.*

Quelle que soit la localité dans laquelle on se trouve, en s'abstrayant du temps, exactement les mêmes transitions sont tirables sur deux valuations d'une même région standard. Plus précisément, on dit que pour un TA $\mathcal{A}$ dont la constante maximale est M, $\equiv_M$ est une bisimulation de $\mathcal{A}$ par $\mathcal{A}$. Pour une définition formelle de bisimulation voir [AD94]. Ce résultat justifie la correction de la construction de l'automate des régions.

L'automate des régions permet de décider l'accessibilité d'une localité dans un TA, mais il faut savoir que son nombre d'états est exponentiel en le nombre d'horloges du TA.

2.1.3 Résultats importants

Proposition 3. *Pour les automates temporisés,*

1. *le problème de l'accessibilité d'une localité est PSPACE-complet,*
2. *le problème de l'universalité est indécidable,*
3. *le problème de l'inclusion de traces est indécidable,*
4. *les TA ne sont pas clos par passage au complémentaire.*

La Figure 5 est un exemple de TA pour lequel on ne pourra pas construire de TA reconnaissant le complémentaire de ses traces. En effet, pour reconnaître le complémentaire des traces de cet automate, il faudrait s'assurer qu'une unité de temps après chaque a, il n'y ait pas d'autre a. Or, il peut y avoir un nombre non borné de a en une unité de temps, ce qui signifie que l'on aurait besoin d'un nombre non borné d'horloges.

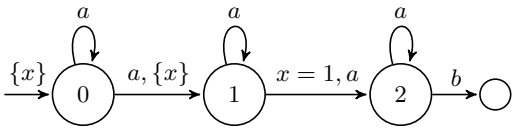


FIG. 5 – Exemple d'automate non complémentable.

2.2 Automates temporisés et déterminisation

Dans la génération de test, il est très intéressant de travailler sur des automates déterministes. En effet, on est amené à calculer, étant donnée une trace, l'ensemble des actions possibles, ce qui est très proche de la déterminisation. Nous allons voir que malheureusement, la classe des TA déterminisables est une sous-classe stricte des TA. Une fois que nous aurons établi ce résultat, nous présenterons une heuristique de déterminisation [BBBB09], puis nous exposerons des classes bien connues de TA déterminisables pour lesquelles ce pseudo-algorithme termine, et enfin nous étudierons une méthode de sur-approximation [KT09].

2.2.1 Être ou ne pas être déterminisable

Un automate déterministe est un automate qui, pour toute trace, a au plus une exécution.

Définition 7. *Un TA $\mathcal{A}$ est déterministe si $\forall l \in L$, $\forall a \in Act$, pour toute paire de transitions $(e_1 = l \xrightarrow{g_1,a,X'_1} l_1$ et $e_2 = l \xrightarrow{g_2,a,X'_2} l_2$ dans $\mathcal{A}$ telles que $e_1 \neq e_2)$, $[\![g_1]\!] \cap [\![g_2]\!] = \emptyset$.*

Proposition 4. *La classe des TA déterministes (DTA) est close par passage au complémentaire.*

On a vu dans la partie précédente un exemple (Figure 5) de TA non complémentable, ce qui signifie que l'on ne peut pas exprimer le langage reconnu par ce TA avec un DTA (sinon il existerait un DTA reconnaissant le complémentaire de ses traces). On obtient donc le corollaire suivant.

Corollaire 1. *Les TA déterministes sont strictement moins expressifs que les TA classiques.*

Proposition 5. *La classe des DTA est close par union et intersection.*

Proposition 6. *Le problème de l'inclusion de traces, c'est-à-dire étant donnés $\mathcal{A}_1$ un TA et $\mathcal{A}_2$) un DTA, est-ce que traces$(\mathcal{A}_1) \subseteq$ traces$(\mathcal{A}_2)$, est PSPACE-complet.*

La classe des automates déterminisables a des propriétés intéressantes, malheureusement on ne sait pas décider de l'appartenance d'un TA à cette classe.

Proposition 7. *Étant donné un TA, la question de l'existence d'un DTA acceptant les mêmes traces temporisées est un problème indécidable.*

2.2.2 Procédure de déterminisation [BBBB09]

On présente, dans cette partie, une construction dont le but est la déterminisation d'un TA. On donnera des conditions suffisantes pour que la construction soit applicable à un TA. Si cette procédure termine, on verra que le résultat sera un déterminisé du TA en argument. Cette procédure s'applique à un TA $\mathcal{A}$ en 4 étapes :

- un dépliage de $\mathcal{A}$ sous forme d'un arbre temporisé infini,
- une abstraction des régions,
- une déterminisation symbolique,
- une réduction du nombre d'horloges, permettant de replier l'arbre en TA.

Nous allons présenter l'algorithme sur un exemple (Figure 6). Pour une présentation formelle, voir [BBBB09].

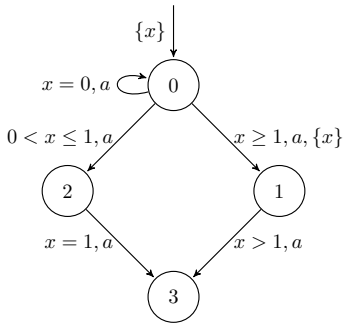


FIG. 6 – Automate sur lequel on explique la construction.

Dépliage : $\mathcal{A}^\infty$

On définit un arbre infini reconnaissant les mêmes traces que $\mathcal{A}$.

- Posons $Z = z_0, z_1, ...$ un ensemble infini d'horloges. L'idée est de déplier $\mathcal{A}$ en utilisant une nouvelle horloge à chaque niveau de l'arbre. Cela permet qu'à tout niveau i, chaque horloge de $\mathcal{A}$ soit encodable avec les horloges de $Z_i = z_0, ..., z_i$. On peut alors exprimer les contraintes sur les transitions en fonction des horloges de Z_i.
- Chaque noeud n de $\mathcal{A}^\infty$ est étiqueté par un couple $(l, \alpha) \in L \times Z^X$ où :
  - l est la localité correspondante de $\mathcal{A}$,
  - α décrit l'encodage des horloges de $\mathcal{A}$ par les horloges de $\mathcal{A}^\infty$.

$\mathcal{A}^\infty$ est déterministe en entrée, c'est-à-dire que pour toutes les exécutions possibles d'une même trace, les valuations d'horloges à l'issue de la lecture de cette trace seront les mêmes. C'est une propriété très importante pour la correction de cette construction. Cette construction préserve les traces de $\mathcal{A}$.

Pour l'exemple on obtient l'arbre de la Figure 7.

Abstraction des régions : $\mathcal{R}(\mathcal{A}^\infty)$

On étend naturellement la notion d'automate des régions à l'arbre infini définit à la première étape. Au niveau i, on considère les régions sur l'ensemble d'horloges Z_i, les autres n'étant pas pertinentes. Cela permet de toujours considérer les régions d'un ensemble fini d'horloges. On fait donc une transformation de chaque transition comme décrit dans la Figure 8. Une transition peut être transformée en plusieurs transitions si plusieurs régions vérifient les hypothèses décrites dans le schéma. Les traces reconnues sont toujours celles de $\mathcal{A}$.

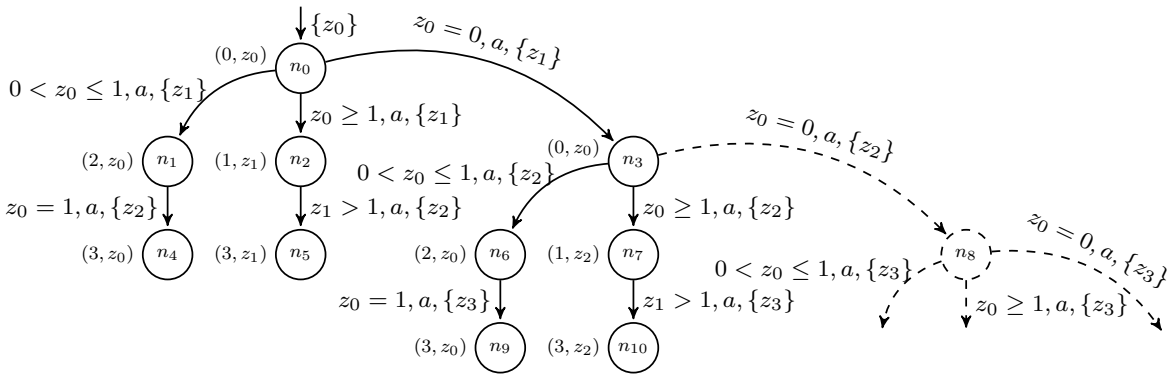
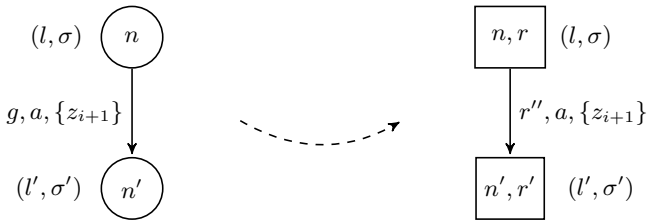


FIG. 7 – Première étape : dépliage de l'automate.



avec $r \in Reg^{Z_i}$, $r'' \in Reg^{Z_i}$ et $r' \in Reg^{Z_{i+1}}$ sont telles que r'' successeur temporel de r, r" $\models g$ $r'_{|Z_i} = r''$ et $r'_{|\{z_{i+1}} = \{0\}$.

FIG. 8 – Deuxième étape : schéma de transformation des transitions.

Déterminisation symbolique : $SymbDet(\mathcal{R}(\mathcal{A}^\infty))$
On déterminise l'arbre infini $\mathcal{R}(\mathcal{A}^\infty)$ sur l'alphabet symbolique composé des régions sur les horloges introduites jusqu'à présent et des actions. Pour notre exemple, on obtient l'arbre déterministe représenté Figure 9.

On obtient un arbre reconnaissant toujours les mêmes traces, mais celui-ci est déterministe. On souhaite maintenant se ramener à un DTA, en réduisant le nombre d'horloges et le nombre de localités.

Repliage : $\mathcal{B}_{\mathcal{A},\gamma}$
L'idée principale à cette étape est d'oublier les horloges inutiles à la volée. Les horloges inutiles sont simplement les horloges qui n'apparaissent pas dans le mapping de X dans Z.

Attention, savoir de quelles horloges on aura besoin dans la suite est un problème indécidable, ici, on ne peut se prononcer que sur les horloges déjà introduites. On en introduira encore potentiellement une infinité sur lesquelles on ne sait se prononcer.

Définition 8. *Une* horloge active à un noeud n étiqueté (l, α) *est une horloge qui apparaît dans* α*. Autrement dit c'est une horloge qui est a priori utile.*

À cette étape, on a besoin que l'arbre de l'étape précédente ait la propriété suivante pour un $\gamma \in \mathbb{N}$.

Définition 9. *Soit* $\gamma \in \mathbb{N}$*, on dit que* $SymbDet(\mathcal{R}(\mathcal{A}^\infty))$ *est* γ*-clock-bounded si dans chaque noeud, le nombre d'horloges actives est borné par* γ*.*

Sous cette hypothèse, la construction fournira un automate déterministe $\mathcal{B}_{\mathcal{A},\gamma}$ reconnaissant les mêmes traces que $\mathcal{A}$. Le principe est le suivant :

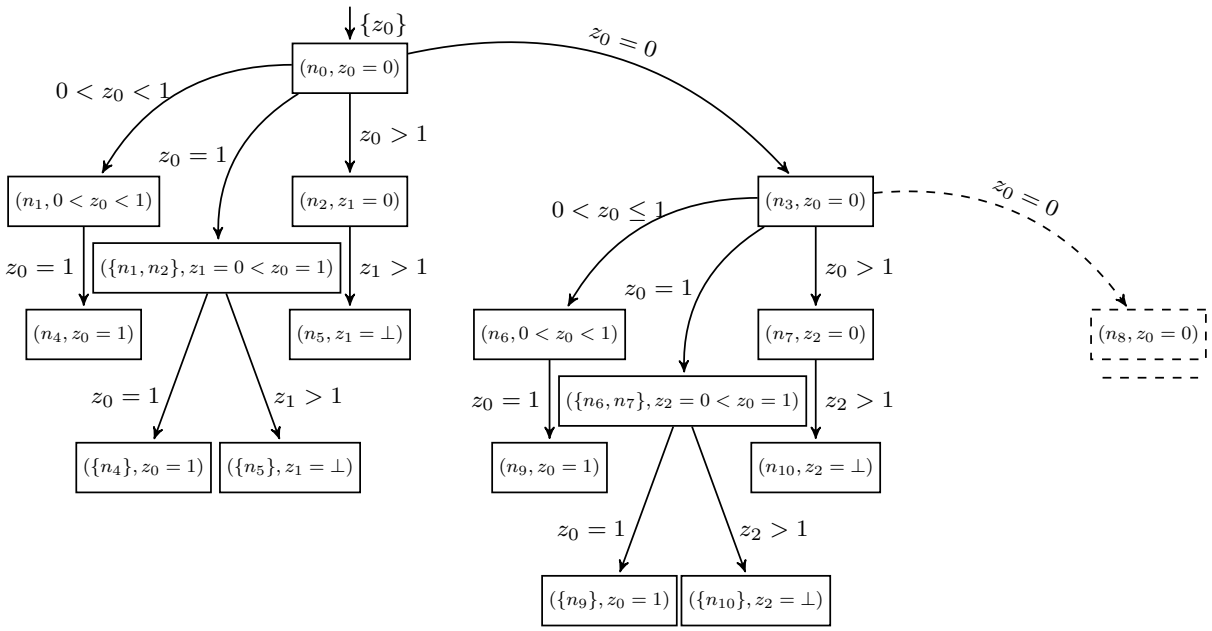


FIG. 9 – troisième étape : déterminisation symbolique sur un alphabet étendu.

- on fixe un ensemble d'horloges $X_\gamma = \{x_1, ..., x_\gamma\}$,
- on renomme toutes les horloges par niveaux croissants de façon déterministe, en utilisant pour chacune, une horloge inutile (par ex. l'horloge inutile de plus bas indice),
- on applique ce renommage aux contraintes et aux étiquettes des noeuds,
- il y a un nombre infini de noeuds, mais un nombre fini d'étiquettes possibles.
  On obtient $\mathcal{B}_{\mathcal{A},\gamma}$ en repliant (fusionnant) plusieurs noeuds de même étiquettes. Le résultat de l'application de cette construction à notre exemple est représenté Figure 10.

Proposition 8. *Si $SymbDet(\mathcal{R}(\mathcal{A}^\infty))$ est γ-clock-bounded, alors $\mathcal{B}_{\mathcal{A},\gamma}$ est un automate temporisé déterministe et $traces(\mathcal{B}_{\mathcal{A},\gamma}) = traces(\mathcal{A})$.*

Cette construction n'est pas effective, elle permet de comprendre le fonctionnement, mais on ne peut pas implémenter le dépliage en arbre infini. En revanche, on connaît la forme de l'automate que l'on souhaite calculer. On peut alors construire le déterminisé à la volée (quand il existe), on peut de plus automatiser ce calcul. Attention, pour un TA quelconque, on ne sait pas si les calculs termineront (c'est un problème indécidable). Les calculs sont faits pour un nombre fixé d'horloges, s'il n'est pas suffisant, le calcul bloque et on peut choisir d'essayer avec plus d'horloges : on ne sait jamais quand arrêter d'incrémenter le nombre d'horloges.

2.2.3 Classes d'automates déterminisables

On vient de voir une première condition suffisante mais indécidable pour que la procédure [BBBB09] termine, on introduit donc ici d'autres conditions un peu plus fortes, mais décidables qui impliquent la première.

Définition 10. *Soit $p \in \mathbb{N}$. Un TA $\mathcal{A}$* satisfait la *p*-hypothèse *si $\forall n \geq p$, $\forall r = (l_0, v_0) \xrightarrow{\theta_1, a_1} (l_1, v_1) ... \xrightarrow{\theta_n, a_n} (l_n, v_n)$ dans $\mathcal{A}$, $\forall x \in X$ alors soit $v_n(x) > M$ soit x est réinitialisé au cours de r.*

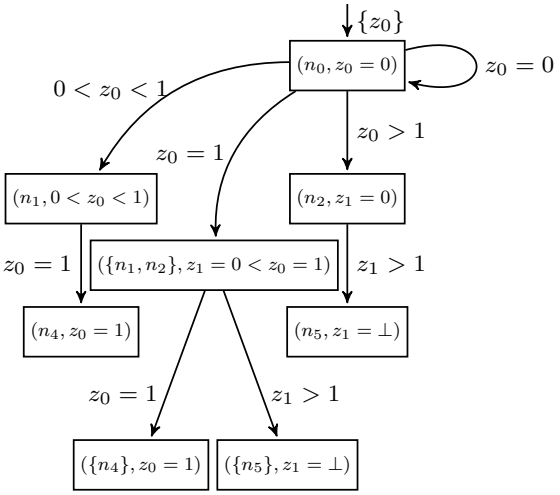


FIG. 10 – Résultat de la construction.

La p-hypothèse est une condition suffisante sur les TA pour pouvoir appliquer la procédure de déterminisation décrite précédemment. En effet, si $\mathcal{A}$ satisfait la p-hypothèse, alors le déterminisé $SymbDet(R(\mathcal{A}^\infty))$ est *p-clock-bounded*.

Définition 11. *Un TA $\mathcal{A}$ est* fortement non-Zeno *si* $\exists K \in \mathbb{N}$ *tel que* $\forall r = l_0 \xrightarrow{\theta_1, a_1} l_1 ... \xrightarrow{\theta_k, a_k} l_k$ *dans* $\mathcal{A}$, $(k \geq K \Rightarrow \Sigma^k{}_{i=1}\theta_i \geq 1)$.

On peut montrer que les TA fortement non-Zeno satisfont la p-hypothèse pour un $p \in \mathbb{N}$ exponentiel en la taille du TA. On introduit maintenant deux autres classes de TA qui ne satisfont pas la p-hypothèse, mais pour lesquelles la procédure termine puisqu'on peut borner le nombre d'horloges actives.

Définition 12. *Un TA est dit event-clock s'il possède seulement une horloge pour chaque action qui est réinitialisée à chaque occurrence de l'action.*

On voit aisément que dans ce cas, dans la construction, l'arbre sera $|\Sigma|$-*clock-bounded*.

Définition 13. *Un automate temporisé $\mathcal{A} = (L, l_0, \Sigma, X, M, E)$ est dit* à réinitialisations entières *si :* $\forall (l, g, a, X', l') \in E$, $((X' \neq \emptyset) \Leftrightarrow (\{(x = c) \in g \mid x \in X, c \in \mathbb{N}\} \neq \emptyset))$

Proposition 9. *Si un TA $\mathcal{A}$ est soit à réinitialisations entières, soit event-clock, soit fortement non-Zeno alors on peut construire un DTA $\mathcal{B}$ qui reconnaît les mêmes traces que $\mathcal{A}$. $\mathcal{B}$ est de taille doublement exponentielle en la taille de $\mathcal{A}$, simplement exponentielle dans le cas event-clock.*

Dans le cas des automates temporisés *event-clock*, la taille est simplement exponentielle en la taille de l'automate initial. Pour des informations supplémentaires sur ces classes d'automates, voir [AFH94] et [SPKM08].

2.2.4 Sur-approximation [KT09]

Krichen et Tripakis ont proposé un algorithme [KT09] qui, étant donné un TA $\mathcal{A}$ et un nombre d'horloges k, fournit un DTA à k horloges reconnaissant un langage contenant le langage du TA initial. Dans [KT09], la procédure est présentée avec une seule horloge, mais les auteurs

expliquent que la construction s'adapte à plusieurs horloges. Il est en fait possible de choisir un comportement plus fin pour les horloges d'observation. Ce comportement doit être donné sous la forme d'un automate fini nommé *squelette* dont l'alphabet est le produit cartésien de l'alphabet de $\mathcal{A}$ par l'ensemble des horloges d'observation. Une transition fait correspondre à une action, un ensemble d'horloges à réinitialiser.

Expliquons le principe de la construction quand on ne s'autorise qu'une seule horloge y que l'on réinitialise à chaque transition. Les localités de l'automate construit sont étiquetées par un ensemble de couples composés d'une localité dans le TA initial et d'une zone des valuations des horloges initiales possibles. Cette zone sur les horloges initiales est exprimée en fonction de y. L'approximation est faite sur les gardes des transitions qui sont exprimées avec la seule horloge de la sur-approximation. La construction se fait à partir de la localité $(l_0, x = y)$ où l_0 est la localité initiale du TA en argument, x une horloge de $\mathcal{A}$ et y l'horloge de l'approximation. Elle s'effectue ensuite par calculs successifs des meilleures sur-approximations des gardes des transitions tirables. La Figure 11 illustre cette construction sur un exemple.

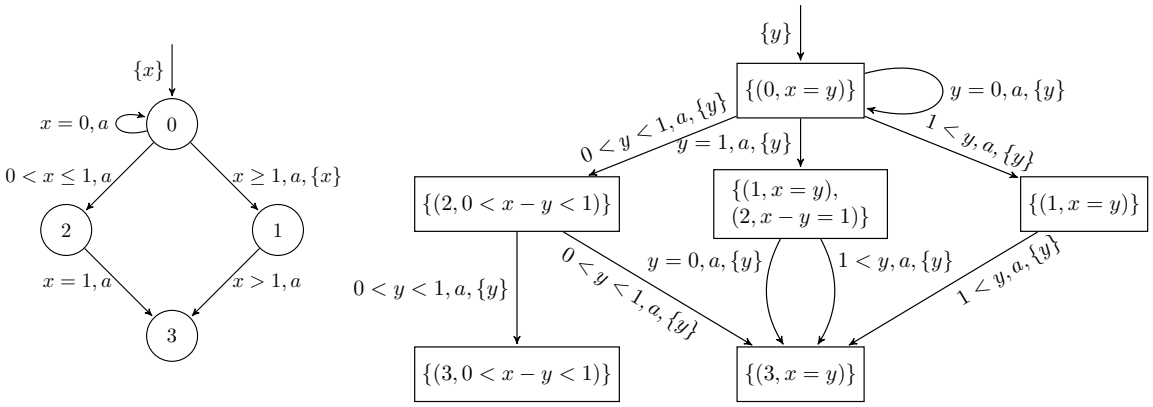


FIG. 11 – Exemple d'un automate et de sa sur-approximation déterministe à une horloge.

2.3 Conclusion de la Section 2

À présent que nous avons introduit les éléments de théorie des automates temporisés nécéssaires à la compréhension de notre travail, nous allons présenter en Section 3, le coeur du stage : notre approche par le jeu pour la déterminisation des TAs. Dans cette partie, nous avons vu deux approches différentes : une qui a privilégié l'exactitude à la terminaison et l'autre qui a décidé de faire une sur-approximation pour pouvoir terminer dans tous les cas. L'approche que nous présentons dans la Section 3 assure la terminaison dans tous les cas et l'exactitude dans plus de cas que ceux traités par la procédure de [BBBB09]. Comme Krichen et Tripakis, lorsqu'on ne peut être exact, on produit une sur-approximation déterministe. Notre méthode est à la fois plus générale que la procédure [BBBB09] et plus précise que la sur-approximation [KT09].

3 Une approche par le jeu pour déterminiser les automates temporisés

3.1 Introduction

Certains automates temporisés, bien que déterminisables, ne vérifient pas les hypothèses nécessaires pour la procédure de déterminisation [BBBB09]. Pour étendre la classe des automates

que l'on sait déterminiser, on veut prendre en compte les dépendances entre horloges de la forme $x = y + c$ avec $c \in \mathbb{N}$ et utiliser ces relations pour obtenir une sur-approximation dans le cas où les horloges données ne sont pas suffisantes. Notre méthode est basée sur les mêmes calculs que dans [KT09] mais elle consiste à chercher les meilleures réinitialisations possibles à chaque transition. Pour cela, on construit un jeu fini *turn-based* de sûreté $\mathcal{G}_{\mathcal{A},(1,M_y)}$ [GTW]. Le joueur qui veut rester dans des états sûrs est "Determinisator", il choisit les réinitialisations. L'autre joueur est "Spoiler" et il choisit la transition. On obtient l'implication suivante :

$$\left(\begin{array}{l}\text{Determinisator a une stratégie}\\ \text{gagnante pour } \mathcal{G}_{\mathcal{A},(1,M_y)}.\end{array}\right) \Rightarrow \left(\begin{array}{l}\text{Il existe un déterminisé de } \mathcal{A} \text{ à une}\\ \text{horloge avec constante maximale } M_y.\end{array}\right)$$

Cette méthode est inspirée de celle utilisée pour le problème de diagnosticabilité avec ressources fixées d'un TA [BCD05].

3.2 Un jeu pour la déterminisation d'un TA $\mathcal{A}$ avec la précision $(1, M_y)$

3.2.1 Description informelle

On se donne un TA quelconque $\mathcal{A}$. On souhaite construire un déterminisé de $\mathcal{A}$ si possible, une sur-approximation déterministe sinon. Pour cela, on se donne une horloge y, ce sera la seule horloge de l'automate que l'on va contruire. Avec cette horloge, on va estimer les valeurs des horloges de $\mathcal{A}$ pour construire les transitions. Notre approche se base sur la construction d'un jeu *turn-based* fini de sûreté dont les joueurs sont Spoiler et Determinisator. Determinisator choisit quand réinitialiser y et les choix de Spoiler sont des couples (a, r) où a est une action et r une région sur y. Une stratégie positionnelle de Determinisator correspond alors à un TA déterministe dont une transition est composée d'une tour de jeu de Spoiler puis de Déterminisator. Dans l'étude des stratégies gagnantes d'un jeu fini de sûreté, les stratégies positionnelles sont suffisantes. On ne considérera donc que ces stratégies pour le moment. On construit notre jeu de façon à ce que ce TA soit une sur-approximation de $\mathcal{A}$. Dans chaque état on retient l'ensemble des configurations dans lesquelles on peut résider, c'est-à-dire dans quelles localités de $\mathcal{A}$ et les estimations des horloges de $\mathcal{A}$ en fonction de y. De plus, on grise les états que Determinisator veut éviter de manière à ce que les stratégies gagnantes de Determinisator correspondent à un déterminisé exact de $\mathcal{A}$. Dans un état, on marque les configurations obtenues par une sur-approximation. Tant qu'il reste une configuration non marquée dans un état, l'état est considéré comme sûr, dans le cas contraire, il est grisé.
On a alors l'implication suivante :

$$\left(\begin{array}{l}\text{Determinisator a une stratégie}\\ \text{gagnante pour } \mathcal{G}_{\mathcal{A},(1,M_y)}.\end{array}\right) \Rightarrow \left(\begin{array}{l}\text{Il existe un déterminisé de } \mathcal{A} \text{ à une}\\ \text{horloge avec constante maximale } M_y.\end{array}\right)$$

On voit en Section 3.3 un exemple illustrant que la réciproque est fausse. On ne peut pas espérer avoir l'équivalence dans le cas général car la déterminisabilité d'un TA à ressources fixées est un problème indécidable [T06].

3.2.2 Construction de $\mathcal{G}_{\mathcal{A},(1,M_y)}$

Étant donné $\mathcal{A} = (L, l_0, \Sigma, X, M_X, E)$ un TA, on fixe $(1, M_y)$ une précision où 1 et M_y sont respectivement le nombre d'horloges et la constante maximale que l'on s'autorise pour la déterminisation. Dans cette partie, on définit notre jeu $\mathcal{G}_{\mathcal{A},(1,M_y)} = (V, \tilde{\Sigma}, \delta, v_0, \mathit{Gray})$ où $V = V_D \sqcup V_S$ est un ensemble d'états avec V_S et V_D qui sont respectivement l'ensemble des états de Spoiler et Determinisator, $\tilde{\Sigma} = (\Sigma \times Reg_{M_y}^{\{y\}}) \cup \{\emptyset, \{y\}\}$, $\delta \subseteq (V_S \times (\Sigma \times Reg_{M_y}^{\{y\}}) \times V_D) \cup$

$(V_D \times \{\emptyset, \{y\}\} \times V_S)$ un ensemble d'arêtes, $v_0 \in V_S$ l'état initial et *Gray* l'ensemble des états que Determinisator doit éviter.

Définition des états. Les états de $\mathcal{G}_{\mathcal{A},(1,M_y)}$ sont définis ainsi :

- $V = V_S \cup V_D$
- $V_S \subseteq \mathcal{P}(L \times \mathcal{P}(\mathbb{R}_+^{X\cup\{y\}}) \times \{\top, \bot\}) \times Reg_{M_y}^{\{y\}}$,
- $V_D \subseteq \mathcal{P}(L \times \mathcal{P}(\mathbb{R}_+^{X\cup\{y\}}) \times \{\top, \bot\}) \times Reg_{M_y}^{\{y\}}$,
- $v_{init} = (\{(l_0, \overrightarrow{\overline{0}_{X\cup\{y\}}} = \{(\overline{0}+t, t) \in \mathbb{R}_+^X \times \mathbb{R}_+\}, \top)\}, \{0\}) \in V_S$,
- $Gray = \{(\{(l_j, C_j, b_j)_{j\in J}\}, r'_y) \in V \mid \forall j \in J,\ b_j = \bot\}$

Un état du jeu est de la forme $(\{(l_j, C_j, b_j)_{j\in J}\}, r'_y)$ où J est fini, $l_j \in L$, C_j est un ensemble de valuations de $X \cup \{y\}$ exprimant les relations entre les horloges de X et y, b_j est un booléen représentant un marqueur et valant $\bot$ sur les configurations obtenues après une sur-approximation, et r'_y est une région sur $\{y\}$. Les états de *Gray* (ou grisés) sont ceux dont toutes les configurations sont marquées $\bot$.

Pour estimer les horloges de $\mathcal{A}$ grâce à l'horloge y du jeu, on définit la projection des ensembles de valuations de $X \cup \{y\}$ sur des ensembles de valuations de X. Étant donné R un ensemble de valuations sur $X \cup X'$, on note $R_{|X'} \subseteq \mathbb{R}_+^{X\cup X'}$ l'ensemble des valuations $v_{|X'}$ telles que $v \in R$. On peut ainsi considérer la garde $[r_y \cap C]_{|X}$ sur X induite par une région r_y sur $\{y\}$ et une conjonction C de relations entre y et les horloges de X. Informellement, on souhaite simuler le comportement d'un automate sur X avec l'horloge y pour seule horloge. r_y est une garde sur y qui, combinée avec les relations de C, produit une garde sur X. C'est ce que l'on nomme garde induite.

Mise à jour des relations entre y et les horloges de X. Pour simuler l'automate $\mathcal{A}$, on utilise une horloge y et les relations entre cette horloge et les horloges de X. Ces relations sont des conjonctions finies de formules atomiques de la forme $x - y \sim c$ où $x \in X$, $\sim \in \{=, <, >\}$ et $c \in [\![-M_y, M_X]\!]$. Lors de la construction des successeurs d'un état, il faut calculer les nouvelles relations entre y et X. On note U_X et U_y les fonctions de mise à jour de ces relations lorsque des horloges de X ou y sont réinitialisées. $U_X(r_y, C, X')$ calcule la relation obtenue en partant de la relation C et de la région r_y par une transition où $X' \subseteq X$ est l'ensemble des horloges de X réinitialisées dans $\mathcal{A}$. $U_y(r_y, C, p)$ calcule la relation obtenue en partant de la relation C et de la région r_y avec le choix de Determinisator $p \in \{\emptyset, \{y\}\}$. Les horloges ne prenant que des valeurs positives, la conjonction avec $\wedge_{x\in X} x \geq 0 \ \wedge\ y \geq 0$ est implicite.

- On définit d'abord les mises à jour des relations $\tilde{U}_X$ et $\tilde{U}_y$ sans forcément respecter les constantes M_X et M_y :

$$\begin{aligned}\tilde{U}_X(r_y, C, X') &= \{(v+t, v_y+t) \in \mathbb{R}_+^X \times \mathbb{R}_+ \mid v_y \in r_y,\ t \geq 0,\ \exists v' \in \mathbb{R}_+^X \text{ tel que } \\ &\quad v'_{[X'\leftarrow 0]} = v \text{ et } (v', v_y) \in C\} \\ &= \overrightarrow{(r_y \cap C)_{[X'\leftarrow 0]}}\end{aligned}$$

$$\begin{aligned}\tilde{U}_y(r_y, C, p) &= \{(v+t, v_y+t) \in \mathbb{R}_+^X \times \mathbb{R}_+ \mid t \geq 0,\ \exists v'_y \in r_y \text{ tel que } v'_{y[p\leftarrow 0]} = v_y \\ &\quad \text{et } (v, v'_y) \in C\} \\ &= \overrightarrow{(r_y \cap C)_{[p\leftarrow 0]}}\end{aligned}$$

- On définit alors $U_X(r_y, C, X')$ et $U_y(r_y, C, p)$ comme les conjonctions les plus petites (pour l'ensemble des valuations les vérifiant puis pour le nombre de formules atomiques les composant) vérifiées par tous les éléments de $\tilde{U}_X(r_y, C, X')$ et $\tilde{U}_y(r_y, C, p)$. On donne des définitions formelles en Annexe.

Successeurs élémentaires par un choix de Spoiler. On définit $Succ_e((l,C,b),(a,r_y))$ l'ensemble des successeurs élémentaires d'une configuration (l,C,b) par le choix (a,r_y) de Spoiler par :

$$Succ_e((l,C,b),(a,r_y)) = \left\{ (l',C',b') \;\middle|\; \begin{array}{l} \exists l \xrightarrow{g,a,X'}_{\mathcal{A}} l' \wedge [r_y \cap C]_{|X} \cap g \neq \emptyset, \\ C' = U_X(r_y,C,X'), \\ b' = ([r_Y \cap C]_{|X} \subseteq g) \wedge b \end{array} \right\}.$$

Moins formellement, Spoiler choisit un couple (a,r_y), autrement dit une action à tirer et une région dans laquelle la tirer. Pour chaque configuration de l'état source, on calcule les successeurs élémentaires par cette transition, on met à jour les relations entre y et les horloges de X en fonction de la réinitialisation d'horloges de X lors des transitions correspondantes. La négation de la condition $[r_y \cap C]_{|X} \subseteq g$ signifie que la transition que l'on construit est nécessaire parce qu'elle peut être tirable. La condition $[r_y \cap C]_{|X} \cap \neg g \neq \emptyset$ signifie que la construction de la transition provoque une sur-approximation. Enfin, une configuration est marquée $\top$ si son prédécesseur est marqué $\top$ et qu'aucune sur-approximation n'est faite sur le calcul de ce successeur.

Successeurs composés par un choix de Spoiler. On définit l'ensemble des successeurs élémentaires d'un état $(\{(l_j,C_j,b_j)_{j\in J}\},r'_y)$ de Spoiler par le choix (a,r_y) de Spoiler, noté $Succ((\{(l_j,C_j,b_j)_{j\in J}\},r'_y),(a,r_y))$:

$$Succ((\{(l_j,C_j,b_j)_{j\in J}\},r'_y),(a,r_y)) = \cup_{j\in J} Succ_e((l_j,C_j,b_j),(a,r_y)),$$

avec la convention $Succ((\{(l_j,C_j,b_j)_{j\in J}\},r'_y),(a,r_y)) = \emptyset$ si r_y n'est pas un successeur de r'_y.

Successeur d'un état de Spoiler. Soit $q_S = (\{(l_j,C_j,b_j)_{j\in J}\},r'_y) \in V_S$. Les successeurs de q_S sont des états de Determinisator et pour $(a,r_y) \in \Sigma \times Reg_{M_y}^{\{y\}}$, $q_S \xrightarrow{a,r_y} (\{(l_i,C_i,b_i)_{i\in I}\},r_y)$ si et seulement si $\{(l_i,C_i,b_i)_{i\in I}\} = Succ(q_S,(a,r_y))$. Moins formellement, les successeurs d'un état de Spoiler sont associés à un choix (a,r_y) et chacun est la réunion des successeurs élémentaires de toutes les configurations de l'état source couplée avec la région choisie r_y. Si r_y n'est pas un successeur temporel de r'_y alors le successeur dans le jeu par ce choix est $(\emptyset,r_y)$ et c'est un état puits qu'il n'est pas nécessaire de construire.

Successeur d'un état de Determinisator. Soit $q_D = (\{(l_i,C_i,b_i)_{i\in I}\},r_y) \in V_D$. Les successeurs de q_D sont des états $(\{(l_i,C'_i,b_i)_{i\in I}\},r'_y) \in V_S$ et pour $p \in \{\emptyset,\{y\}\}$, $q_D \xrightarrow{p} (\{(l_i,C'_i,b_i)_{i\in I}\},r'_y)$ si et seulement si $((\forall i \in I,\ C'_i = U_y(r_y,C_i,p)) \wedge r'_y = r_{y_{[p\leftarrow 0]}})$. Moins formellement, Determinisator peut réinitialiser y ou pas ; pour calculer les successeurs, on met simplement à jour les relations entre y et les horloges de X et la région sur $\{y\}$ dans laquelle on se trouve. Dans le cas où Determinisator choisit de ne pas réinitialiser y, on a $C'_i = C_i$ et $r'_y = r_y$.

En résumé, Spoiler choisit une action à tirer et une région dans laquelle la tirer, Determinisator décide alors de réinitialiser y ou pas. Le but de Determinisator est d'éviter les états dont toutes les configurations sont marquées $\bot$, ce sont les états pour lesquels on ne sait pas assurer qu'ils ne provoquent pas une sur-approximation stricte des comportements de $\mathcal{A}$. La Fig. 12 représente un tour de jeu où les états de Spoiler sont des rectangles et ceux de Determinisator sont des ellipses. Pour un choix (a,r_y) de Spoiler, Determinisator a deux choix, il réinitialise y ou pas.

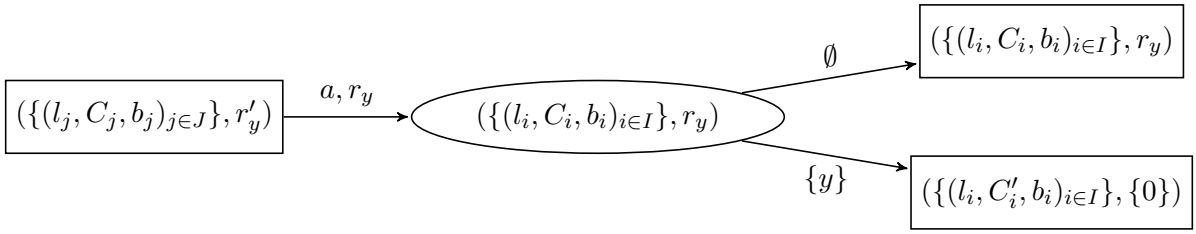


FIG. 12 – Forme d'un tour de $\mathcal{G}_{\mathcal{A},(1,M_y)}$.

3.2.3 Automate correspondant à une stratégie : propriétés

À chaque stratégie de Determinisator on associe le TA à une horloge et constante maximale M_y, obtenu en supprimant les états de Determinisator dans $\mathcal{G}_{\mathcal{A},(1,M_y)}$ par fusion de chaque choix de Spoiler avec le choix de Determinisator déterminé par la stratégie. Cet automate est déterministe par construction. Par exemple, la Fig. 13 représente cette fusion appliquée au tour représenté en Fig. 12 si la stratégie de Determinisator retient le choix $\{y\}$.

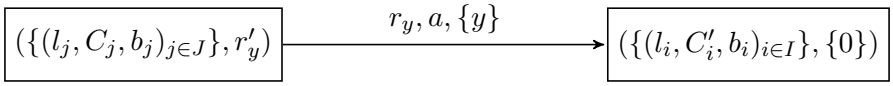


FIG. 13 – Fusion des transitions de la Fig. 12.

Définition 14. *Soit S une stratégie de Determinisator pour $\mathcal{G}_{\mathcal{A},(1,M_y)}$. Le TA obtenu par fusion est alors $Aut(S) = (V_S, \Sigma, v_0, \{y\}, E_S, M_y)$ où $v \xrightarrow{r,a,p} v' \in E_S$ si $(v, v') \in V_S^2$ et il existe $v_D \in V_D$ tel que $v \xrightarrow{a,r}_{\mathcal{G}_{\mathcal{A},(1,M_y)}} v_D \xrightarrow{p}_{\mathcal{G}_{\mathcal{A},(1,M_y)}} v'$.*

Théorème 1. *Pour toute stratégie de Determinisator S dans $\mathcal{G}_{\mathcal{A},(1,M_y)}$:*

$$traces(\mathcal{A}) \subseteq traces(Aut(S)).$$

Autrement dit, tout TA correspondant à une stratégie de Determinisator pour le jeu $\mathcal{G}_{\mathcal{A},(1,M_y)}$ est une sur-approximation déterministe de $\mathcal{A}$. La preuve de ce théorème est en annexe. Voyons maintenant que la sur-approximation est exacte pour toutes les stratégies gagnantes de Determinisator. La preuve de ce résultat est également en annexe.

Théorème 2. *Pour toute stratégie gagnante de Determinisator S dans $\mathcal{G}_{\mathcal{A},(1,M_y)}$,*

$$traces(\mathcal{A}) = traces(Aut(S)).$$

3.2.4 Un exemple

Dans cette section, on présente un exemple simple d'automate temporisé non déterministe (Fig. 14), le jeu correspondant construit avec la précision $(1, 1)$ (Fig. 15) où une stratégie gagnante de Determinisator pour ce jeu est tracée en rouge et enfin le déterminisé associé (Fig. 16). Dans la représentation du jeu, on omet le nom des états de Determinisator qui sont les mêmes que leurs successeurs par le choix $\emptyset$. On ne précise pas non plus le nom des états sans successeurs. Les états de *Gray* sont grisés.

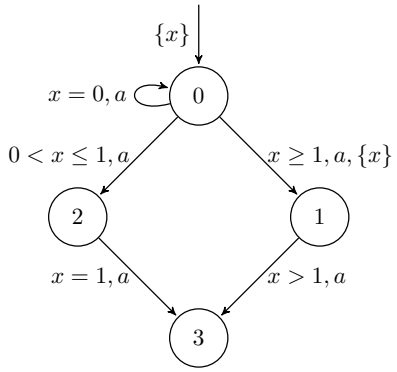


FIG. 14 – Exemple d'automate non déterministe.

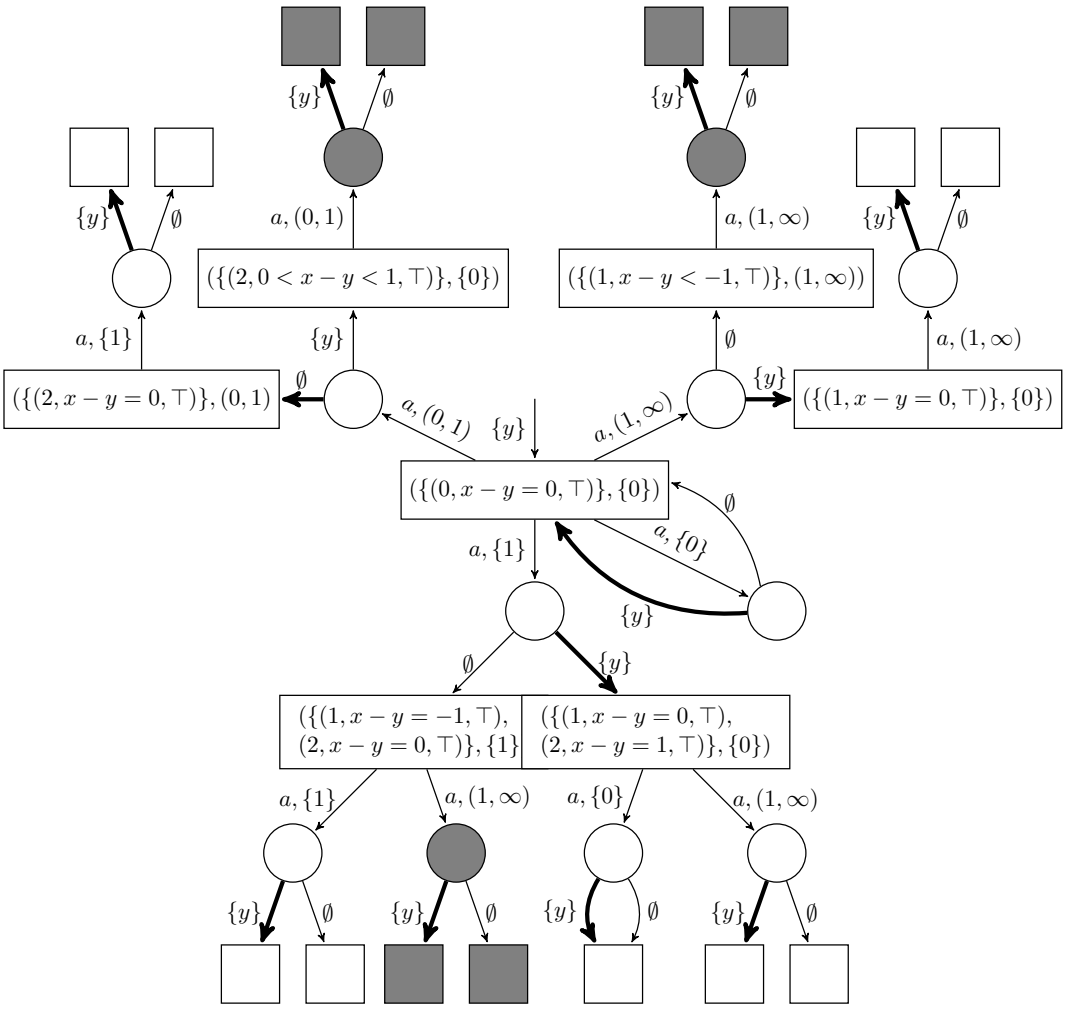


FIG. 15 – Exemple d'une stratégie gagnante de Determinisator.

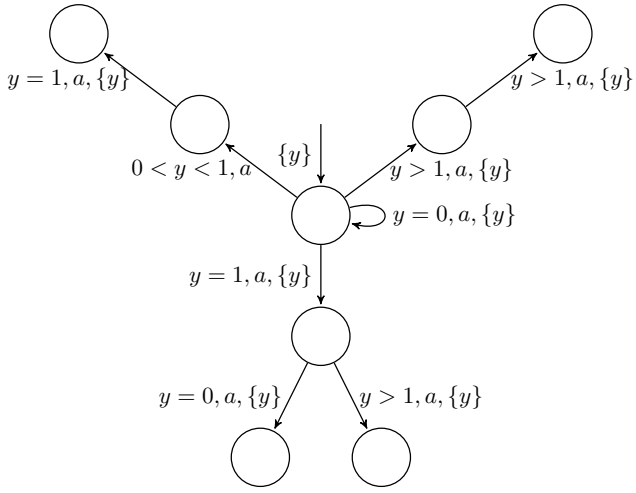


FIG. 16 – Déterminisé correspondant à la stratégie gagnante en Fig. 15.

3.3 Propriétés de notre approche

3.3.1 Attention aux inclusions de traces !

Dans notre méthode, nous faisons une sur-approximation des gardes que l'on n'arrive pas à exprimer avec l'horloge d'observation. Autrement dit, on ajoute des comportements à l'automate initial pour obtenir un TA déterministe. Lorsqu'on n'ajoute pas de traces, on obtient un déterminisé exact et c'est ce que l'on souhaite avoir le plus souvent possible. De plus, on veut savoir quand c'est le cas. Pour cela, on cherche à griser le moins d'états du jeu possible tout en gardant l'exactitude des états blancs. Tant qu'on exprime exactement toutes les traces, on ne grise pas, mais on peut faire des sur-approximations de gardes, sans qu'il faille griser. En effet, si les traces que l'on pense ajouter sur une transition sont couvertes par une autre transition, il n'y a pas lieu de griser l'état. Par exemple, le TA en Fig. 17 possède deux branches dont une est non déterminisable mais dont les traces sont incluses dans celles de l'autre branche qui est déterministe. Le TA est donc déterminisable. Grâce aux marqueurs attribués aux configurations de nos états, on prend bien en compte un tel cas et toutes les stratégies de Determinisator pour les jeux associés aux précisions $(1, M_y)$ avec $M_y \geq 0$ seront gagnantes.

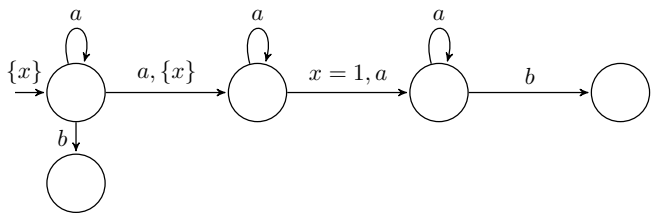


FIG. 17 – Exemple d'un TA avec une branche non déterminisable incluse dans une branche déterminisable.

3.3.2 La réciproque de l'implication est quand même fausse

Malgré ce que nous expliquons en Section 3.3.1, dans certains cas Determinisator n'a pas de stratégie gagnante pour le jeu de précision $(1, M_y)$ alors que l'automate initial est déterminisable

avec cette précision. Le TA en Fig. 18 est un exemple pour lequel il n'existe pas de statégie gagnante pour le jeu de précision $(1, 1)$. Pourtant, la stratégie en Fig. 19 correspond à un déterminisé exact. Cette imprécision vient du fait que les comportements sur-approximés sur chaque branche sont couverts par les comportements des autres branches. On fait une sur-approximation stricte sur chaque branche, mais globalement, toutes les traces que l'on ajoute sont bien des traces de $\mathcal{A}$. En effet, la garde $0 < y < 1$ sur y lorsqu'on a la relation $0 < x - y < 1$ induit la garde $0 < x < 2$ sur x, or $(0 < x < 2) = (0 < x < 1) \vee (x = 1) \vee (1 < x < 2)$.

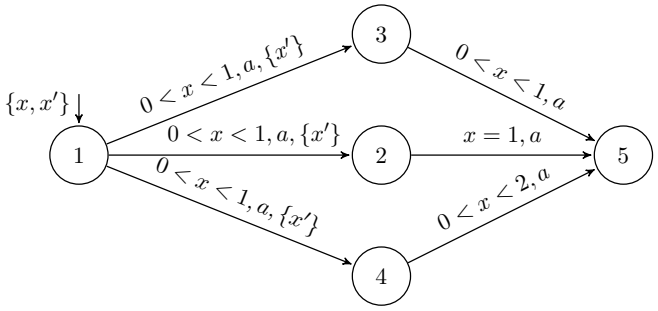


FIG. 18 – Exemple de TA tel que pour toute constante M_y, le jeu associé à la précision $(1, M_y)$ n'admet aucune stratégie gagnante pour Determinisator.

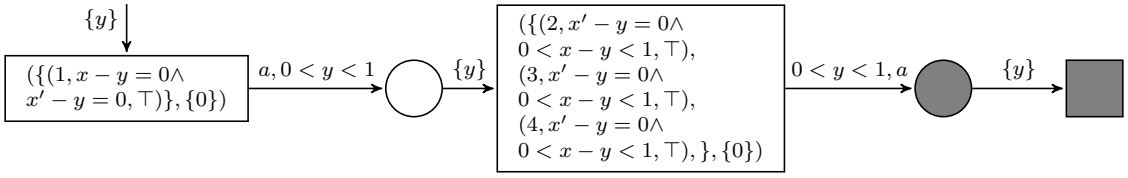


FIG. 19 – Exemple de stratégie perdante de Determinisator correspondant à un déterminisé exact.

3.3.3 Déterminisation des TAs à réinitialisations entières avec une seule horloge

Un TA à réinitialisations entières (voir la Définition 13, page 13) est un TA qui réinitialise des horloges uniquement sur les transitions où l'on teste une égalité d'horloge $(x = c)$ avec $c \in \mathbb{N}$. Ceci implique que les horloges ont toujours des parties fractionnaires égales. Pour un tel automate $\mathcal{A}$ de constante maximale M_X, la méthode [BBBB09] produit un déterminisé de taille doublement exponentielle en la taille de $\mathcal{A}$ avec M_X horloges. (Proposition 9). Notre méthode permet d'améliorer ce résultat en termes de ressources et en préservant la taille.

Proposition 10. *Si un TA $\mathcal{A}$ est à réinitialisations entières, alors on peut construire un DTA de taille doublement exponentielle en la taille de $\mathcal{A}$ à une seule horloge et même constante maximale qui reconnaît le même langage que $\mathcal{A}$.*

Démonstration. Soit $\mathcal{A} = (L, l_0, Act, X, M_X, E)$ un TA à réinitialisations entières. Considérons le DTA obtenu en utilisant le jeu associé à la précision $(1, M_X)$ et en choisissant la stratégie qui réinitialise y à chaque transition sur une garde contenant une contrainte de la forme $(x = c)$. Puisque $\mathcal{A}$ est à réinitialisations entières, aucune horloge de X n'est réinitialisée sur une autre

transition. Pour tout $x \in X$, on a alors en permanence $y \leq x$ et une relation de la forme $x-y=c$ avec $0 \leq c \leq M_X$, ou $x-y > M_X$ et dans ce cas $x > M_X$. On peut donc exprimer toutes les gardes nécessaires sur X avec des gardes sur y en respectant la constante maximale M_X. Toutes les localités de l'automate obtenu sont de la forme $(\{(l_i, C_i, \top)_{i\in I}\}, r_y)$ où I est fini, $l_i \in L$, pour tout $x \in X$, $C_i(x) = (x - y = c_x)$ avec $c_x \in [\![-M_X, M_X]\!]$ ou $C_i(x) > M_X$ et r_y une région sur $\{y\}$. La taille du TA obtenu est donc bornée par $2(M_X+1).2^{L.(M_X+2)^{|X|}}$. En effet, les localités du TA obtenu sont des états de Spoiler dans le jeu. En calculant le nombre maximal d'états de Spoiler pour ce jeu, on trouve cette borne. Dans chaque état, on a une région sur $\{y\}$, il y en a $2(M_X+1)$, et un ensemble de configurations composées d'une localité, d'une conjonction C_i telle que pour tout $x \in X$, $C_i(x) = (x - y = c_x)$ avec $c_x \in [\![-M_X, M_X]\!]$ ou $C_i(x) > M_X$ et d'un booléen constant, il y a donc $L.(M_X+2)^{|X|}$ configurations possibles et donc $2^{L.(M_X+2)^{|X|}}$ ensembles de configurations. □

3.4 Extension à plusieurs horloges : précision (k, M_Y)

On peut augmenter la précision de la méthode, c'est-à-dire les chances d'avoir un jeu avec une stratégie gagnante pour Determinisator, en utilisant un ensemble fini d'horloges $Y = \{y_1, ..., y_k\}$ pour la construction du jeu, au lieu de se limiter à une seule horloge y. La construction est similaire à celle avec une seule horloge. On considère des régions sur Y partout où on considérait des régions sur $\{y\}$. Les choix de Determinisator sont tous les $Y' \subseteq Y$. Précisons la définition de la mise à jour des relations $\tilde{U}$:

$$\begin{aligned}
\tilde{U}_X(r_Y, C, X') &= \{(v+t, v_Y+t) \in \mathbb{R}_+^X \times \mathbb{R}_+^Y \mid v_Y \in r_Y,\ t \geq 0,\ \exists v' \in \mathbb{R}_+^X,\ v'_{[X'\leftarrow 0]} = v \\
&\quad \text{et } (v', v_Y) \in C\} \\
&= \overrightarrow{(r_Y \cap C)}_{[X'\leftarrow 0]} \\
\tilde{U}_Y(r_Y, C, Y') &= \{(v+t, v_Y+t) \in \mathbb{R}_+^X \times \mathbb{R}_+^Y \mid t \geq 0,\ \exists v'_Y \in r_Y,\ v'_{Y[Y'\leftarrow 0]} = v_Y \text{ et } (v, v'_Y) \in C\} \\
&= \overrightarrow{(r_Y \cap C)}_{[Y'\leftarrow 0]}
\end{aligned}$$

Les tours de jeu sont alors étendus avec les nouveaux choix comme en Fig. 20.

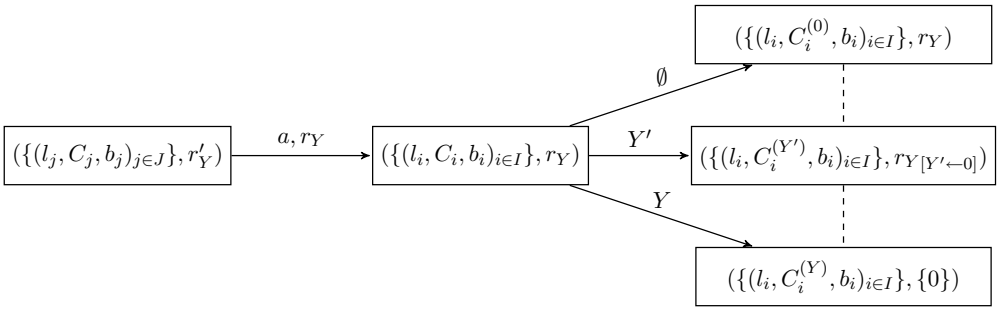


FIG. 20 – Exemple de transitions de $\mathcal{G}_{\mathcal{A},(k,M_Y)}$.

Pour tout le reste, la construction est semblable. De plus, les stratégies de Déterminisator correspondent encore à une sur-approximation de l'automate initial et, dans le cas où elle est gagnante, l'approximation est exacte. Les localités du TA obtenu par notre méthode étant des états de Spoiler, il y en a au plus $(2^{|Y|}.|Y|!.(2M_Y+2)^{|Y|}).(2^{|L|.2.(|X|.|Y|)^{3(M_X+M_Y+1)+(M_X+M_Y+1)^2}})$. La taille du résultat est donc doublement exponentielle en la taille de l'argument.

3.5 Comparaisons

On a présenté, en Section 2.2, deux approches existantes pour la déterminisation des automates temporisés proposées dans [BBBB09] et [KT09]. Les trois approches produisent un résultat de taille doublement exponentielle en la taille de l'argument (Section 3.4), mais ont des propriétés différentes. On voit dans cette section en quoi notre méthode améliore certains résultats de ces approches.

3.5.1 La méthode [KT09]

Krichen et Tripakis ont présenté une méthode de sur-approximation déterministe dans [KT09]. C'est-à-dire qu'ils construisent un TA déterministe reconnaissant un langage plus gros pour l'inclusion que le langage de l'automate initial. Notre procédure, étant données des ressources fixées (nombres d'horloges et constante maximale), retourne un déterminisé si elle peut, une sur-approximation sinon. Pour la méthode [KT09], les ressources sont également fixées, mais le comportement des horloges de la sur-approximation aussi. Il est donné sous la forme d'un automate fini appelé *squelette* qui détermine quand les horloges sont réinitialisées. Les auteurs construisent alors une sur-approximation de l'automate initial en respectant ce squelette. La sur-approximation se fait lors de l'expression des gardes de l'automate initial avec les horloges de la sur-approximation qui peuvent ne pas suffire.

Notre méthode est basée sur les mêmes calculs que celle-ci. Notre but était d'obtenir un déterminisé exact le plus souvent possible ou une sur-approximation la plus précise possible. On a tout d'abord étendu la notion de squelette aux squelettes temporisés (non décrit dans ce rapport) ce qui est parfois nécessaire pour obtenir un déterminisé exact. Ensuite, on s'est naturellement demandé comment choisir un bon squelette. C'est-à-dire comment choisir les horloges à réinitialiser lors des transitions pour préserver un maximum d'informations. On a formalisé ce choix par un jeu turn-based fini de sûreté à deux joueurs : Determinisator et Spoiler. Spoiler choisit quelle action faire et quand. Determinisator choisit quelles horloges réinitialiser. Determinisator doit éviter les états pour lesquels on risque d'avoir fait une sur-approximation. On ne sait pas en général décider quels états sont sûrs, en cas de doute, on marque comme *Gray* pour s'assurer que toute stratégie gagnante corresponde à un déterminisé exact. Cette incertitude s'explique par l'indécidabilité de l'inclusion des langages, et la conséquence est qu'on peut avoir non-existence d'une stratégie gagnante alors qu'une stratégie perdante correspond à un déterminisé exact. En se munissant d'une mémoire, on pourrait améliorer notre précision quant à la sûreté des états, mais la mémoire nécessaire à l'exactitude peut être non-bornée. En revanche, notre méthode contrairement à celle de Krichen et Tripakis, préserve les DTA dès que les ressources sont suffisantes.

Notre méthode préserve les TA déterministes. Décrire le comportement des horloges de l'observateur par un squelette sous forme d'automate fini, comme dans [KT09], n'est pas suffisant en général. On peut avoir besoin de différencier ce comportement en fonction de la région dans laquelle on se situe. On illustre en Fig. 21 l'utilité de temporiser les squelettes.

Notre approche permet de temporiser le choix des réinitialisations, ainsi, contrairement à la méthode présentée dans [KT09], notre approche par le jeu est exacte sur les DTA :

Théorème 3. *Si $\mathcal{A}$ est un DTA à k horloges et constante maximale M_X, alors le jeu $\mathcal{G}_{\mathcal{A},(k,M_X)}$ admet une stratégie gagnante pour Determinisator.*

Démonstration. $\mathcal{A}$ étant déterministe, pour construire une transition du jeu, on utilise une seule transition de $\mathcal{A}$. Il suffit à Determinisator de suivre le comportement des horloges de $\mathcal{A}$ pour obtenir une stratégie gagnante. □

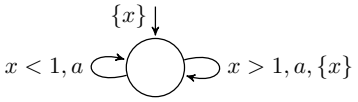


FIG. 21 – Exemple de TA n'admettant de squelette fini pour aucune ressource finie, mais déterminisable avec une seule horloge grâce à la théorie des jeux.

Si on restreint les ressources, on peut ne pas pouvoir déterminiser un DTA déterministe. Mais on montre, avec le même argument que dans la preuve du Théorème 3, qu'étant donné un DTA $\mathcal{A}$, le jeu correspondant à une précision (k, M_Y) admet une stratégie gagnante si et seulement si $\mathcal{A}$ est déterminisable avec ces ressources.

3.5.2 La méthode [BBBB09] : de nouveaux automates déterminisés

La méthode [BBBB09] s'appuie sur un mapping des horloges du TA initial dans les horloges du TA en construction. À chaque horloge du TA initial, on fait correspondre une horloge ayant exactement la valeur souhaitée. Dans le cas où cette méthode termine sur un TA $\mathcal{A}$, la taille du déterminisé obtenu est doublement exponentielle en la taille de $\mathcal{A}$. Dans la méthode utilisant un jeu, on s'autorise une sorte de mapping à constante près c'est-à-dire qu'on utilise des relations de la forme $x = y + c$ au lieu de $x = y$ et on conserve les inégalités lorsqu'on n'a pas de représentation exacte. Cela nous permet à la fois d'être plus souple pour la déterminisation et de fournir une sur-approximation déterministe lorsqu'on n'est pas capable de déterminiser.

Un exemple. L'automate $\mathcal{A}$ de la Fig. 22 ne termine pas dans la procédure [BBBB09] car cette procédure ne prend pas en compte les relations entre horloges. On illustre le problème en Fig. 23. On voit sur cette branche du dépliage déterminisé qu'il faudra un nombre non borné d'horloges. D'autre part, on donne en Fig. 24, une stratégie gagnante du jeu correspondant à $\mathcal{A}$ avec précision $(1, 2)$ et en Fig. 25 le déterminisé correspondant. Pour ces deux figures, on a fusionné les branches ne se différenciant que par la région. L'approche profite de la relation $x = y + 1$ pour exprimer exactement les gardes sur x avec y.

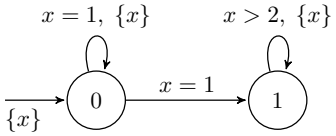


FIG. 22 – Exemple de TA ne passant pas dans la procédure [BBBB09] mais déterminisable avec une seule horloge grâce à la théorie des jeux.

Cet exemple illustre un type de comportements d'horloges que l'on traite très bien, contrairement à [BBBB09]. On voit dans la suite de cette section que cela nous permet d'étendre les classes d'automates temporisés que l'on sait déterminiser, et que notre approche traite les cas simples d'inclusion de traces, ce qui n'est pas du tout le cas de la procédure de déterminisation [BBBB09].

Affaiblissement de la propriété γ-*clock-bounded*. Si un automate temporisé $\mathcal{A}$ est tel que $SymbDet(\mathcal{R}(\mathcal{A}^\infty))$ (Définition 8) soit γ-*clock-bounded*, alors la procédure [BBBB09] termine. On

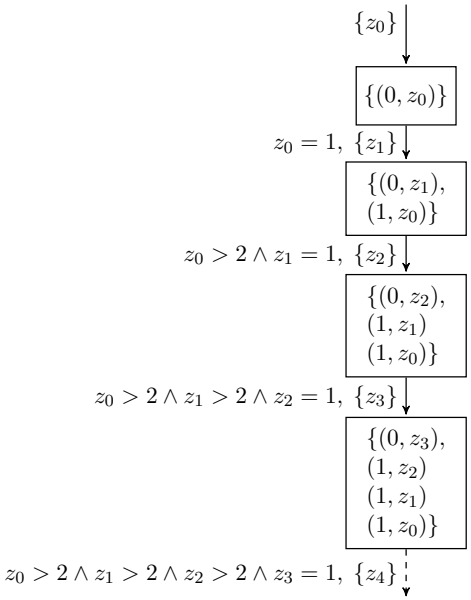


FIG. 23 – Branche déterminisée du dépliage du TA de la Fig. 22.

peut affaiblir cette propriété en considérant les parties fractionnaires en garantissant tout de même que notre jeu admette une stratégie gagnante et donc fournisse un déterminisé exact. Ce ne sera bien sûr pas une condition nécessaire.

Définition 15. *L'arbre $SymbDet(\mathcal{R}(\mathcal{A}^\infty))$ est γ-frac-clock-bounded si dans chaque état de $SymbDet(\mathcal{R}(\mathcal{A}^\infty))$, le cardinal de l'ensemble des parties fractionnaires des horloges actives est borné par γ.*

Proposition 11. *Soit $\mathcal{A}$ un TA avec k horloges et constante maximale M. Si $SymbDet(\mathcal{R}(\mathcal{A}^\infty))$ est γ-frac-clock-bounded alors le jeu $\mathcal{G}_{\mathcal{A},(\gamma,M)}$ admet une stratégie gagnante pour Determinisator.*

Démonstration. Décrivons une stratégie gagnante pour Determinisator dans $\mathcal{G}_{\mathcal{A},(\gamma,M)}$. On note $Y = \{y_1, ..., y_\gamma\}$ l'ensemble des horloges dont dispose Determinisator. On veut avoir en permanence une horloge y_i par partie fractionnaire d'horloge active telle que y_i soit inférieure ou égale à toute horloge $x \in X$ ayant même partie fractionnaire. Dans ce cas, on saura exprimer chaque garde de façon exacte. Montrons par récurrence que l'on peut se maintenir dans une configuration de ce type.

À l'état initial, toutes les horloges de X et de Y ont la même partie fractionnaire. La récurrence est donc bien initialisée.

Supposons que nous soyons dans un état conforme à ce que l'on souhaite et montrons que pour tout choix de Spoiler, Determinisator est capable de faire un choix qui préservera cette conformité. Un choix de Spoiler détermine une garde sur Y. Deux cas peuvent se produire :

- Si cette garde contient une contrainte de la forme $y_i = c$ alors Determinisator réinitialise y_i. En effet, ce choix permet de préserver les égalités de parties fractionnaires existantes excepté pour les horloges x de X qui sont réinitialisées lors de cette transition mais pour lesquelles on a à présent $x - y_i = 0$. Les inégalités sont également préservées.

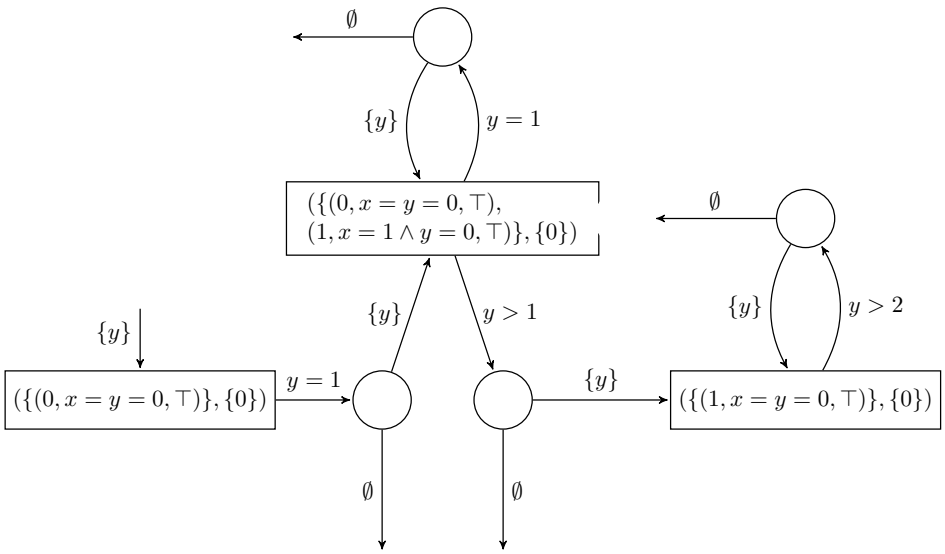


FIG. 24 – Stratégie gagnante dans le jeu correspondant au TA de la Fig. 22 avec une horloge.

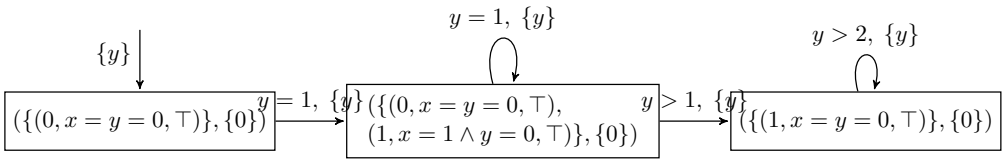


FIG. 25 – Déterminisé du TA de la Fig. 22 correspondant à la stratégie gagnante de la Fig. 24.

– Sinon, par hypothèse sur $\mathcal{A}$, il existe nécessairement une horloge de Y disponible ; c'est-à-dire qu'une fois qu'on a réinitialisé les horloges de X associées aux transitions considérées, il existe une horloge de Y qui n'est pas nécessaire à l'expression des horloges de X (si 2 horloges de Y ont même partie fractionnaire, on dit que la plus grande en valeur est disponible ; si elles sont égales, on prend la plus petite pour un ordre arbitrairement fixé sur Y au départ). On prend la plus petite des horloges disponibles et Determinisator la réinitialise. On préserve à nouveau la conformité.

On a donc montré par récurrence qu'on avait une stratégie gagnante pour Determinisator, *a priori* non-positionnelle. $\mathcal{G}_{\mathcal{A},(k,M_X)}$ étant un jeu *turn-based* fini de sûreté, il admet une stratégie gagnante positionnelle pour Determinisator [GTW]. □

Corollaire 2. *Pour tout TA à constante maximale M pour lequel la procédure [BBBB09] produit un DTA à k horloges, Determinisator a une stratégie gagnante dans $\mathcal{G}_{\mathcal{A},(k,M)}$.*

Cette propriété n'étant pas décidable sur un TA quelconque, on cherche des conditions plus fortes, qui soient décidables.

Affaiblissement de la p-hypothèse. Un automate temporisé vérifie la p-hypothèse (Définition 10, page 12) si lors de toute exécution de longueur supérieure à p, chaque horloge est soit réinitialisée au cours de l'exécution, soit plus grande que la constante maximale à la fin. Cette

propriété est décidable et assure que la procédure [BBBB09] termine. En effet si $\mathcal{A}$ vérifie la p-hypothèse alors $SymbDet(\mathcal{R}(\mathcal{A}^\infty))$ est *p-clock-bounded* [BBBB09].

On affaiblit cette condition comme suit :

Définition 16. *Soient $\mathcal{A}$ un TA et $p \in \mathbb{N}$. $\mathcal{A}$ vérifie la* p-frac-hypothèse *s'il existe une partition $X = H \cup H'$ telle que pour tout $n \geq p$, pour toute exécution $\rho = (l_0, v_0 = \overline{0}) \xrightarrow{a_0,t_0} (l_1, v_1) \xrightarrow{a_1,t_1} \ldots \xrightarrow{a_{n-1},t_{n-1}} (l_n, v_n)$ de $\mathcal{A}$, pour toute horloge $x \in H$, soit x est réinitialisée au cours de l'exécution, soit $v_n(x) > M_Y$ et pour tout $m \leq n$ et toute horloge $x' \in H'$, il existe $h \in H$ tel que $\lfloor v_m(x') \rfloor = \lfloor v_m(h) \rfloor$.*

Cette définition fournit une propriété affaiblie mais décidable sur les automates qui permet d'assurer que notre approche s'applique. En effet de même que si $\mathcal{A}$ vérifie la p-hypothèse alors $SymbDet(\mathcal{R}(\mathcal{A}^\infty))$ est *p-clock-bounded*, si $\mathcal{A}$ vérifie la p-frac-hypothèse alors $SymbDet(\mathcal{R}(\mathcal{A}^\infty))$ est *p-frac-clock-bounded*.

Extension d'une classe déterminisable par [BBBB09] Un TA est dit *event-clock* s'il y a exactement une horloge par action qui est réinitialisée lorsque l'action associée est tirée. La classe des TA *event-clock* est une classe de TA déterminisables car leurs dépliages sont $|\Sigma|$-*clock-bounded* ([BBBB09]). Cela vient du fait que les mises à jour dans l'automate initial ne dépendent que de la trace de l'exécution et non des localités. On peut donc étendre la classe des TA *event-clock* tout en conservant sa déterminisabilité.

Définition 17. *Un TA $\mathcal{A} = (L, l_0, L_{acc}, \Sigma, X, E)$ est dit* finiment vertébré *(resp.* temporellement vertébré*) s'il admet un DFA (resp. DTA) complet sur Σ décrivant les réinitialisations de ses horloges.*

Le dépliage d'un TA vertébré est bien $|X|$-*clock-bounded*. On affaiblit maintenant ces propriétés pour qu'elles impliquent que le dépliage soit $|X|$-$frac$-*clock-bounded*.

Définition 18. *Un TA $\mathcal{A} = (L, l_0, L_{acc}, \Sigma, X, E)$ est dit* frac-finiment vertébré *(resp.* frac-temporellement vertébré *s'il existe une partition $X = H \cup H'$ telle qu'il existe un DFA (resp. DTA) complet sur Σ décrivant les réinitialisations des horloges de H et si pour toute horloge $x' \in H'$ et tout état (l, v) de $\mathcal{A}$, il existe $h \in H$ tel que $\lfloor v(x') \rfloor = \lfloor v(h) \rfloor$.*

Les automates temporisés frac-vertébrés sont alors déterminisables par notre approche avec la précision $(|H|, M_X)$. Par construction de notre jeu, on peut encore affaiblir cette propriété en ne contraignant les horloges de H' que dans les localités où elles sont actives. En effet, notre approche est robuste aux comportements d'horloges inactives qui pouvaient empêcher la procédure [BBBB09] de terminer.

3.6 Choisir une bonne stratégie perdante

3.6.1 Une heuristique pour choisir une bonne sur-approximation

Dans certain cas, on ne peut pas obtenir de déterminisé exact car il n'y a pas nécessairement de stratégie gagnante pour Determinisator dans le jeu construit. Dans ce cas, on choisit une stratégie perdante et on a ainsi une sur-approximation déterministe du TA initial. Toutes les stratégies perdantes n'ont pas la même précision. On définit alors un critère de sélection basé sur la distance à *Gray*. Ainsi on favorise les sur-approximations exactes sur des préfixes les plus longs possible.

Définition 19. *Soit un graphe $\mathcal{G} = (V, v_0, E, Gray)$ avec $Gray \subseteq V$. Si $(v, v') \in E$, on note $v \to v'$. On définit la distance de v_0 à $Gray$ dans $\mathcal{G}$, notée $d(\mathcal{G}, Gray)$ par :*

$$d(\mathcal{G}, Gray) = min\{n \mid v_0 \to v_1 \to ... \to v_n,\ v_n \in Gray\}.$$

Cette définition induit un ordre partiel $\trianglelefteq$ sur les stratégies de Determinisator. Étant données deux stratégies de Determinisator S_1 et S_2, on a $S_1 \trianglelefteq S_2$ si pour toute stratégie S_S de Spoiler $d(\mathcal{G}_1, Gray) \geq d(\mathcal{G}_2, Gray)$ où $\mathcal{G}_1$ et $\mathcal{G}_2$ sont les graphes obtenus en considérant le sous-jeu correspondant aux stratégies S_S et respectivement S_1 et S_2. L'ensemble $Strat_D$ des stratégies positionnelles de Determinisator est fini, il admet donc un élément minimal pas nécessairement unique. Il paraît alors naturel de considérer qu'une bonne stratégie pour Determinisator doit être un élément minimal de $(Strat_D, \trianglelefteq)$. Nous proposons une procédure effective pour calculer ces stratégies.

Définition 20. *On définit par induction la suite suivante de sous-ensembles de V :*
$\begin{cases} Attr_0^S(Gray) & = & Gray, \\ Attr_{i+1}^S(Gray) & = & Attr_i^S(Gray) \cup (V_S \cap Pre(Attr_i^S(Gray))) \cup (V_D \cap \widetilde{Pre}(Attr_i^S(Gray))), \end{cases}$
où pour un ensemble de sommets V' d'un graphe, $Pre(V')$ est l'ensemble des sommets dont un prédécesseur est dans V' et $\widetilde{Pre}(V')$ est l'ensemble des sommets dont tous les successeurs sont dans V'. La fonction de rang $rg : V \to \mathbb{N}$ est définie pour tout $v_S \in V_S$ par $rg(v_S) = min(\{i \mid v_S \in Attr_i^S(Gray)\})$ avec la convention $min(\emptyset) = \infty$ et pour tout $v_D \in V_D$ par $rg(v_D) = 1 + max(\{rg(v) \mid v_D \in Pre(v)\})$.

Les stratégies de Determinisator pour le sous-jeu obtenu G_{min} en élaguant toutes les transitions d'un état v_D de Determinisator vers un état v_S de Spoiler tels que $rg(v_D) \neq 1 + rg(v_S)$ sont exactement les stratégies minimales de $(Strat_D, \leq)$. On peut affiner ce choix, par exemple en appliquant ce calcul à un sous-jeu de G_{min}.

3.6.2 Une simplification du jeu

Dans cette sous-section, on montre que les sur-approximations peuvent être tout aussi précises (pour notre odre partiel) si Determinisator ne peut réinitialiser qu'au plus une horloge à la fois.

Proposition 12. *Soit $\mathcal{A}$ un TA. On peut construire un TA $\mathcal{B}$ sur le même ensemble d'horloges ayant les mêmes traces que $\mathcal{A}$ tel qu'à chaque transition, au plus une horloge soit réinitialisée.*

Démonstration. Soit $\mathcal{A} = (L_\mathcal{A}, Y, \Sigma, l_{0\mathcal{A}}, E_\mathcal{A}, M)$. On définit $\mathcal{B} = (L_\mathcal{B}, Y, \Sigma, l_{0\mathcal{B}}, E_\mathcal{B}, M)$ de la façon suivante.

Les localités de $L_\mathcal{B}$ sont des éléments de $L_\mathcal{A} \times \mathcal{P}(Y) \times \mathcal{P}(Y^Y)$.

Pour supprimer les réinitialisations de sous-ensembles d'une transition, on ne réinitialise qu'une horloge et on garde l'information des réinitialisations des autres dans la localité cible. Plus précisément, on stocke dans chaque localité, l'ensemble des mappings des horloges de l'automate initial dans les horloges de l'automate construit. Garder tous les mappings possibles permet de réduire le nombre de localités. Pour effectivement avoir tous les mappings possibles, on garde également l'ensemble des horloges nulles de l'automate en construction à l'arrivée dans la localité, c'est-à-dire l'ensemble des horloges sur lesquelles on peut mapper les horloges qui ont été réinitialisées dans l'automate initial.

On munit Y^Y d'un ordre total.

L'état initial de $\mathcal{B}$ est $l_{0\mathcal{B}} = (l_{0\mathcal{A}}, Y, Y^Y)$. En effet, toutes les horloges de $\mathcal{B}$ sont nulles ainsi que celle de l'automate initial, donc on peut accepter tous les mappings.

Pour tout $(l, Y', p) \in L_{\mathcal{B}}$ déjà construit, pour tout $l \xrightarrow{g,a,X'}_{\mathcal{A}} l'$, on construit au plus deux transitions de $\mathcal{B}$. D'une part la transition tirable immédiatement (les horloges nulles dans la localité source sont alors nulles) et le complémentaire d'autre part. Ce découpage permet de garder tous les mappings possibles pour la localité cible dans $\mathcal{B}$. Formellement, on construit $(l, Y', p) \xrightarrow{g_1,a,\emptyset}_{\mathcal{B}} (l', Y', p'_1)$ et $(l, Y', p) \xrightarrow{g_2,a,X''}_{\mathcal{B}} (l', Y'', p'_2)$ où en posant $\sigma = min(p)$:

- $g_1 = (g \cap (\sigma^{-1}(Y') = 0)) \circ \sigma$,
- $g_2 = (g \setminus (\sigma^{-1}(Y') = 0)) \circ \sigma$,
- $X'' = \begin{cases} \emptyset & \text{si } X' = \emptyset \\ \{y\} & \text{où } y \text{ est la plus petite horloge disponible} \end{cases}$
  où une horloge y est *disponible* si $\sigma^{-1}(y) \subseteq X'$,
- $Y'' = \{y\}$,
- $p'_1 = up_1(p)$ où $\forall \alpha \in Y^Y,\ up_1(\alpha) = \left\{ \theta \in Y^Y \mid \begin{array}{l} \theta(x) = \alpha(x) \text{ si } x \notin X' \cup \alpha^{-1}(Y') \\ \theta(x) \in Y' \text{ sinon} \end{array} \right\}$,
- $p'_2 = up_2(p)$ où $\forall \alpha \in Y^Y,\ up_2(\alpha) = \left\{ \theta \in Y^Y \mid \begin{array}{l} \theta(x) = \alpha(x) \text{ si } x \notin X' \\ \theta(x) \in Y'' \text{ sinon} \end{array} \right\}$.

Dans le cas où l'une des deux gardes serait vide, il n'est pas nécessaire de construire la transition correspondante. On vérifie aisément que $traces(\mathcal{B}) = traces(\mathcal{A})$. □

Théorème 4. *Soit un TA $\mathcal{A}$. Soit G_1 (resp. G_2) un jeu construit à partir de $\mathcal{A}$ avec précision (k, M_Y) en autorisant Determinisator à réinitialiser un sous-ensemble fini de l'ensemble des horloges (resp. au plus une horloge). On alors l'équivalence suivante : Determinisator a une stratégie gagnante pour G_1 si et seulement Determinisator a une stratégie gagnante pour G_2.*

Démonstration. L'implication "⇐" est triviale car toute stratégie de Determinisator dans G_2 est une stratégie pour G_1. Soit S_1 une stratégie gagnante de Determinisator pour G_1. En appliquant la Proposition 12 à $Aut(S_1)$, on obtient un TA définissant une stratégie gagnante *a priori* non-positionnelle pour Determinisator dans G_2. G_2 étant un jeu fini de sûreté *turn-based*, on a l'existence d'une stratégie gagnante positionnelle pour Determinisator dans ce jeu [GTW]. On a donc bien équivalence. □

Plus généralement, pour toute stratégie dans notre jeu, on aura toujours une stratégie dans le jeu restreint aux réinitialisations simples, au moins aussi bonne pour notre critère.

Théorème 5. *Soit un TA $\mathcal{A}$. Soit $\mathcal{G}_{\mathcal{A},(k,M_Y)}$ le jeu construit à partir de $\mathcal{A}$ avec précision (k, M_Y) en autorisant Determinisator à réinitialiser un sous-ensemble fini de l'ensemble des horloges. Pour chaque stratégie positionnelle S de Determinisator, on a alors une meilleure stratégie positionnelle (au sens de notre ordre partiel) qui n'utilise que des réinitialisations simples.*

Autrement dit, il nous suffit de considérer le sous jeu autorisant Determinisator à réinitialiser au plus une horloge. La démonstration de ce théorème est en annexe.

3.7 Conclusion de la Section 3

Nous avons proposé une approche pour la déterminisation des automates temporisés aboutissant à une déterminisation exacte dans strictement plus de cas que [BBBB09] et fournissant une sur-approximation plus précise que celle de [KT09] sinon. Elle préserve le déterminisme dès que les ressources sont suffisantes et a permi d'étendre les classes d'automates que l'on sait déterminiser. Tout cela en préservant l'ordre de grandeur de l'automate temporisé obtenu. On voit dans la section suivante comment adapter notre approximation pour l'utiliser dans la problématique de la génération de test, tout en levant des hypothèses contraignantes de [KT09].

4 Test d'automates temporisés

4.1 Introduction

Intéressons nous maintenant à l'application de la déterminisation des TA dans le domaine du test de systèmes temps-réel, plus précisément pour la génération automatique de tests. Pour modéliser les comportements d'un système temps-réel, on utilise le modèle des automates temporisés avec un alphabet partitionné en deux ensembles : les entrées et les sorties. Étant données une spécification dont on connaît tout (pour nous sous forme de TAIO) et une implémentation dans une « boîte-noire » dont on ne pourra qu'observer les entrées et les sorties, le problème que l'on se pose est de savoir si l'implémentation est conforme à la spécification. On considèrera que l'implémentation à tester est modélisable (bien qu'on n'ait pas la modélisation) par un TAIO non-bloquant et complet en entrées (*input-enabled*) et que la spécification à tester est donnée sous forme d'un TAIO non-bloquant. Il est essentiel d'autoriser les modèles non-déterministes et partiellement observables pour la simplicité de la modélisation et l'expressivité, mais ceci implique alors une déterminisation lors de la génération de tests off-line. Nous voyons dans cette partie comment formaliser les concepts utilisés en test. Après avoir défini quelques notions utiles pour la formalisation, nous présentons la relation de conformité proposée dans [KT09]. Ensuite, nous développons deux parties. D'une part, nous proposons, une formalisation des cas de test, des propriétés qu'ils peuvent satisfaire et la construction d'un testeur qui permet de générer efficacement des cas de test corrects ou du test à la volée. D'autre part, nous présentons une formalisation des objectifs de test et de leur utilisation, puis une méthode pour construire un testeur qui permettra de générer le test efficacement en respectant l'objectif. Enfin, nous discuterons les solutions quant à la modélisation de l'urgence.

4.2 Modèle pour les spécifications

On modélise les spécifications par des automates temporisés entrées/sorties (TAIO) et on suppose que les implémentations sont modélisables par des TAIO. Par exemple, la spécification de la Fig. 26 impose que si l'on reçoit un $?a$ alors on émet un $!b$ entre 2 et 8 unités de temps après, si on reçoit une autre entrée le comportement n'est pas spécifié. Pour la formalisation de la

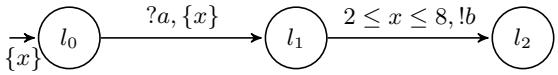


FIG. 26 – Exemple simple d'une spécification *Spec* donnée sous forme de TAIO.

notion de conformité, on fait deux hypothèses sur le modèle de l'implémentation : on suppose que l'implémentation est modélisable par un TAIO complet en entrées et non-bloquant. Autrement dit l'implémentation accepte toutes les entrées dans tous les états et le temps n'est pas bloqué, c'est-à-dire que de tout état q et pour tout délai t, sans recevoir d'entrée, l'implémentation a des exécutions de durée supérieure à t à partir de q.

Définition 21. *Un TAIO $\mathcal{A}$ est* complet en entrées *si* $\forall q \in Reach(\mathcal{A})$, $\forall ?a \in Act_?$, $q \xrightarrow{?a}$.

Définition 22. *Un TAIO $\mathcal{A}$ est* non-bloquant *si* $\forall q \in Reach(\mathcal{A})$, $\forall t \in \mathbb{R}_+$, $\exists r \in (Act_! \cup \mathbb{R}_+)^*$, $time(r) = t \wedge q \xrightarrow{r}$.

Actions non-observables. Pour modéliser les actions non-observables, on peut ajouter une action à l'alphabet du modèle des TAIO. Il faut alors redéfinir la notion de traces, car ce sont

les traces observables que l'on veut étudier. Il suffit pour cela de considérer la projection des traces obtenues en omettant les occurrences d'actions non-observables et en sommant les délais consécutifs. Une transition étiquetée par une action non-observable est traitée comme une ϵ-transition. On fait ici l'hypothèse qu'il n'y a pas de cycle d'actions non-observables. Sans cette hypothèse, on ne peut pas systématiquement se ramener au cas sans action inobservable, les TA avec ϵ-transitions étant strictement plus expressifs que les TA classiques. De plus, c'est une hypothèse raisonnable dans le domaine du test et on peut ainsi propager les gardes et les réinitialisations des transitions non-observables dans les celles des transitions suivantes comme dans [BDGP98], tout en préservant les traces observables. On se contente ici du modèle sans action non-observable, défini en Section 2, page 4.

Pour placer le test de conformité dans un cadre formel, on définit des notions qui servent en particulier à formaliser la notion de conformité en Section 4.3.

Étant donnés une trace σ et un état q de $\mathcal{A}$, on définit $\mathcal{A}$ *After* σ et *sorties*(q) qui sont respectivement l'ensemble des états accessibles par une exécution ayant σ pour trace et l'ensemble des observations possibles (actions de sorties et délais) dans l'état q.

Définition 23. *Soient $\sigma \in traces(\mathcal{A})$, $q \in Q$, alors :*
- $\mathcal{A}$ *After* $\sigma = \{q \in Q \mid q_0 \xrightarrow{\sigma} q\}$,
- *sorties*$(q) = \{!a \in Act_! \mid q \xrightarrow{!a}\} \cup \mathbb{R}_+$ *généralisé en* *sorties*$(Q') = \cup_{q \in Q'}$ *sorties*(q) *pour* $Q' \subseteq Q$,
- *entrées*$(q) = \{?a \in Act_? \mid q \xrightarrow{?a}\}$ *généralisé en* *entrées*$(Q') = \cup_{q \in Q'}$ *entrées*(q) *pour* $Q' \subseteq Q$.

Sur l'exemple *Spec* de la Figure 26 :
- *Spec After* $?a = \{(l_1, 0)\}$,
- *sorties*$((l_1, 1)) =$*sorties*(*Spec After* a.1) $= \mathbb{R}_+$ et *sorties*$((l_1, 4)) = \{!b\} \cup \mathbb{R}_+$,
- *entrées*$((l_0, 3)) = \{?a\}$ et *entrées*$((l_1, 1)) = \emptyset$.

4.3 Relation de conformité

On souhaite vérifier qu'un système temps-réel implémente correctement une spécification. Pour formaliser cette notion de conformité, on se place dans un cadre particulier où l'implémentation est modélisé par un TAIO non-bloquant et complet en entrées et la spécification par un TAIO non-bloquant. Dans le cadre du test, l'implémentation dont nous étudierons la conformité ne sera pas connue. Nous utiliserons alors le test pour essayer de vérifier la conformité de l'implémentation à la spécification. Une relation de conformité définit l'ensemble des modèles d'implémentations conformes à une spécification. On présente une seule relation de conformité, la relation **tioco**. Pour d'autres relations de conformité et des comparatifs, voir [ST08]. La relation **tioco** est une extension aux TAIO de la relation **ioco** définie pour des automates finis à entrées/sorties [Tre96].

Définition 24. *Soient $\mathcal{S}$ un TAIO non-bloquant et Imp un TAIO non-bloquant et complet en entrées, alors : Imp* ***tioco*** *$\mathcal{S} = \forall \sigma \in traces(\mathcal{S}) \cup \mathbb{R}_+$, sorties$(Imp$ After $\sigma) \subseteq$ sorties$(\mathcal{S}$ After $\sigma)$.*

Autrement dit, toutes les sorties observables de $\mathcal{I}$ après une trace temporisée de $\mathcal{S}$ doivent être incluses dans celles de $\mathcal{S}$ (spécifiées par $\mathcal{S}$) après la même trace.

Pour l'exemple de la Figure 27, on obtient $Imp1$ **tioco** *Spec* et $Imp2$ **tioco** *Spec*. En revanche, $Imp3$ n'est pas conforme à *Spec* pour **tioco** car *sorties*$(Imp3$ *After* $?a$ 1$) = \{!b\} \cup \mathbb{R}_+ \not\subseteq \mathbb{R}_+$ ($!b$

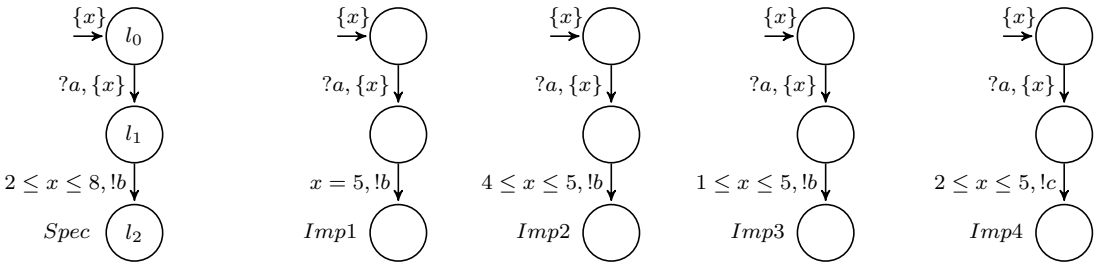


FIG. 27 – Exemples d'implémentations dont on commente la conformité à la spécification *Spec*.

peut être émis trop tôt) et $Imp4$ non plus car $sorties(Imp4\ After\ ?a\ 2) = \{!c\} \cup \mathbb{R}_+ \not\subseteq \{!b\} \cup \mathbb{R}_+$ (l'emission de $!c$ est non spécifiée).

La formalisation de la conformité d'une implémentation à une spécification est donnée par la relation **tioco**. Dans le cadre du test, le modèle de l'implémentation n'est pas connu et quand bien même, **tioco** est indécidable. Pour essayer de détecter des erreurs de conformité, on fait l'hypothèse que l'implémentation est modélisable par un TAIO non-bloquant et complet en entrées et on génère des tests pour filtrer le plus d'implémentations défectueuses possibles.

4.4 Cas de test

Pour tester la conformité d'une implémentation à une spécification, on utilise des cas de test. Dans cette section, on définit la notion de cas de test sous forme de TAIO, les propriétés qu'ils peuvent satisfaire et comment les utiliser.

Pour formaliser la notion de verdict, on définit un ensemble de traces finissant dans un sous-ensemble de localités et l'accessibilité d'un état.

Définition 25. *Soit* $\mathcal{A} = (L^{\mathcal{A}}, l_0^{\mathcal{A}}, Act^{\mathcal{A}}, X^{\mathcal{A}}, M^{\mathcal{A}}, E^{\mathcal{A}})$ *un TAIO, l'ensemble des* traces temporisées de $\mathcal{A}$ finissant en $L' \subseteq L^{\mathcal{A}}$*, noté* $traces_{L'}(\mathcal{A})$*, est défini par :* $traces_{L'}(\mathcal{A}) = \{r \mid r \in (\mathbb{R}_+.Act)^+.\mathbb{R}_+ \wedge\ (l_0, \overline{0}) \xrightarrow{r} (l, v)\ \wedge l \in L'\}$.

Définition 26. *Soit* $\mathcal{A} = (L^{\mathcal{A}}, l_0^{\mathcal{A}}, Act^{\mathcal{A}}, X^{\mathcal{A}}, M^{\mathcal{A}}, E^{\mathcal{A}})$ *un TAIO, l'ensemble des états* accessibles *de l'état* (l, v) *dans* $\mathcal{A}$*, noté* $Reach_{(l,v)}(\mathcal{A})$*, est défini par :* $Reach_{(l,v)}(\mathcal{A}) = \{(l', v') \in L^{\mathcal{A}} \times \mathbb{R}_+ \mid \exists r \in (\mathbb{R}_+.Act)^+.\mathbb{R}_+,\ (l, v) \xrightarrow{r} (l', v')\}$*. On dit qu'une localité* l *est* accessible *d'un état, s'il existe une valuation* v *telle que* (l, v) *soit accessible.*

Un cas de test est un TAIO associé à une spécification qui satisfait trois propriétés : son alphabet est le miroir de celui de la spécification, il accepte toutes les sorties de l'implémentation à tester et il a deux localités puits.

Définition 27. *Un* cas de test *pour une spécification* $S = (L^S, l_0^S, Act^S, X^S, M^S, E^S)$ *est un TAIO* $TC = (L^{TC}, l_0^{TC}, Act^{TC}, X^{TC}, M^{TC}, E^{TC})$ *muni de deux localités puits* ***Pass***, ***Fail*** *dans* L^{TC} *tel que :*

- $Act_!^{TC} = Act_?^S$ *et* $Act_?^{TC} = Act_!^S$,
- TC *est complet en entrées.*

Les localités puits correspondent aux verdicts émis par le cas de test. Un exemple de cas de test pour une spécification est représenté Fig. 28.

Un cas de test TC pour une spécification S définit un ensemble $traces_{\mathbf{Pass}}(TC)$ de traces acceptées, un ensemble $traces_{\mathbf{Fail}}(TC)$ de traces refusées et un ensemble $traces_{\mathbf{None}}(TC)$ de traces sans verdict :

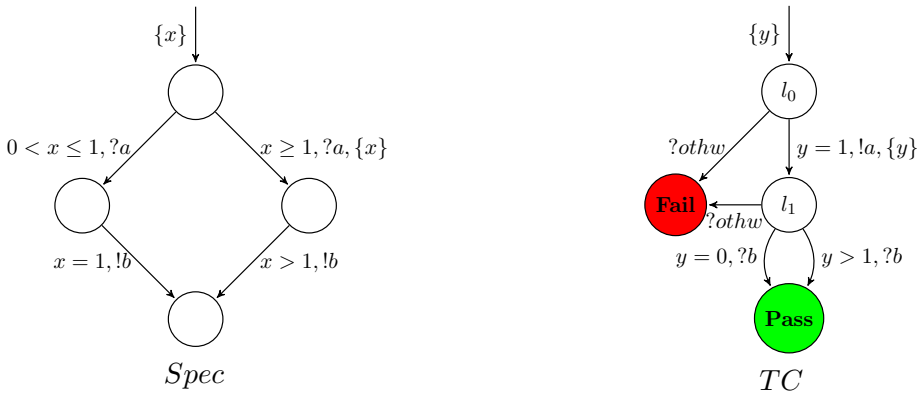


FIG. 28 – Exemple d'une spécification et d'un cas de test pour cette spécification.

- $traces_{\mathbf{Pass}}(TC) = traces_{\{\mathbf{Pass}\}}(TC)$,
- $traces_{\mathbf{Fail}}(TC) = traces_{\{\mathbf{Fail}\}}(TC)$,
- $traces_{\mathbf{None}}(TC) = traces(TC) \setminus (traces_{\{\mathbf{Pass}\}}(TC) \cup traces_{\{\mathbf{Fail}\}}(TC))$.

L'exécution d'un test sur une implémentation est modélisée par leur composition paralèlle.

Définition 28 (Composition paralèlle)**.** *Deux TAIO* $\mathcal{A}^i = (L^{\mathcal{A}^i}, l_0^{\mathcal{A}^i}, Act^{\mathcal{A}^i}, X^{\mathcal{A}^i}, M^{\mathcal{A}^i}, E^{\mathcal{A}^i})$, $i = 1, 2$ *avec des alphabets en miroirs sont* compatibles *pour la composition si* $X^{\mathcal{A}^1} \cap X^{\mathcal{A}^2} = \emptyset$ *(leur ensemble d'horloges sont disjoints). Dans ce cas,* $\mathcal{A}^1 \| \mathcal{A}^2 = (L^{\mathcal{A}}, l_0^{\mathcal{A}}, Act^{\mathcal{A}}, X^{\mathcal{A}}, M^{\mathcal{A}}, E^{\mathcal{A}})$ *est un OTAIO défini par :*

- $L^{\mathcal{A}} = L^{\mathcal{A}^1} \times L^{\mathcal{A}^2}$,
- $l_0^{\mathcal{A}} = (l_0^{\mathcal{A}^1}, l_0^{\mathcal{A}^2})$,
- $Act^{\mathcal{A}} = Act^{\mathcal{A}^1}$,
- $X^{\mathcal{A}} = X^{\mathcal{A}^1} \cup X^{\mathcal{A}^2}$,
- $M^{\mathcal{A}} = max(M^{\mathcal{A}^1}, M^{\mathcal{A}^2})$,
- $((l^1, l^2), g, a, X', (l'^1, l'^2)) \in E^{\mathcal{A}}$ *s'il existe* $(l^1, g^1, a, X'^1, l'^1) \in E^{\mathcal{A}^1}$ *et* $(l^2, g^2, \overline{a}, X'^2, l'^2) \in E^{\mathcal{A}^2}$ *telles que :*
  - $\overline{a}$ *est l'action miroir de* a,
  - $g = g^1 \wedge g^2$,
  - $X' = X'^1 \cup X'^2$.

La Fig. 29 illustre l'exécution du cas de test de la Fig. 28 sur une implémentation. On définit la possibilité de rejet d'une implémentation par un cas de test par Imp *fails* TC si $L^{Imp} \times \{\mathbf{Fail}\} \cap Reach_{(l_0^{TC}, \overline{0})}(Imp \| TC) \neq \emptyset$. Autrement dit, si une localité contenant **Fail** est accessible dans la composition en paralèlle de l'implémentation et du cas de test. Dans l'exemple de la Fig. 29, la localité $(l'_3, \mathbf{Fail})$ est accessible depuis l'état initial, donc Imp *fails* TC. Dans le domaine du test, on ne peut pas effectivement construire le produit car on ne dispose pas de la modélisation de l'implémentation. On cherche donc des exécutions éventuelles qui prouveraient que Imp *fails* TC.

Un cas de test pour une spécification n'émet pas nécessairement le verdict attendu, on définit donc les notions de correction et d'exactitude d'un cas de test. Informellement, un cas de test est correct s'il ne produit pas de faux négatifs et il est exact s'il ne produit pas de faux positifs, c'est-à-dire s'il émet **Fail** dès qu'il observe une erreur de conformité.

Définition 29. *Un cas de test* TC *est* correct *pour la spécification* S *si une implémentation n'est rejetée par le test que si elle n'est pas conforme à* S*, c'est-à-dire si pour toute implémentation* Imp, $ImpfailsTC \Rightarrow \neg(Imp \mathbf{tioco} S)$.

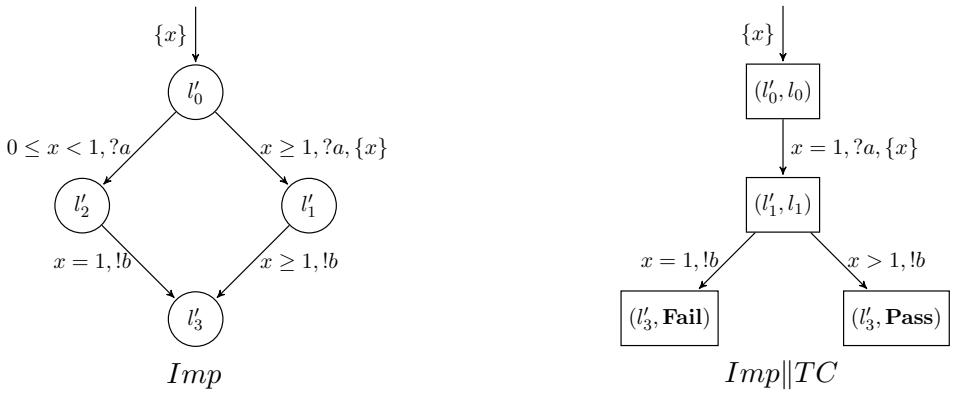


FIG. 29 – Exemple d'une implémentation et de son intéraction avec le cas de test TC de la Fig. 28.

Autrement dit, si **Fail** n'est émis que si TC observe une sortie non-spécifiée après une trace de S, c'est-à-dire $traces_{\mathbf{Fail}}(TC) \subseteq (traces(S).Act_?^{TC}.\mathbb{R}_+ \cap \overline{traces(S)})$.

Définition 30. *Un cas de test TC est* exact *pour la spécification S si **Fail** est émis dès que TC observe une sortie non-spécifiée après une trace de S, c'est-à-dire* $((traces_{\mathbf{None}}(TC).Act_?^{TC}.\mathbb{R}_+) \cap \overline{traces(S)}) \subseteq (traces_{\mathbf{Fail}}(TC))$.

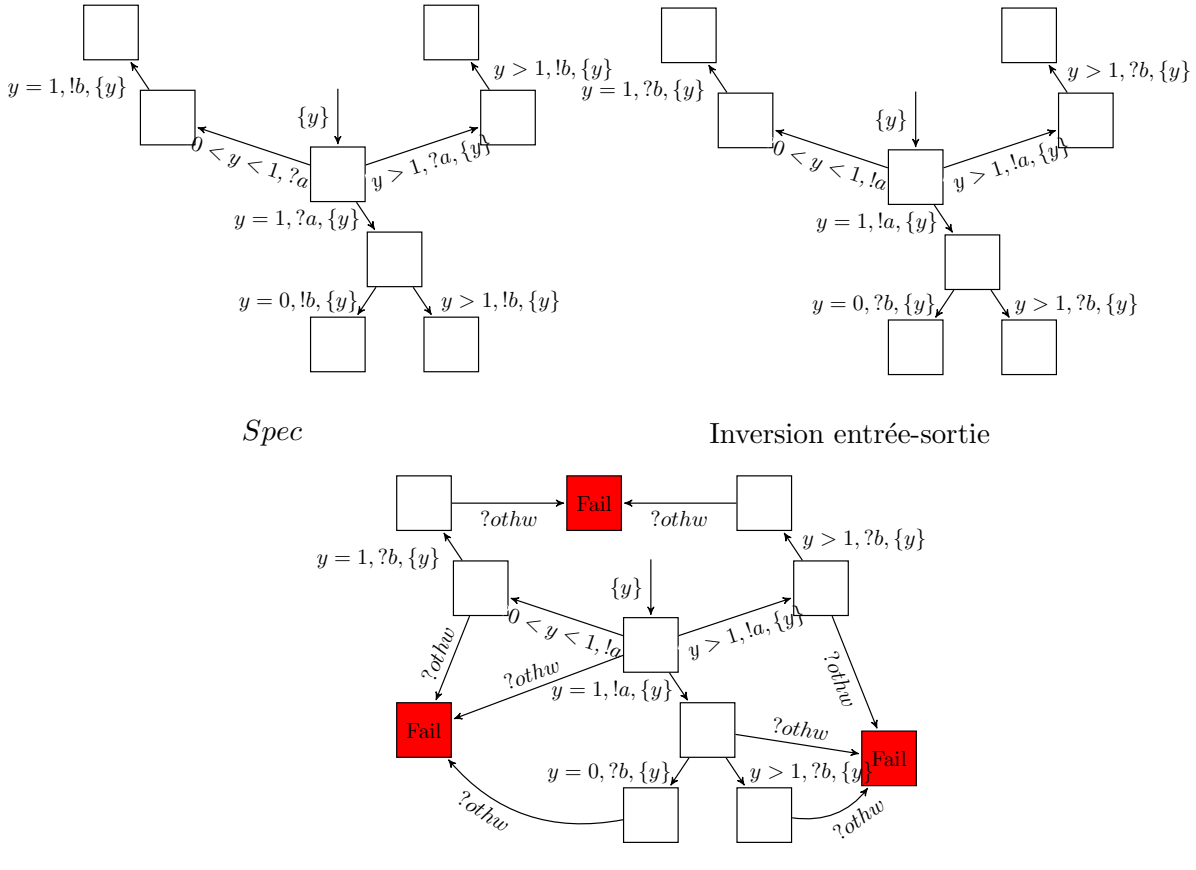
4.5 Génération de cas de test

Dans cette section, on propose une méthode pour construire à partir d'une spécification, un testeur associé permettant de générer le test. Cela à partir d'une spécification *a priori* non-déterministe.

Testeur canonique. Si la spécification est déterministe, pour générer les cas de test, on construit un testeur canonique qu'il suffira de déplier pour obtenir des cas de test corrects et exacts. On pourra également s'en servir pour générer des tests à la volée. Pour cela, on parcourt le testeur canonique tant qu'on n'a pas observé d'erreur de conformité (verdict **Fail**). Si on arrête le test sans avoir émis **Fail**, on donne le verdict **Pass**. Le testeur canonique est obtenu à partir d'une spécification déterministe en inversant les entrées et les sorties pour interagir avec l'implémentation à tester et en complétant en entrées (donc pour les sorties de l'implémentation) vers une localité puits **Fail**. La Fig. 30 illustre cette procédure.

Le déterminisme de la spécification est nécessaire pour l'étape d'ajout des verdicts, sinon on pourrait émettre **Fail** pour une trace acceptée par la spécification sur une exécution différente ayant la même trace. Dans le cas où la spécification n'est pas déterministe, on applique notre approche pour la déterminisation, puis on construit le testeur canonique pour la spécification déterministe approximée. Le résultat permet alors de générer des cas de test corrects et même exacts si Determinisator a gagné le jeu, c'est-à-dire si notre approche a fourni un déterminisé exact.

Adaptation de notre approche à la génération de tests. La définition de notre jeu a été donnée pour un TA en Section 3.2.2, cette définition est naturellement valable sur les TAIO. Pour utiliser la sur-approximation dans le domaine du test, Krichen et Tripakis font l'hypothèse forte que la spécification est complète en entrées. Avec cette hypothèse, **tioco** coïncide avec l'inclusion



Spec Inversion entrée-sortie

Ajout des verdicts

FIG. 30 – Exemple de construction d'un testeur.

de traces et est donc préservée par les sur-approximations. Dans le paragraphe suivant, on propose une adaptation de notre jeu de manière à ce que l'approximation préserve **tioco** sans cette hypothèse. Autrement dit, pour une spécification $Spec$ et une stratégie S de Determinisator pour notre jeu, on veut que $Aut(S)$, l'automate déterministe correspondant à S, soit tel que pour toute implémentation Imp non-bloquante et complète en entrées, Imp est conforme à $Spec$ implique que Imp est conforme à $Aut(S)$. La sur-approximation n'est pas suffisante. En effet, si dans une spécification $Spec$ il existe une localité l dont aucune transition ne sort pour une entrée $?a$, cela signifie que si $?a$ a lieu, le comportement n'est pas spécifié. Si dans une sur-approximation $Spec'$ de $Spec$ on ajoute une transition pour l'entrée $?a$ ayant pour source l, on spécifie plus strictement les comportements et on ne préserve pas la conformité. Pour adapter notre approche, on définit tout d'abord une relation de raffinement sur les spécifications destinée à préserver la correction des tests.

Définition 31. *Étant données deux spécifications $Spec_1$ et $Spec_2$, $Spec_1$ raffine $Spec_2$, noté $Spec_1 \preceq Spec_2$, si :*

- $\forall \sigma \in traces(Spec_2)$, $sorties(Spec_1\ After\ \sigma) \subseteq sorties(Spec_2\ After\ \sigma)$,
- $\forall \sigma \in traces(Spec_1)$, $entrées(Spec_2\ After\ \sigma) \subseteq entrées(Spec_1\ After\ \sigma)$.

Autrement dit, cela signifie que $Spec_2$ sur-approxime les sorties de $Spec_1$ et sous-approxime en entrées. Cette relation est en fait équivalente à la simulation alternée d'automates temporisés [DLL+10] qui est une extension temporisée de la simulation alternée [AHKV98]. On remarque que grâce à l'hypothèse de complétude en entrées des implémentations, pour toute spécification $Spec$ et toute implémentation Imp, Imp **tioco** $Spec$ si et seulement si $Imp \preceq Spec$. La proposition suivante établit que la relation de raffinement préserve la conformité. En corollaire, elle préserve la correction des cas de test.

Proposition 13. *Étant données deux spécifications $Spec_1$ et $Spec_2$, $Spec_1 \preceq Spec_2$ si et seulement si pour tout implémentation Imp non-bloquante et complète en entrées, Imp **tioco** $Spec_1$ implique Imp **tioco** $Spec_2$.*

Démonstration. On montre un résultat plus fort : la transitivité de la relation de raffinement. Grâce à l'équivalence de **tioco** et $\preceq$ sous l'hypothèse que l'implémentation est complète en entrées, cela suffit à prouver la Proposition 13. Soient $Spec_1$, $Spec_2$ et $Spec_3$ trois spécifications telles que $Spec_1 \preceq Spec_2$ et $Spec_2 \preceq Spec_3$. Montrons que $Spec_1 \preceq Spec_3$. Soit $\sigma \in traces(Spec_3)$, montrons que $sorties(Spec_1\ After\ \sigma) \subseteq sorties(Spec_3\ After\ \sigma)$. Si $\sigma \in traces(Spec_3) \cap traces(Spec_2)$ alors, par définition, on a $sorties(Spec_1\ After\ \sigma) \subseteq sorties(Spec_2\ After\ \sigma) \subseteq sorties(Spec_3\ After\ \sigma)$. Sinon $\sigma'.!a \in traces(Spec_3) \setminus traces(Spec_2)$, or $Spec_2 \preceq Spec_3$ donc il existe nécessairement une sortie $!a$ et deux traces σ' et σ'' telles que $\sigma = \sigma'.!a.\sigma''$ avec $\sigma' \in traces(Spec_2)$ et $\sigma'.!a \in traces(Spec_3) \setminus traces(Spec_2)$. Or $Spec_1 \preceq Spec_2$, donc par définition $sorties(Spec_1\ After\ \sigma') \subseteq sorties(Spec_2\ After\ \sigma')$ et donc $\sigma'.!a \in traces(Spec_3) \setminus traces(Spec_1)$. On obtient que $sorties(Spec_1\ After\ \sigma'.!a) = \emptyset$ et donc que $sorties(Spec_1\ After\ \sigma) \subseteq sorties(Spec_3\ After\ \sigma)$. On montre exactement de la même manière que si $\sigma \in traces(Spec_1)$ alors entrées$(Spec_3\ After\ \sigma) \subseteq$ entrées$(Spec_1\ After\ \sigma)$. □

Corollaire 3. *Étant données deux spécifications $Spec_1$ et $Spec_2$ telles que $Spec_1 \preceq Spec_2$, alors si un cas de test est correct pour $Spec_2$, il est correct pour $Spec_1$.*

On modifie notre jeu de façon à ce que pour une spécification $Spec$ sous forme de TAIO et des ressources (k, M), toute stratégie S de Determinisator pour le jeu $\mathcal{G}_{Spec,(k,M)}$ corresponde à une approximation $Aut(S)$ de la spécification telle que $Spec \preceq Aut(S)$. On adapte donc notre approximation pour le test en maintenant la sur-approximation sur les sorties, mais en sous-approximant sur les entrées. La définition du jeu est modifiée pour les successeurs élémentaires

par un choix de Spoiler $(?a, r_Y)$ où $?a$ est une entrée et on étend le coloriage des états. On définit d'abord $Succ_e^?((l, C, b), (?a, r_Y))$ l'ensemble des successeurs élémentaires de (l, C, b) par un choix $(?a, r_Y)$ de Spoiler où $?a$ est une entrée par :

$$Succ_e^?((l, C, b), (?a, r_Y)) = \left\{ (l', C', b') \;\middle|\; \begin{array}{l} \exists l \xrightarrow{g,a,X'}_{\mathcal{A}} l' \wedge ([r_Y \cap C]_{|X} \subseteq g), \\ C' = U_X(r_Y, C, X'), \\ b' = b \end{array} \right\}$$

où $U_X(r_Y, C, X')$ est la mise à jour des relations entre les horloges de X et Y, tout le contexte étant défini en Section 3.2.2. Les successeurs élémentaires par un choix de Spoiler $(!b, r_Y)$ ne sont pas modifiés. Moins formellement, au lieu de marquer les configurations pour lesquelles on sur-approxime la garde, on ne construit que celles sur lesquelles on est exact. Ces configurations sont ainsi marquées uniquement si tous leurs prédécesseurs l'étaient.

Pour préserver les propriétés des stratégies de Determinisator, on étend le coloriage. On grise toujours les états dont toutes les configurations sont marquées, mais on grise également les états obtenus par une sous-approximation stricte. Plus précisément, dès que pour une configuration (l, C, b) d'un état q_S de Spoiler et un choix de Spoiler $(?a, r_Y)$, on a $Succ_e^?((l, C, b), (?a, r_Y)) \neq Succ_e((l, C, b), (?a, r_Y))$, l'état cible de la transition étiquetée par $(?a, r_Y)$ et ayant pour source q_S est grisé (dans *Gray*).

Si on ne grisait pas ces états, on risquerait de ne pas préserver tous les choix de Spoiler pouvant faire perdre Determinisator et une stratégie gagnante pourrait correspondre à une approximation stricte. La Fig. 31 présente un contre-exemple simple. La stratégie de Determinisator consistant à réinitialiser l'horloge y à chaque transition serait gagnante dans le jeu adapté que nous venons de définir sans extension de coloriage. Cette stratégie est représentée entièrement en Fig. 32 et l'automate correspondant à cette stratégie fournissant clairement une approximation stricte de l'automate temporisé de la Fig.31.

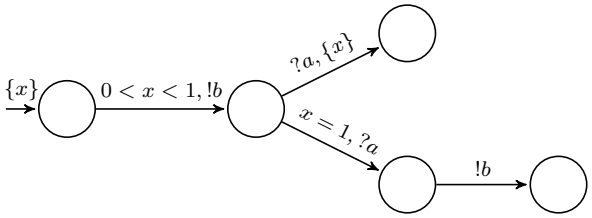


FIG. 31 – Exemple simple d'un TAIO illustrant la nécessité d'étendre le coloriage.

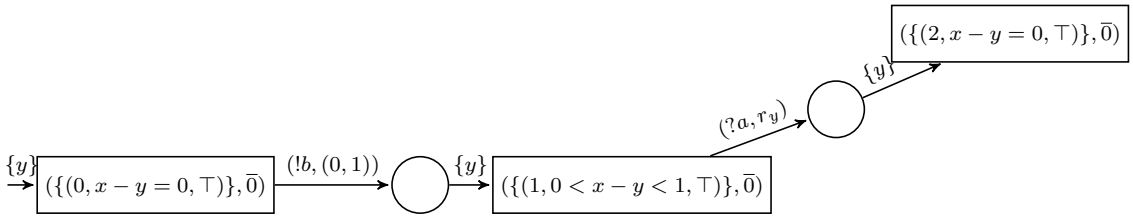


FIG. 32 – Construction d'une stratégie gagnante pour Determinisator correspondant à une approximation inexacte du TA en Fig 31 dans le jeu en n'étendant pas le coloriage.

On pourrait être plus précis dans le coloriage des états, c'est-à-dire en colorier moins, en introduisant une nouvelle valeur pour le marqueur ainsi qu'une nouvelle couleur marquant à

la fois l'appartenance à *Gray* et le fait que les transitions entrantes dans cet état doivent être supprimées pour obtenir le TA $Spec'$ tel que $Spec \preceq Spec'$.

Dans la version initiale du jeu, la couleur dépendait uniquement de l'étiquette d'un état. Au contraire, ici, on ne construit pas les configurations litigieuses lors des sous-approximations sur les entrées, la couleur d'un état ne dépend donc plus seulement des configurations construites. Il y aura donc, si nécessaire, deux copies d'un même état d'un joueur avec des couleurs différentes.

Étant donnée une spécification non-déterministe et pas nécessairement complète en entrées, on peut donc générer des cas de test corrects à partir d'une approximation déterministe obtenue par cette adaptation du jeu. L'exactitude n'est conservée que dans le cas d'une déterminisation exacte. Le test ne pouvant que certifier la non-conformité, c'est bien la correction qui est primordiale.

Bilan. Pour générer des cas de test corrects à partir d'une spécification non-déterministe S, on peut donc adaptater notre approche par le jeu pour obtenir une approximation déterministe en préservant **tioco** et construire le testeur canonique associé à cette approximation. Les cas de test générés à partir de ce testeur canonique seront alors corrects pour S.

4.6 Génération de cas de test avec objectif

Une spécification peut être vaste, spécifier beaucoup de comportements. Dans ce cas, on peut définir un objectif de test pour cibler les comportements à tester. Le verdict sera **Pass** pour une trace si elle ne contient pas d'erreur de conformité par rapport à S et si elle est acceptée par l'objectif. Dans le cas où l'on observe une trace acceptée par la spécification mais refusée par l'objectif, le verdict sera inconclusif (**Inconc**).

4.6.1 Des modèles pour les objectifs de test

Dans cette sous-section, on propose une formalisation des objectifs de test pour une spécification sous forme de TAIO, $S = (L^S, l_0^S, Act^S, X^S, M^S, E^S)$. Cette formalisation est inspirée de celle de [JJRZ05] proposée pour les automates à variables. Pour cela on a besoin d'étendre le modèle des TAIO en permettant l'observation d'un autre TAIO. On définit donc l'extension suivante :

Définition 32 (Syntaxe des OTAIO)**.** *Un* automate temporisé entrée/sortie ouvert *(OTAIO pour* opened timed automaton input/output*) est un tuple* (L, l_0, Act, X, M, E) *où :*

- L *est un ensemble fini de* localités,
- $l_0 \in L$ *est la* localité initiale,
- $Act = Act_? \cup Act_!$ *est un* alphabet fini *(entrée/sortie),*
- $X = X_p \sqcup X_o$ *est l'ensemble des horloges, partitionné en un ensemble* X_p *d'horloges propres et un ensemble* X_o *d'horloges observées,*
- M *est la constante maximale,*
- $E \subseteq L \times Guard_M^X \times (Act \times 2^{X_p}) \times L$ *est un ensemble d'*arêtes.

Les gardes d'un OTAIO portent sur l'ensemble des horloges (propres et observées), en revanche, les réinitialisations ne peuvent concerner que les horloges propres. Les éventuelles réinitialisations des horloges observées sont faites par l'environnement. Elles correspondent à des transitions externes de $\mathcal{A}$ et ne sont pas spécifiées dans $\mathcal{A}$. On remarque qu'un TAIO est un OTAIO dont l'ensemble des horloges observées est vide.

Sémantique des OTAIO. La sémantique des TAIO est étendue avec des transitions externes qui représentent les réinitialisations éventuelles des horloges observées, par l'environnement.

Définition 33. *La* sémantique *d'un OTAIO* $\mathcal{A} = (L, l_0, Act, X, M, E)$ *est donnée par un* système de transitions $(Q,\ q_0,\ \Sigma,\ \rightarrow)$ *où :*

- $Q = L \times \mathbb{R}_+^X$ *est l'ensemble des* états *de* $\mathcal{A}$,
- $q_0 = (l_0, \overline{0})$ *est* l'état initial *de* $\mathcal{A}$,
- $\rightarrow \subseteq Q \times (Act \cup \mathbb{R}_+ \cup 2^{X_o}) \times Q$ *est la* relation de transitions *associée. Elle est composée des* transitions discrètes $(l, v) \xrightarrow{a} (l', v')$ *où* $a \in Act$ *telles qu'il existe* $(l, g, a, X'_p, l') \in E$ *avec* $v \vDash g$ *et* $v' = v_{[X'_p \leftarrow 0]}$, *des* transitions temporelles $(l, v) \xrightarrow{\theta} (l, v + \theta)$ *où* $\theta \in \mathbb{R}_+$ *et des* transitions externes $(l, v) \xrightarrow{X'_o} (l, v_{[X'_o \leftarrow 0]})$ *où* $X'_o \subseteq X_o$ *est arbitraire.*

On définit maintenant la notion de produit de OTAIO pour exprimer le cas où certaines horloges observées d'un OTAIO sont contrôlées par un autre OTAIO.

Définition 34 (Produit)**.** *Deux OTAIO* $\mathcal{O}^i = (L^{\mathcal{O}^i}, l_0^{\mathcal{O}^i}, Act, X^{\mathcal{O}^i}, M^{\mathcal{O}^i}, E^{\mathcal{O}^i}),\ i = 1, 2$ *avec le même alphabet sont* compatibles *pour le produit si* $X_p^{\mathcal{O}^1} \cap X_p^{\mathcal{O}^2} = \emptyset$ *(leurs horloges propres sont disjointes). Dans ce cas,* $\mathcal{O}^1 \times \mathcal{O}^2 = (L^{\mathcal{O}}, l_0^{\mathcal{O}}, Act, X^{\mathcal{O}}, M^{\mathcal{O}}, E^{\mathcal{O}})$ *est un OTAIO défini par :*

- $L^{\mathcal{O}} = L^{\mathcal{O}^1} \times L^{\mathcal{O}^2}$,
- $l_0^{\mathcal{O}} = (l_0^{\mathcal{O}^1}, l_0^{\mathcal{O}^2})$,
- $X^{\mathcal{O}} = X^{\mathcal{O}^1} \cup X^{\mathcal{O}^2}$ *avec* $X_p^{\mathcal{O}} = X_p^{\mathcal{O}^1} \cup X_p^{\mathcal{O}^2}$ *et* $X_o^{\mathcal{O}} = (X_o^{\mathcal{O}^1} \cup X_o^{\mathcal{O}^2}) \setminus X_p^{\mathcal{O}}$,
- $M^{\mathcal{O}} = max(M^{\mathcal{O}^1}, M^{\mathcal{O}^2})$,
- $((l^1, l^2), g, a, X'_p, (l'^1, l'^2)) \in E^{\mathcal{O}}$ *s'il existe* $(l^1, g^1, a, X_p'^1, l'^1) \in E^{\mathcal{O}^1}$ *et* $(l^2, g^2, a, X_p'^2, l'^2) \in E^{\mathcal{O}^2}$ *avec :*
  - $g = g^1 \wedge g^2$,
  - $X'_p = X_p'^1 \cup X_p'^2$.

Si $\mathcal{O}^1$ et $\mathcal{O}^2$ sont compatibles pour le produit, alors $traces(\mathcal{O}^1 \times \mathcal{O}^2) \subseteq traces(\mathcal{O}^1) \cap traces(\mathcal{O}^2)$.

Le modèle des TAIO nous sert à définir un objectif de test pour une spécification qui est autorisé à observer les horloges de la spécification. Dans tout ce qui suit, nous considérerons la spécification $S = (L^S, l_0^S, Act^S, X_p^S \cup X_o^S, M^S, E^S)$

Définition 35. *Un* objectif de test *pour une spécification* S *est un OTAIO,* $TP = (L^{TP}, l_0^{TP}, Act^{TP}, X_p^{TP} \cup X_o^{TP}, M^{TP}, E^{TP})$, *avec* $X_o^{TP} = X_p^S$, *muni d'un sous-ensemble* $\textbf{\textit{Accept}} \subseteq L^{TP}$ *de localités puits.* TP *est complet sauf pour les localités de* ***Accept****.*

Un objectif de test TP pour une spécification S définit un sous-ensemble $Atraces(S, TP)$ de traces de S acceptées, et un sous-ensemble $Rtraces(S, TP)$ de traces de S refusées (celles qui ne peuvent pas être prolongées en une trace acceptée) :

$$Atraces(S, TP) = traces_{L^S \times Accept}(S \times TP),$$

$$Rtraces(S, TP) = traces(S) \setminus pref_{\leq}(Atraces(S, TP)),$$

où, étant donné un ensemble de traces Tr, $pref_{\leq}(Tr)$ est l'ensemble des traces pouvant être prolongées en un élément de Tr.

On affine maintenant la notion de cas de test qui sera associé à une spécification et à un objectif. Là encore, il satisfait trois propriétés, son alphabet est le miroir de celui de la spécification, il accepte toutes les sorties de l'implémentation à tester et il est muni de trois localités puits représentant les verdicts émis, ainsi que d'un ensemble de couples (localité, région) représentant les inconclusifs.

Définition 36. *Un* cas de test *pour une spécification S est un automate temporisé entrées/sorties déterministe, $TC = (L^{TC}, l_0^{TC}, Act^{TC}, X^{TC}, M^{TC}, E^{TC})$ muni de trois localités sans successeurs* ***Pass***, ***Fail***, *et* ***InconcLoc*** *dans L^{TC} et d'un ensemble* ***Inconc*** *de couples (l, r) où $l \in L^{TC}$ et r une région sur X^{TC} tels que :*

- $Act_!^{TC} = Act_?^S$ *et* $Act_?^{TC} = Act_!^S$,
- TC *est complet en entrées,*
- $\{\textbf{\textit{InconcLoc}}\} \times Reg_{X^{TC}}^{M^{TC}} \subseteq \textbf{\textit{Inconc}}$.

Un cas de test TC pour une spécification S définit un ensemble $traces_{\textbf{Pass}}(TC)$ de traces acceptées, un ensemble $traces_{\textbf{Fail}}(TC)$ de traces refusées et un ensemble $traces_{\textbf{None}}(TC)$ de traces sans verdict. Dans le cas où on utilise un objectif, le cas de test définit également un ensemble $traces_{\textbf{Inconc}}(TC)$ de traces inconclusives, les traces qui ne constituent pas d'erreurs de conformité mais qui sont refusées par l'objectif :

- $traces_{\textbf{Pass}}(TC) = traces_{\{\textbf{Pass}\}}(TC)$,
- $traces_{\textbf{Fail}}(TC) = traces_{\{\textbf{Fail}\}}(TC)$,
- $traces_{\textbf{Inconc}}(TC) = \{r \in (\mathbb{R}_+.Act)^+.\mathbb{R}_+ \mid \exists R,\ q_0 \xrightarrow{r} (l, v)\ \wedge v \in R \wedge (l, R) \in \textbf{Inconc}\}$,
- $traces_{\textbf{None}}(TC) = traces(TC) \setminus (traces_{\{\textbf{Pass}\}}(TC) \cup traces_{\{\textbf{Fail}\}}(TC) \cup traces_{\{\textbf{Inconc}\}}(TC))$.

On écrit Imp *fails* TC si $L^{Imp} \times \{\textbf{Fail}\} \cap Reach_{(l_0^{TC}, \overline{0})}(Imp \| TC) \neq \emptyset$. Comme dans la Section 4.4, on définit la correction et l'exactitude d'un cas de test. La correction est définie de la même façon, en revanche, l'exactitude spécifie les emissions du verdict **Inconc**.

Définition 37. *Un cas de test TC est* correct *pour la spécification S et l'objectif TP si* ***Fail*** *n'est émis que si TC observe une sortie non-spécifiée après une trace de S, c'est-à-dire* $(traces_{\textbf{Fail}}(TC)) \subseteq ((traces(S).Act_?.\mathbb{R}_+) \cap \overline{traces(S)})$.

Définition 38. *Un cas de test TC est* exact *pour la spécification S et l'objectif TP si :*

- ***Fail*** *est émis dès que TC observe une erreur de conformité à S, autrement dit* $((traces_{\textbf{None}}(TC).Act_?.\mathbb{R}_+) \cap \overline{traces(S)}) \subseteq (traces_{\textbf{Fail}}(TC))$,
- ***Inconc*** *est émis si et seulement si la trace observée par TC est acceptée par S mais est refusée par TP, c'est-à-dire* $traces_{\textbf{Inconc}}(TC) = traces_{\textbf{None}}(TC).Act_?.\mathbb{R}_+ \cap Rtraces(S, TP))$.

4.6.2 Construction du testeur

On voit maintenant comment construire l'équivalent du testeur canonique, en présence d'un objectif de test. On se place tout d'abord dans le cas où la spécification et l'objectif de test sont déterministes. Le produit de l'objectif et de la spécification est alors déterministe. Dans ce cas, le testeur est obtenu par inversion des entrées et des sorties et ajout des verdicts. L'ajout des verdicts est plus subtil que sans objectif. Pour émettre **Inconc** au plus vite, on calcule la co-accessibilité d'**Accept**. On dit qu'un état q est co-accessible d'un état q' si q' est accessible depuis q.

Définition 39. *L'ensemble des états* co-accessibles *de l'état (l', v') dans un TAIO $\mathcal{A} = (L^{\mathcal{A}}, l_0^{\mathcal{A}}, Act^{\mathcal{A}}, X^{\mathcal{A}}, M^{\mathcal{A}}, E^{\mathcal{A}})$, noté $co\text{-}Reach_{(l', v')}(\mathcal{A})$ est défini par : $co\text{-}Reach_{(l', v')}(\mathcal{A}) = \{(l, v) \in L^{\mathcal{A}} \times \mathbb{R}_+ \mid \exists r \in (\mathbb{R}_+.Act)^+.\mathbb{R}_+,\ (l, v) \xrightarrow{r} (l', v')\}$. On dit qu'un état est* co-accessible *d'une localité l, s'il existe une valuation v telle que l'état soit co-accessible de (l, v).*

On remarque que grâce aux propriétés des régions, si v et v' sont des valuations d'une région R, alors pour toute localité l et tout état q, (l, v) est co-accessible de q si et seulement si (l, v') est co-accessible de q.

Pour ajouter les verdicts, on affine tout d'abord les gardes des transitions de l'automate pour calculer la co-accessibilité d'**Accept**. Les transitions qui mènent à un état non co-accessible

d'**Accept** sont détournées vers l'état puits **InconcLoc**. De plus, dans un soucis d'optimalité, on définit le verdict **Inconc** sur des couples (localité, région). Effectivement, dans certaines localités, en laissant s'écouler le temps, les gardes peuvent ne plus jamais être tirables, on est alors passé de co-accessible à non-co-accessible de **Accept** sans changer de localité. En fait, une fois toutes les gardes affinées, on peut ajouter le verdict **Inconc** sur tous les couples (l, R) où $l \in L \setminus (L^S \times \mathbf{Accept})$ et R une région telles qu'il n'existe pas de transition ayant pour source l, tirable dans R ou dans un de ses successeurs temporels et ayant pour cible une autre localité que **InconcLoc**. Enfin, on ajoute une localité **Fail** et on complète l'automate en entrées (sorties de l'implémentation) vers cette localité, les localités de $L^S \times \mathbf{Accept}$ correspondent aux émissions du verdict **Pass**.

Plus formellement, l'étape où l'on affine les gardes pour qu'elles préservent la co-accessibilité se fait par le calcul du plus petit point fixe pour l'inclusion des gardes tel que pour toute garde g', $g' = g' \cap \mathit{pred}(\mathit{co\text{-}Reach}(\mathbf{Accept}))$. Si on note g la garde initiale affinée en g' alors on ajoute la transition ayant même source et même action que la transition correspondante, **InconcLoc** pour cible et $g \setminus g'$ pour garde. Cette opération sur les gardes est classique, voir par exemple [Bou09].

La Fig. 33 représente une spécification et un objectif de test sur lesquels on illustre la construction du testeur Fig 34.

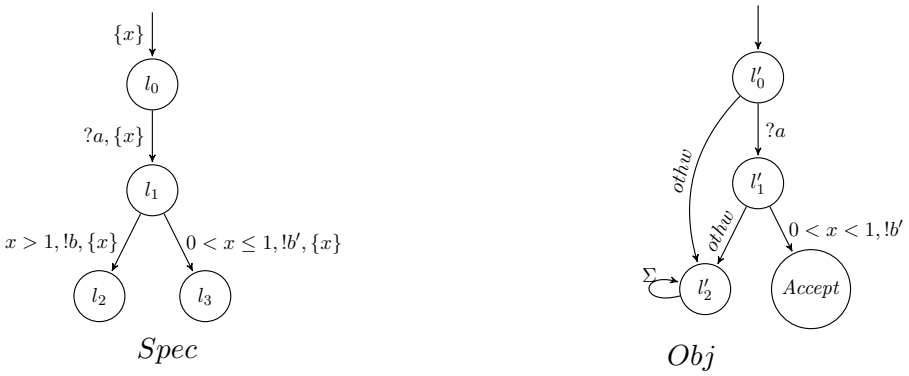


FIG. 33 – Exemple d'une spécification et d'un objectif de test.

Dans le cas général, la spécification comme l'objectif de test sont potentiellement non-déterministes. On utilise alors avant tout l'adaptation de notre approche présentée en Section 4.4, en affinant la coloration des états du jeu. Plus précisément, on définit une troisième couleur *LGray* pour les états contenant des configurations dont la localité est dans $L^S \times \mathbf{Accept}$ mais toutes marquées. On ne peut donc pas donner un verdict **Accept** sûr car même si la trace est dans le produit, on n'a pas l'assurance que c'est une trace acceptée par l'objectif. On souhaite donc privilégier une stratégie restant dans des états blancs, à une passant par un tel état, mais on préfère toujours une stratégie sans état de *Gray*. On applique alors la construction précédente à l'approximation déterministe du produit. Le testeur obtenu par cette approche permet de donner des verdicts précis grâce à la couleur des états du jeu. On peut faire la différence entre un **Pass** exact (donné sur des états blancs) et un potentiellement approximé (donné en passant dans un état *Gray* ou *LGray*).

4.7 Modélisation de l'urgence

Pour la modélisation de systèmes temps-réel, il est particulièrement intéressant d'exprimer la notion d'urgence, que ce soit par des étiquettes sur les transitions indiquant le degré d'urgence ou par des invariants dans les localités. Krichen et Tripakis utilisent un modèle avec degrés d'urgence sur les transitions (*urgent*, *reportable*, *paresseux*), mais toutes les transitions

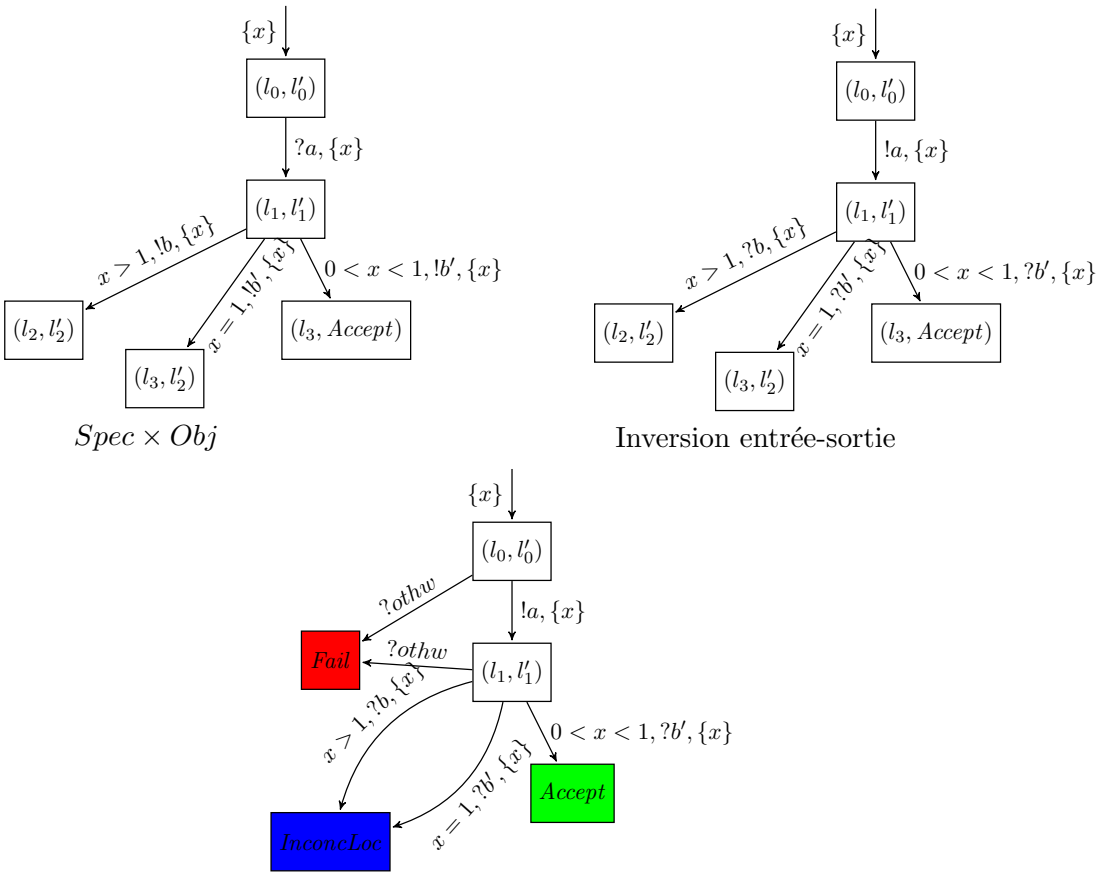


FIG. 34 – Construction du testeur associé à *Spec* et *Obj*.

de leur sur-approximation sont paresseuses. Pour le moment, la définition de notre approche ne permettrait pas non plus de préserver l'expressivité d'un tel modèle. Mais il serait possible de préserver la notion d'urgence dans notre déterminisation. En effet, il suffit de calculer les successeurs élémentaires en faisant attention à la compatibilité avec les urgences et d'ensuite attribuer l'urgence la plus restrictive qui soit compatible.

4.8 Conclusion de la Section 4

Nous avons tout d'abord proposé une formalisation précise des concepts utilisés dans le domaine du test de conformité. Puis nous avons donné une méthode de construction d'un testeur à partir duquel générer des cas de test corrects efficacement. Cette méthode est basée sur une adaptation de notre approche pour la déterminisation présentée en Section 3. Nous avons ainsi donné un cadre formel à la génération de test d'automates temporisés avec et sans objectif.

5 Conclusion

Les objectifs de ce stage étaient doubles, d'une part améliorer la déterminisation des automates temporisés et d'autre part formaliser le test et l'utilisation de la déterminisation des automates temporisés pour la génération de test.

Tout d'abord, nous souhaitions combiner deux approches existantes pour la déterminisation : une procédure de déterminisation terminant sur certaines classes d'automates et une sur-approximation déterministe. Le but était de produire, sur un ensemble donné d'horloges, un déterminisé exact si possible et une sur-approximation sinon. Autrement dit, la procédure devait terminer sur tous les automates temporisés tout en préservant l'exactitude le plus souvent possible. Nous avons développé une méthode inspirée de l'approche présentée dans [BCD05] pour le diagnostic qui, en plus de réunir ces avantages, améliore les deux approches existantes.

De plus, nous souhaitions donner un cadre formel à la génération de test et préciser l'utilisation de la déterminisation à la problématique du test. Nous nous sommes inpirés de la formalisation de [JJRZ05] pour les automates étendus avec des variables et nous avons donné des définitions précises des concepts tels que les cas de test et les objectifs de test. Nous avons ensuite affiné notre méthode de déterminisation pour obtenir une approximation préservant la conformité.

Le travail sur la déterminisation est terminé, il fera l'objet d'une présentation à MOVEP2010 et un article est en cours de rédaction. En revanche, nous souhaitons poursuivre la partie test du stage en adaptant notre approche aux modèles avec urgence et proposer une méthode de génération automatique de test. Ce travail sera poursuivi cet été avec la collaboration de Moez Krichen.

Références

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

[AFH94] R. Alur, L. Fix, and T. A. Henzinger. A determinizable class of timed automata. In *Proceedings of International Conference on Computer Aided Verification (CAV'94)*, volume 818 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 1994.

[AHKV98] R. Alur, T. A. Henzinger, O. Kupferman, and M. Y. Vardi. Alternating refinement relations. In *Proceedings of International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 of *Lecture Notes in Computer Science*, pages 163–178. Springer-Verlag, 1998.

[BB05] L. Brandan Briones and E. Brinksma. A test generation framework for quiescent real-time systems. In *Proceedings of International Workshop, Formal Approaches to Software Testing, FTAIO 2004*, volume 3395 of *Lecture Notes in Computer Science*. Springer, 2005.

[BBBB09] C. Baier, N. Bertrand, P. Bouyer, and T. Brihaye. When are timed automata determinizable ? In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP'09)*, volume 5556 of *Lectures Notes in Computer Science*, pages 43–54. Springer, 2009.

[BCD05] P. Bouyer, F. Chevalier, and D. D'Souza. Fault diagnosis using timed automata. In *Proceedings of International Conference on Foundations of Software Science and Computational Structures (FOSSACS'05)*, volume 3441 of *Lecture Notes in Computer Science*, pages 219–233. Springer, 2005.

[BDGP98] B. Bérard, V. Diekert, P. Gastin, and A. Petit. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36(2–3) :145–182, 1998.

[Bou09] P. Bouyer. *From Qualitative to Quantitative Analysis of Timed Systems*. Mémoire d'habilitation, Université Paris 7, Paris, France, January 2009.

[DLL+10] A. David, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski. Timed i/o automata : a complete specification theory for real-time systems. In *Proceedings of International Conference on Hybrid Systems : Computation and Control (HSCC'10)*, pages 91–100. ACM, 2010.

[DY96] C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In *Proceedings of IEEE Real-Time Systems Symposium (RTSS'96)*, pages 73–81. IEEE, 1996.

[GTW] E. Grädel, W. Thomas, and T. Wilke, editors.

[JJRZ05] B. Jeannet, T. Jéron, V. Rusu, and E. Zinovieva. Symbolic test selection based on approximate analysis. In *Proceedings of International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'05)*, volume 3440 of *Lecture Notes in Computer Science*, pages 349–364. Springer, 2005.

[KT09] M. Krichen and S. Tripakis. Conformance testing for real-time systems. *Formal Methods in System Design*, 34(3) :238–304, 2009.

[NS03] B. Nielsen and A. Skou. Automated test generation from timed automata. *International Journal on Software Tools for Technology Transfer, vol 5*, 2003.

[SPKM08] P. Vijay Suman, Paritosh K. Pandya, Shankara Narayanan Krishna, and Lakshmi Manasa. Timed automata with integer resets: Language inclusion and expressiveness. In *Proceedings of International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'08)*, volume 5215 of *Lecture Notes in Computer Science*, pages 78–92. Springer, 2008.

[ST08] J. Schmaltz and J. Tretmans. On conformance testing for timed systems. In *Proceedings of International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'08)*, volume 5215 of *Lecture Notes in Computer Science*, pages 250–264. Springer, 2008.

[Tre96] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. In *Software-Concepts and Tools 17(3), 103-120*, 1996.

[Tri06] S. Tripakis. Folk theorems on the determinization and minimization of timed automata. *Information Processing Letters*, 99(6) :222–226, 2006.

Annexe technique

Dans cette annexe, on présente les preuves omises au cours de la Section 3.

Preuve du Théorème 1

Démonstration. Soit S une stratégie de Determinisator pour le jeu $\mathcal{G}_{\mathcal{A},(1,M_y)}$, montrons par récurrence que pour tout $n \in \mathbb{N}$:
P_n = Pour chaque exécution de longueur n de $\mathcal{A}$, il existe une exécution de $Aut(S)$ dont les localités contiennent les configurations correspondant aux localités de l'exécution dans $\mathcal{A}$, et ayant la même trace.
Formellement :

$$P_n = \left(\begin{array}{l} \text{Pour toute exécution de } \mathcal{A}\ (l_0, v_0 = \overline{0}) \xrightarrow{a_0,t_0} (l_1, v_1) \xrightarrow{a_1,t_1} \dots \xrightarrow{a_{n-1},t_{n-1}} (l_n, v_n), \text{il existe une} \\ \text{exécution de } Aut(S)((\{(l_0, \overrightarrow{\overline{0}_{X\cup\{y\}}}, \top)\}, \{0\}), \tilde{v}_0) \xrightarrow{a_0,t_0} ((\{(l_{j_{(1)}}, C_{j_{(1)}}, b_{j_{(1)}})_{j_{(1)}\in J_{(1)}}\}, r_{y,(1)}), \tilde{v}_1) \\ \xrightarrow{a_1,t_1} \dots \xrightarrow{a_{n-1},t_{n-1}} ((\{(l_{j_{(n)}}, C_{j_{(n)}}, b_{j_{(n)}})_{j_{(n)}\in J_{(n)}}\}, r_{y,(n)}), \tilde{v}_n) \text{ telle que pour tout } i \in [\![1, n]\!], \\ \text{il existe } j_{0(i)} \in J_{(i)} \text{ tel que } l_{j_{0(i)}} = l_i \text{ et } (v_i, \tilde{v}_i) \in C_{j_{0(i)}} \end{array} \right).$$

Montrons P_0. La seule exécution de longueur 0 de $\mathcal{A}$ est $(l_0, \overline{0})$ et l'état initial de $Aut(S)$ est $v_{init} = (\{(l_0, \overrightarrow{\overline{0}_{X\cup\{y\}}}, \top)\}, \{0\})$ de plus, $\overline{0} \in \overrightarrow{\overline{0}_{X\cup\{y\}}}$. P_0 est donc vraie.

Soit $n \in \mathbb{N}$, supposons P_n vraie et montrons l'hérédité de cette propriété. Soit une exécution de $\mathcal{A}$ $\rho = (l_0, v_0 = \overline{0}) \xrightarrow{a_0,t_0} \dots \xrightarrow{a_{n-1},t_{n-1}} (l_n, v_n) \xrightarrow{a_n,t_n} (l_{n+1}, v_{n+1})$ de longueur $n+1$, alors $(l_0, v_0 = \overline{0}) \xrightarrow{a_0,t_0} \dots \xrightarrow{a_{n-1},t_{n-1}} (l_n, v_n)$ est une exécution de longueur n de $\mathcal{A}$, et par hypothèse de récurrence, il existe une exécution de $Aut(S)$

$$((\{(l_0, \overrightarrow{\overline{0}_{X\cup\{y\}}}, \top)\}, \{0\}), \tilde{v}_0) \xrightarrow{a_0,t_0} \dots \xrightarrow{a_{n-1},t_{n-1}} ((\{(l_{j_{(n)}}, C_{j_{(n)}}, b_{j_{(n)}})_{j_{(n)}\in J_{(n)}}\}, r_{y,(n)}), \tilde{v}_n)$$

telle que pour tout $i \in [\![1, n]\!]$, il existe $j_{0(i)} \in J_{(i)}$ tel que $l_{j_{0(i)}} = l_i$ et $(v_i, \tilde{v}_i) \in C_{j_{0(i)}}$. En particulier, il existe $j_{0(n)} \in J_{(n)}$ tel que $l_{j_{0(n)}} = l_n$ et $(v_n, \tilde{v}_n) \in C_{j_{0(n)}}$. Par construction du jeu, il nous suffit, de montrer qu'il existe un successeur temporel $r'_{y,(n)}$ de $r_{y,(n)}$ tel que $v_n + t_n \in [r'_{y,(n)} \cap C_{j_{0(n)}}]_{|X} \cap g$. ρ est une exécution de $\mathcal{A}$ donc il existe une transition $l_n \xrightarrow{g,a_n,X'} l_{n+1}$ dans $\mathcal{A}$ telle que $v_n + t_n \in g$. De plus, pour tout $v_y \in \mathbb{R}_+$, $(v_n, v_y) \in C_{j_{0(n)}}$ implique $(v_n + t_n, v_y + t_n) \in C_{j_{0(n)}}$. Or $(v_n, \tilde{v}_n) \in C_{j_{0(n)}}$, on conclut alors en définissant $r'_{y,(n)}$ comme l'unique région sur $\{y\}$ contenant $\tilde{v}_n + t_n$. On a alors $((\{(l_{j_{(n)}}, C_{j_{(n)}}, b_{j_{(n)}})_{j_{(n)}\in J_{(n)}}\}, r_{y,(n)}), \tilde{v}_n) \xrightarrow{a_n,t_n} ((\{(l_{j_{(n+1)}}, C_{j_{(n+1)}}, b_{j_{(n+1)}})_{j_{(n+1)}\in J_{(n+1)}}\}, r_{y,(n+1)}), \tilde{v}_{n+1})$ et $j_{0(n+1)} \in J_{(n+1)}$ tel que $l_{j_{0(n+1)}} = l_{n+1}$ et $(v_{n+1}, \tilde{v}_{n+1}) \in C_{j_{0(n+1)}}$ par définition de la mise à jour des relations entre y et les horloges de X.

On a donc montré par récurrence que P_n est vraie pour tout $n \in \mathbb{N}$. La trace de l'exécution dans $Aut(S)$ est la même que celle dans $\mathcal{A}$, on a donc bien $traces(\mathcal{A}) \subseteq traces(Aut(S))$. □

Preuve du Théorème 2

Démonstration. Soit S une stratégie gagnante de Determinisator pour le jeu $\mathcal{G}_{\mathcal{A},(1,M_y)}$. Soit une exécution de $Aut(S)$,

$$\rho = ((\{(l_{j_{(0)}}, C_{j_{(0)}}, b_{j_{(0)}})_{j_{(0)}\in J_{(0)}}\}, r_{y,(0)}), \tilde{v}_0) \xrightarrow{a_0,t_0} ((\{(l_{j_{(1)}}, C_{j_{(1)}}, b_{j_{(1)}})_{j_{(1)}\in J_{(1)}}\}, r_{y,(1)}), \tilde{v}_1) \xrightarrow{a_1,t_1} \dots \xrightarrow{a_n,t_n} ((\{(l_{j_{(n+1)}}, C_{j_{(n+1)}}, b_{j_{(n+1)}})_{j_{(n+1)}\in J_{(n+1)}}\}, r_{y,(n+1)}), \tilde{v}_{n+1}).$$

Montrons qu'il existe une exécution de $\mathcal{A}$ ayant même trace que ρ. S est une stratégie gagnante, donc toutes ses localités contiennent au moins une configuration marquée $\top$, c'est le cas en particulier de $(\{(l_{j_{(n+1)}}, C_{j_{(n+1)}}, b_{j_{(n+1)}})_{j_{(n+1)}\in J_{(n+1)}}\}, r_{y,(n+1)})$. On note l_{n+1} la localité de la configuration considérée. Cela signifie que cette configuration est un successeur élémentaire d'une configuration marquée $\top$ par une transition de $\mathcal{A}$ sans faire de sur-approximation. On note l_n la localité de cette configuration. Par récurrence directe, $(l_0, \overline{0}) \xrightarrow{a_0,t_0} (l_1, v_1) \xrightarrow{a_1,t_1} \dots \xrightarrow{a_n,t_n} (l_{n+1}, v_{n+1})$ est une exécution de $\mathcal{A}$. Cette exécution a même trace que ρ. □

Calcul explicite de la mise à jour des relations d'horloges En Section 3.2, dans notre définition des transitions, on a défini la mise à jour des relations entre horloges comme la plus petite conjonction vérifiant certaines propriétés. On définit, maintenant explicitement les mises à jour, étant donnés C représentant les relations entre y et les horloges de X, r_y la région sur $\{y\}$ dans laquelle on se trouve, X' les horloges de $\mathcal{A}$ qui sont réinitialisées et un choix p de Determinisator. Rappelons que les relations sont sous forme de conjonctions finies de formules atomiques de la forme $x - y \sim c$ où $x \in X$, $\sim \in \{=, <, >\}$ et $c \in [\![-M_y, M_X]\!]$. On note, pour $x \in X$, $C(x)$ la conjonction des contraintes sur x dans C.

– Réinitialisation des horloges de $X' \subseteq X$:

Pour les horloges de X qui ne sont pas réinitialisées ($X \setminus X'$), les relations avec y sont pas modifiées car y n'a pas été réinitialisée. En revanche, il faut définir la mise à jour pour les horloges de X', c'est le rôle de u_X.

$$U_X(r_y, C, X') = (C_{|X\setminus X'}) \ \wedge \ (\wedge_{x\in X'} \ u_X(r_y, x)) \text{ où}$$

$$\text{avec } u_X(r_y, x) = \begin{cases} x - y = -c & \text{si } r_y = \{c\},\ c \in [\![0, M_y]\!] \\ (x - y < -c) \ \wedge \ (x - y > -(c+1)) & \text{si } r_y = (c, c+1),\ c \in [\![0, M_y - 1]\!] \\ x - y < -M_y & \text{si } r_y = (M_y, \infty) \\ \emptyset & \text{sinon} \end{cases}.$$

– Réinitialisation de y :

La réinitialisation de y implique la modification éventuelle des relations avec chaque horloge de X, c'est le rôle de u_y.

$$U_y(r_y, C, \{y\}) = (\wedge_{x\in X\setminus X'} \ u_y(r_y, x, C)) \text{ où } u_y \text{ est défini ci-dessous.}$$

Comme dans la définition donnée en Section 3.2, on commence par définir la mise à jour $\tilde{u}_y$ sans prendre en compte les constantes M_X et M_y. On la compose ensuite avec une fonction *utile* qui rétablit ces constantes extrémales pour définir u_y.

$$\tilde{u}_y(r_y, x, C) = \begin{cases} x - y \sim c' + c & \text{si } r_y = \{c\} \text{ et } C(x) = x - y \sim c' \\ (x - y > c' + c) \ \wedge \ (x - y < d' + c) & \text{si } r_y = \{c\} \text{ et } C(x) = (x - y > c') \ \wedge \ (x - y < d') \\ (x - y > c' + c) \ \wedge \ (x - y < c' + (c+1)) & \text{si } r_y = (c, c+1) \text{ et } C(x) = x - y = c' \\ (x - y < c' + (c+1)) & \text{si } r_y = (c, c+1) \text{ et } C(x) = x - y < c' \\ (x - y > c' + c) \ \wedge \ (x - y < (d') + (c+1)) & \text{si } r_y = (c, c+1) \text{ et } C(x) = (x - y > c') \ \wedge \ (x - y < d') \\ x - y > c' + M_y & \text{si } r_y = (M_y, \infty) \text{ et } C(x) = \begin{cases} x - y = c' \\ x - y > c' \\ (x - y > c') \ \wedge \ (x - y < d')) \end{cases} \\ \emptyset & \text{sinon} \end{cases}$$

$$u_y = utile \circ \tilde{u}_y$$

$$\text{avec } utile(f) = \begin{cases} x - y \sim c & \text{si } f = (x - y \sim c) \text{ et } c \in [\![-M_y, M_X]\!] \\ utile(x - y < c) \ \wedge \ utile(x - y > d) & \text{si } f = (x - y < c) \ \wedge \ (x - y > d) \text{ et } c, d \in [\![-M_y, M_X]\!] \\ x - y < -M_y & \text{si } f = (x - y \sim c) \text{ et } c < -M_y \wedge \ \sim \in \{=, <\} \\ x - y > M_X & \text{si } f = (x - y \sim c) \text{ et } c > M_X \wedge \ \sim \in \{=, >\} \end{cases}.$$

– Aucune réinitialisation :

Dans ce cas, les relations ne changent pas, $U_X(r_y, C, \emptyset) = C$ et $U_y(r_y, C, \emptyset) = C$.

À partir de cette expression des relations, étant donnés C et une région r_y sur y, il est possible d'exprimer la garde induite sur X, $[r_y \cap C]_{|X}$, comme conjonction de formules atomiques. On justifie alors aisément qu'il n'est pas utile de conserver les contraintes avec une constante hors de l'intervalle $[\![-M_y, M_X]\!]$ pour l'expression des C car elles ne serviraient qu'à exprimer des gardes non pertinentes. En effet, si l'on souhaite exprimer la garde $x \sim c$ grâce à la relation $x - y \sim' c'$ où $c \leq M_X$, $c' < -M_y$ ou $c' > M_X$ et $(\sim, \sim') \in \{=, <, >, \leq, \geq\}$, si on obtient une garde non triviale, c'est une formule atomique $y \sim'' c - c'$ où $\sim'' \in \{=, <, >, \leq, \geq\}$:

– si $c' < -M_y$, on a $c - c' > 0 - (-M_y) = M_y$ ce qui signifie que la garde est interdite car elle ne respecte pas la précision,

– si $c' > M_X$, on a $c - c' < M_X - M_X = 0$ ce qui signifie que la garde est triviale.

Preuve du Théorème 5 Cette preuve utilise le lemme technique suivant :

Lemme 1. *Soient C une conjonction finie de formules atomiques de la forme $x - y \sim c$ où $(x, y) \in X \times Y$, $\sim \in \{=, <, >\}$ et $c \in [\![-M_Y, M_X]\!]$, r une région sur Y. Si $(\sigma, \tilde{r}, \tilde{C}) \in Y^Y \times Reg^Y_{M_Y} \times \mathcal{P}(\mathbb{R}_+^{X \cup Y})$ tels que $\tilde{r} \circ \sigma = r$ et $\tilde{C} \circ \sigma = C$ en étendant σ comme un élément de $(Y \cup X)^{(Y \cup X)}$ par l'identité sur X alors $[\tilde{C} \cap \tilde{r}]_{|X} \subseteq [C \cap r]_{|X}$.*

Démonstration. Soit $v \in [\tilde{C} \cap \tilde{r}]_{|X}$. Alors il existe $\tilde{v}_Y \in \tilde{r}$ tel que $(v, \tilde{v}_Y) \in \tilde{C}$. Par hypothèse, on a alors $\tilde{v}_y \circ \sigma \in r$ et $(v, \tilde{v}_Y) \circ \sigma \in C$ or $(v, \tilde{v}_Y) \circ \sigma = (v, \tilde{v}_Y \circ \sigma)$ donc $v \in [C \cap r]_{|X}$. □

Démonstation du Théorème 5. Soit S une stratégie positionnelle de Determinisator pour $\mathcal{G}_{\mathcal{A},(k,M_Y)}$. Soit $\mathcal{B}$ l'automate obtenu en appliquant la Proposition 12 à $Aut(S)$. Cet automate induit une stratégie non positionnelle pour notre jeu. On note $\tilde{S}$ la stratégie positionnelle faisant les premiers choix induits par $\mathcal{B}$. On montre que cette stratégie est meilleure que S. Soit S_S une stratégie de Spoiler. Montrons par récurrence que pour tout $n \in \mathbb{N}$:
P_n = Si la partie de longueur n où Spoiler suit la stratégie S_S et Determinisator suit $\tilde{S}$ ne passe pas deux fois dans le même état de Determinisator alors la distance à *Gray* de la partie $(S_S, \tilde{S})$ est plus grande que celle de (S_S, S).

Formellement :

$$P_n = \left(\begin{array}{l} \text{Les parties de longueur } n \text{ ne passant pas deux fois par un même état de Determinisator} \\ \text{dans } \tilde{S} \text{ où on a fusionné les tours de jeux des deux joueurs et où Determinisator suit} \\ \text{respectivement } S \text{ et } \tilde{S} \text{ tandis que Spoiler suit } S_S \text{ sont telles que :} \\ (S_S, S) : (\{(l_0, \overrightarrow{0_{X \cup \{y\}}}, \top)\}, \{0\}), v_0) \xrightarrow{a_0, t_0} (\{(l_{j_{(1)}}, C_{j_{(1)}}, b_{j_{(1)}})_{j_{(1)} \in J_{(1)}}\}, r_{Y,(1)}), v_1) \xrightarrow{a_1, t_1} \ldots \\ \xrightarrow{a_{n-1}, t_{n-1}} (\{(l_{j_{(n)}}, C_{j_{(n)}}, b_{j_{(n)}})_{j_{(n)} \in J_{(n)}}\}, r_{Y,(n)}), v_n) \\ (S_S, \tilde{S}) : (\{(l_0, \overrightarrow{0_{X \cup \{y\}}}, \top)\}, \{0\}), \tilde{v}_0) \xrightarrow{a_0, t_0} (\{(l_{j_{(1)}}, \tilde{C}_{j_{(1)}}, \tilde{b}_{j_{(1)}})_{j_{(1)} \in \tilde{J}_{(1)}}\}, \tilde{r}_{Y,(1)}), \tilde{v}_1) \xrightarrow{a_1, t_1} \ldots \\ \xrightarrow{a_{n-1}, t_{n-1}} (\{(l_{j_{(n)}}, \tilde{C}_{j_{(n)}}, \tilde{b}_{j_{(n)}})_{j_{(n)} \in \tilde{J}_{(n)}}\}, \tilde{r}_{Y,(n)}), \tilde{v}_n) \\ \forall i \in [\![1, n]\!],\ \exists \sigma \in Y^Y \text{ tel que : } \tilde{v}_i \circ \sigma = v_i,\ \tilde{r}_i \circ \sigma = r_i,\ \tilde{J}_{(i)} \subseteq J_{(i)},\ \forall j \in \tilde{J}_{(i)}\ \tilde{C}_j \circ \sigma = C_j \wedge \\ b_j \Rightarrow ((j \in \tilde{J}_{(i)}) \wedge \tilde{b}_j). \end{array} \right)$$

On initialise la récurrence à $n = 0$. Les deux parties de longueur 0 sont identiques, P_0 est donc vraie.

Soit $n \in \mathbb{N}$, supposons P_n vraie et montrons l'hérédité de cette propriété. Par l'hypothèse de récurrence et le Lemme 1 on a : $\forall j_{(n)} \in \tilde{J}_{(n)},\ [\tilde{C}_{j_{(n)}} \cap \tilde{r}_{Y,(n)}]_{|X} \subseteq [C_{j_{(n)}} \cap r_{Y,(n)}]_{|X}$. La transition tirée $(\{(l_{j_{(n)}}, \tilde{C}_{j_{(n)}}, \tilde{b}_{j_{(n)}})_{j_{(n)} \in \tilde{J}_{(n)}}\}, \tilde{r}_{Y,(n)}), \tilde{v}_n) \xrightarrow{a_n, t_n} (\{(l_{j_{(n+1)}}, \tilde{C}_{j_{(n+1)}}, \tilde{b}_{j_{(n+1)}})_{j_{(n+1)} \in \tilde{J}_{(n+1)}}\}, \tilde{r}_{Y,(n+1)}), \tilde{v}_{n+1})$ correspond à un choix $(a_n, \tilde{R})$ avec $\tilde{R}$ successeur temporel de $\tilde{r}_{Y,(n)}$. On a alors que $R = \tilde{R} \circ \sigma$ est le successeur temporel de $r_{Y,(n)}$ tel que (a_n, R) soit le choix de Spoiler correspondant à la transition $(\{(l_{j_{(n)}}, C_{j_{(n)}}, b_{j_{(n)}})_{j_{(n)} \in J_{(n)}}\}, r_{Y,(n)}), v_n) \xrightarrow{a_n, t_n} (\{(l_{j_{(n+1)}}, C_{j_{(n+1)}}, b_{j_{(n+1)}})_{j_{(n+1)} \in J_{(n+1)}}\}, r_{Y,(n+1)}), v_{n+1})$ et $\forall j_{(n)} \in \tilde{J}_{(n)},\ [\tilde{C}_{j_{(n)}} \cap \tilde{r}_{Y,(n)}]_{|X} \subseteq [C_{j_{(n)}} \cap R]_{|X}$.

Par inclusion des gardes induites, les successeurs de la partie $(S_S, \tilde{S})$ correspondent bien à des configurations de la partie (S_S, S) tels que ce soient les mêmes localités et que les marqueurs de la partie (S_S, S) impliquent les autres. Par construction, il existe une unique valuation w de X qui correspond aux dernières transitions. Si un marqueur $b_{j_{(n+1)}} = \top$ alors $[\tilde{C}_{j_{(n)}} \cap \tilde{r}_{Y,(n)}]_{|X} \subseteq [C_{j_{(n)}} \cap r_{Y,(n)}]_{|X} \Rightarrow ((j_{(n)} \in J_{(n)}) \vee ([\tilde{C}_{j_{(n)}} \cap \tilde{r}_{Y,(n)}]_{|X} = \emptyset))$, or $w \in [\tilde{C}_{j_{(n)}} \cap \tilde{r}_{Y,(n)}]_{|X}$. Il nous suffit maintenant de montrer l'existence de σ pour le dernier état, or n'importe lequel des mappings de l'état courant de $Aut(S)$ convient.

On a montré que tout chemin blanc dans la partie (S_S, S) correspond à un chemin blanc dans la partie $(S_S, \tilde{S})$. En effet, si le chemin forme une boucle dans $(S_S, \tilde{S})$, elle est blanche. Plus précisément, si une stratégie S_S de Spoiler correspond à une partie de longueur n $(S_S, \tilde{S})$ qui repasse par le même état de Determinisator, on peut trouver une famille finie de stratégies de Spoiler qui correspondent à des parties finie ne repassant pas par le même état de Determinisator et couvrant la partie $(S_S, \tilde{S})$. □