THÈSE

présentée à l'École Normale Supérieure de Cachan

pour obtenir le grade de

Docteur de l'École Normale Supérieure de Cachan

par : Fabrice CHEVALIER

Spécialité : INFORMATIQUE

Logiques pour les systèmes temporisés : contrôle et expressivité

Soutenue le 25 juin 2007, devant un jury composé de :

– Patricia BOUYER	directrice de thèse
– Véronique BRUYÈRE	rapporteuse
– Franck CASSEZ	examinateur
– Paul GASTIN	directeur de thèse
– Joël OUAKNINE	rapporteur
– Wolfgang THOMAS	examinateur
– Pascal WEIL	président du jury

Résumé

Ce travail s'inscrit dans le cadre de la vérification formelle de programmes : il s'agit de vérifier si le modèle d'un système vérifie le modèle d'une propriété. Certains systèmes sont placés dans un environnement extérieur, il s'agit alors de guider ou contrôler le système pour qu'il satisfasse la propriété. Dans le but de modéliser des systèmes et propriétés relatives à l'écoulement du temps, nous nous intéressons aux systèmes hybrides et aux logiques temporisées.

Nous établissons dans un premier temps que le problème de contrôle pour les systèmes hybrides o-minimaux est décidable. Dans le but de modéliser des propriétés d'optimisation, nous introduisons alors une extension avec coûts des systèmes hybrides o-minimaux. Nous montrons que ce modèle est expressif mais analysable puisque le model-checking ainsi que le contrôle optimal sont décidables, alors que les problèmes équivalents sont indécidables dans le cadre des automates temporisés.

Nous étudions également les logiques temporisées, notamment les logiques MTL et TPTL qui sont des extensions de LTL avec des contraintes quantitatives. Ces logiques permettent par exemple d'exprimer des propriétés comme « une alarme se mettra à sonner moins de 10s après le début d'un incendie ». Nous prouvons alors une conjecture énoncée par Alur et Henzinger au début des années 1990 et établissons que la logique TPTL est strictement plus expressive que MTL. Dans le cadre du contrôle temporisé pour des spécifications MTL, nous identifions la frontière de décidabilité selon les ressources allouées au contrôleur.

Enfin nous introduisons une classe d'automates paramétrés par des opérateurs. Notre approche permet d'obtenir des résultats génériques d'expressivité entre automates, logiques du premier et du second ordre, et logiques temporelles. Ces résultats permettent de déduire différentes représentations pour MTL+Past, par une logique du premier ordre, ainsi que sous la forme d'automates, ce qui permet de prouver une propriété de stabilité à l'infini de cette logique.

Mots clefs : contrôle, systèmes hybrides, logiques temporisées, expressivité.

Abstract

This thesis deals with formal verification of programs, which consists in verifying that the model of a system satisfies the model of a property. Most embedded systems are *open* systems, *i.e.* they interact with an environment and must be guided or controlled to satisfy their specifications. For modelling systems and properties with timing constraints, we study hybrid systems and real-time logics.

We show that the control problem for o-minimal hybrid systems is decidable. To express optimization properties, we introduce a weighted extension of o-minimal hybrid systems. We show that this model is expressive though analysable, as model-checking and optimal control are decidable, whereas similar problems are undecidable for timed automata.

We study real-time logics as well, in particular MTL and TPTL which are extensions of LTL with quantitative constraints. These logics allow to express properties like 'an alarm will ring less than 10s after a fire has started'. We then prove a conjecture stated by Alur and

Henzinger at the beginning of the 90s and show that the logic TPTL is strictly more expressive than MTL. We also study timed control for MTL specifications, and identify the decidability border depending on the resources allowed to the controller.

Finally we introduce a class of automata parametrized by operators. Our approach yields generic expressivity results between automata, first and second-order logics, and temporal logics. These results enable to deduce different representations for MTL+Past, as a first-order logic, and by automata, from which we can deduce an ultimate stability property of this logic.

Keywords : Control, hybrid systems, real-time logics, expressivity.

Remerciements

« *Dans une thèse, la partie la plus difficile à écrire est l'introduction.* »
Adage erroné

Je tiens tout d'abord à remercier les membres de mon jury : Patricia Bouyer, Véronique Bruyère, Franck Cassez, Paul Gastin, Joël Ouaknine, Wolfgang Thomas et Pascal Weil. Je remercie en particulier Véronique et Joël qui ont accepté d'être rapporteurs de ma thèse. Je souhaite remercier chaleureusement Patricia pour ces trois années de recherche. Elle m'a aiguillé vers des problèmes intéressants et variés, et grâce à son enthousiasme les nombreuses heures passées devant le tableau blanc furent un réel plaisir. Je la remercie également pour ses conseils, son aide et sa disponibilité, souvent au-delà du raisonnable.

Je remercie Nicolas et Thomas, avec qui j'ai eu beaucoup de plaisir à travailler. I would like to thank Deepak and Pavithra, for our collaborations and for welcoming me in Bangalore, where I had a very pleasant stay.

Je tiens également à remercier les membres du LSV pour l'environnement de recherche et la bonne ambiance qui règne au sein du laboratoire. Je remercie François, Hubert, Jean et Paul pour avoir supervisé mes enseignements, ainsi que Claudine et Manuel pour leurs précieux conseils. Merci à Pierre-Alain et Arnaud pour nos partages pas toujours équitables du projet de programmation. Je remercie les membres de la salle Renaudeau, sorte de LSV à l'intérieur du LSV, en particulier le noyau dur : Sébastien, Pierre-Alain, Régis et Arnaud ; les nouveaux ne sont pas mal non plus : Rémi, Najla, Florent, Jean-Loup et Jules.

On ne peut évoquer le LSV sans oublier les activités extra-professionnelles qui y sont organisées. Merci donc aux joueurs de *Wanted !*, et parmis les plus assidus Thierry, Houda, Marie, Pascal, Steve, Nathalie et Patricia[1]. Merci également aux joueurs de squash du midi : Nicolas, Nathalie, Benedikt, Ralf, Pascal, Patricia, Arnaud, Fanny, Thomas et Laurent. Je remercie aussi les nombreux joueurs de Xblast, puissent-ils transmettre cette tradition durant de nombreuses années !

Après toutes ces activités il reste encore des personnes à remercier : Nathalie bien sûr, mais elle est déjà bien assez citée, Lionel pour les soirées discussion/télé, Agnès et François pour les invitations et les jeux ainsi que Myriam et David pour nos aventures bahamo-bagnolaises.

Je remercie ma belle-famille pour son accueil toujours chaleureux à Strasbourg : Mireille, Pierre, Julien et Marie. Je remercie ma famille pour sa présence et son soutien depuis toujours. Enfin je remercie celle sans qui toutes ces années n'auraient pas eu la même couleur : Frédérique.

[1] Il s'agit bien de ma directrice de thèse.

Table des matières

Chapitre 1

Introduction

Les enjeux de la vérification

L'informatique est devenue en quelques dizaines d'années un outil incontournable dans de nombreux domaines tels que les bases de données, les transports, les télécommunications, *etc.* Nombre de ces applications ont un rôle critique, et une défaillance peut avoir de lourdes conséquences humaines ou financières. Ainsi, de 1985 à 1987, des patients traités par l'appareil médical Therac-25 reçurent des doses massives de radiations, pouvant aller jusqu'à cent fois le traitement préconisé, ce qui provoqua la mort d'au moins cinq d'entre eux. Les causes de ces accidents sont largement imputables à la composante logicielle de l'appareil : procédure de certification incomplète, réutilisation incorrecte de code, programmation en langage de bas-niveau (assembleur). D'autres exemples de tels dysfonctionnements dûs à des erreurs logicielles incluent l'échec de la première fusée Ariane 5 en 1996, ou la cotation incorrecte de l'action de Google en avril 2006 (le prix de l'action est passé de 388 $ à 38$ en quelques minutes, toutes les transactions durant près d'une heure ont dû être annulées). Il apparaît donc indispensable de développer des méthodes systématiques d'analyse et de vérification des systèmes informatiques.

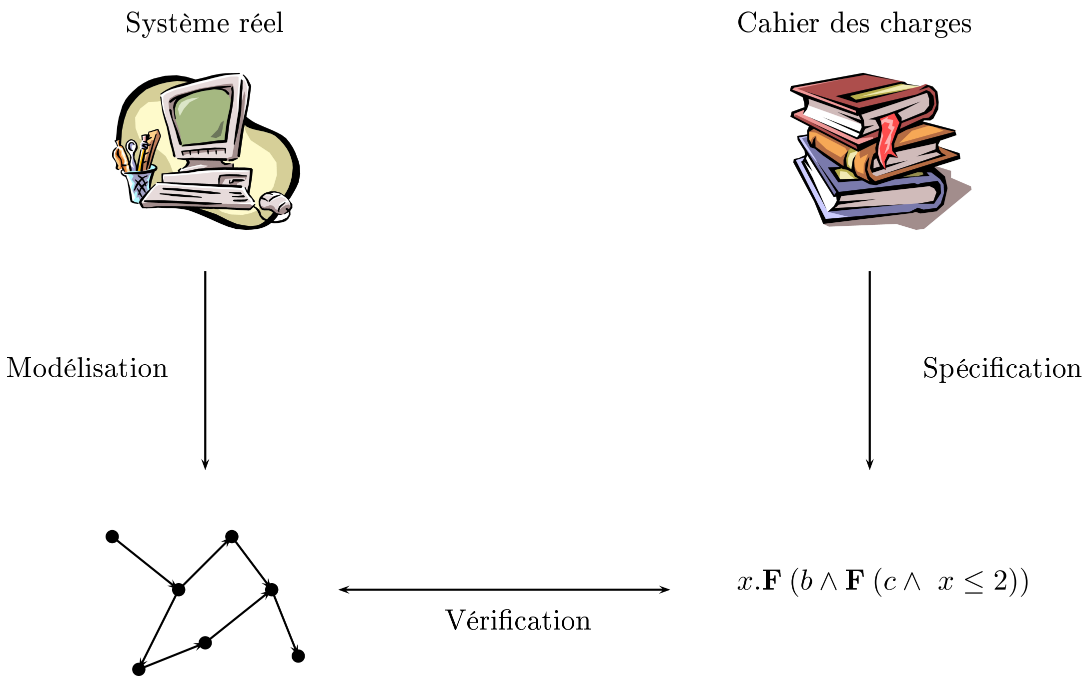
Les méthodes formelles

Une des raisons de la présence d'erreurs dans les logiciels réside dans le fait que les besoins informatiques sont typiquement exprimés sous forme de spécifications ou de cahier des charges ; ces documents, souvent rédigés en langage naturel, peuvent donner lieu à des erreurs d'interprétation lors de leur implémentation. Les *méthodes formelles* visent à établir un cadre mathématique où exprimer les ressources et besoins informatiques.

La vérification formelle se déroule généralement en trois phases :

- une phase de *modélisation* où le système réel est décrit par un modèle mathématique censé reproduire aussi fidèlement que possible son fonctionnement,
- une phase de *spécification* qui consiste à exprimer les propriétés devant être vérifiées par le système ; ces propriétés peuvent être décrites dans un formalisme logique,
- une phase de *vérification*, qui s'assure que le modèle vérifie les propriétés requises.

Nous pouvons distinguer trois principales familles de méthodes formelles, qui se différencient principalement par la manière dont la phase de vérification est réalisée :



- *le test* consiste à générer des scénarios de fonctionnement du système, et à vérifier que l'exécution du système est correcte lors de cette simulation. Différentes techniques existent pour générer un ensemble de tests « intéressants ».
- *la démonstration automatique* permet de prouver formellement que le modèle satisfait la spécification en réalisant une preuve mathématique. Il existe des assistants de preuve, qui permettent de générer automatiquement de telles preuves à partir d'indications fournies par l'utilisateur.
- *le model-checking* (ou vérification de modèle) permet de vérifier automatiquement que le modèle du système vérifie le modèle de la spécification.

Nous nous plaçons dans le cadre du *model-checking*, cette approche combine certains avantages du test et de la démonstration automatique : elle est automatisée et ne nécessite pas de travail de la part de l'utilisateur une fois que le modèle et la spécification sont établis ; elle est également exhaustive, la vérification ne se base pas sur quelques scénarios fixés qui peuvent ne pas prendre en compte tous les cas possibles, mais fournit une preuve formelle que le modèle satisfait la spécification. En contrepartie le cadre du *model-checking* est plus restrictif dans le sens où il pourra s'appliquer à une classe moins grande de systèmes. Le but sera donc de trouver des classes de systèmes et de spécifications les plus larges (ou expressives) possibles pour lesquelles le *model-checking* est décidable, c'est-à-dire qu'il existe un algorithme pouvant déterminer si le modèle satisfait la spécification.

Nous étudierons également un problème qui étend le cadre du *model-checking* : le problème de contrôle. Certains systèmes sont dits *ouverts* car ils sont placés dans un environnement extérieur ; le système ne peut décider du comportement de l'environnement et doit réagir en fonction des actions de ce dernier. Il faut alors guider (ou contrôler) le système de façon à ce qu'il satisfasse la spécification, quoi que fasse l'environnement ; cela devient un problème de *synthèse de systèmes* où le but est de construire un système qui permet de contrôler un autre système pour satisfaire une spécification.

Systèmes hybrides et logiques temporisées

Nous étudions plus particulièrement les *systèmes hybrides* [ACH+95] qui sont des systèmes ayant différents modes discrets, et équipés de variables soumises à des équations différentielles. Le formalisme des systèmes hybrides est très expressif mais il ne peut être utilisé directement pour le *model-checking* car il est indécidable. Il possède néanmoins de nombreuses sous-classes décidables comme les automates hybrides rectangulaires initialisés [Hen96], les automates temporisés [AD90, AD94], ou les systèmes hybrides o-minimaux [LPS00]. Un automate temporisé est un système hybride où toutes les variables évoluent à la même vitesse (celle du temps universel) et peuvent être comparées à des constantes ainsi qu'être remises à zéro. Nous nous intéresserons aussi aux systèmes hybrides o-minimaux : ces systèmes ont une dynamique continue très riche, mais un fonctionnement discret assez simple, puisque toutes les variables doivent être réinitialisées lors de chaque changement de mode. Nous étudierons en particulier les problèmes de *model-checking* et de contrôle pour les automates temporisés et les systèmes hybrides o-minimaux.

Nous avons décrit les modèles auxquels nous nous intéresserons, nous décrivons maintenant les types de spécifications que nous considérerons : les logiques temporisées. Les logiques temporelles sont largement utilisées dans le contexte de la vérification [Pnu77] ; en particulier, la logique temporelle de temps linéaire (LTL) permet d'exprimer des propriétés sur l'exécution d'un système comme *« toute apparition d'un problème déclenchera l'alarme dans le futur »*. Au début des années 90, des contraintes « temps-réel » ont été ajoutées aux logiques temporelles [AH89, Koy90] ; ces logiques temporisées permettent d'exprimer des propriétés quantitatives sur le temps comme *« toute apparition d'un problème déclenchera l'alarme en moins de cinq minutes »*. Nous considérerons différentes logiques temporisées pouvant exprimer ce type de propriétés et étudierons leur pouvoir d'expression et leur représentation par des automates.

Contributions de cette thèse

La thèse est divisée en trois parties, nous en détaillons les contributions partie par partie.

Partie I

Dans la première partie, nous nous intéressons au contrôle et au *model-checking* des systèmes hybrides, en particulier des systèmes o-minimaux.

Nous nous donnons une propriété à vérifier φ et un système $\mathcal{S}$ plongé dans un environnement ; ainsi certaines actions sont contrôlables et d'autres non. Le problème de contrôle consiste à déterminer s'il est possible de construire un contrôleur $\mathcal{C}$, de telle façon que le système contrôlé $\mathcal{S} \parallel \mathcal{C}$ satisfasse la propriété φ, quoi que fasse l'environnement.

De nombreux résultats ont déjà été obtenus dans ce cadre, comme l'indécidabilité du contrôle pour les automates hybrides [HHM99], ou l'existence de semi-algorithmes pour ce problème [dAHM01]. Dans le chapitre 3, nous étudions le problème de contrôle pour les systèmes o-minimaux, et considérons deux cadres, celui d'une observation totale de la dynamique par le contrôleur, et celui d'une observation partielle de la dynamique. Dans le cadre de l'observation totale, une technique usuelle pour le contrôle de systèmes infinis est l'utilisation de la bisimulation à temps abstrait [AHLP00], en effet deux états bisimilaires sont soit tous deux gagnants, soit tous deux perdants pour le problème de contrôle. Nous montrons que dans le cadre des jeux hybrides, la bisimulation à temps abstrait n'est plus un bon outil pour étudier le problème de contrôle : nous exhibons un système qui contient deux états bisimilaires mais dont l'un est gagnant et l'autre perdant. Nous proposons donc une autre approche : nous utilisons une technique d'encodage des trajectoires par des mots introduite dans [BMRT04, BM05]. Nous montrons que pour un système o-minimal dont la structure est décidable, le problème de contrôle est décidable ; nous donnons un algorithme de point-fixe pour calculer l'ensemble des états gagnants, et prouvons que des stratégies gagnantes peuvent être calculées. Nous traitons ensuite le cadre du contrôle o-minimal sous observation partielle, nous proposons un nouveau codage par mots, appelé codage par supermots, qui permet de traduire de façon correcte l'hypothèse d'observation partielle et d'obtenir des résultats de décidabilité et de calculabilité similaires à ceux obtenus sous observation totale.

Dans le chapitre 4, nous étudions une classe de systèmes o-minimaux étendus avec des fonctions de coût, ce formalisme permet de modéliser naturellement des problèmes comme la gestion de ressources. Les systèmes pondérés (*i.e.* avec une fonction de coût) ont été largement étudiés dans le cadre des automates temporisés donnant lieu aux automates temporisés pondérés [ALP01, BFH$^+$01]. Dans ce modèle, une nouvelle variable de coût est ajoutée au système, mais celle-ci est une variable *d'observation*, c'est-à-dire qu'elle n'influe pas sur le comportement du système. Ce modèle est intéressant pour vérifier des propriétés quantitatives des systèmes temporisés, ce qui a été confirmé par la décidabilité du problème d'atteignabilité optimale (trouver le meilleur moyen en terme de coût pour atteindre un état) [ALP01, BFH$^+$01], le développement d'Uppaal Cora [Cor06], et la calculabilité du coût optimal moyen (trouver le meilleur moyen pour le système d'assurer un coût par unité de temps le plus bas possible) [BBL04]. Cependant, des propriétés plus complexes comme le contrôle optimal (trouver le coût le plus bas pour atteindre un état donné, quoi que fasse l'environnement), ou le *model-checking* de WCTL (une logique étendant CTL avec des modalités sur les coûts) ont été montrés indécidable pour les automates temporisés pondérés avec trois horloges ou plus [BBR04, BBR05, BBM06]. Bien que ces problèmes aient été montrés décidable pour les automates temporisés pondérés à une horloge [BLMR06, BLM07], ces résultats de décidabilité sont assez restrictifs.

Nous proposons une extension naturelle des systèmes o-minimaux enrichis d'une fonction de coût positive qui croît avec le temps et qui peut être utilisée comme critère d'optimisation, comme dans le cadre des automates temporisés pondérés. Il est intéressant de remarquer que le système obtenu n'est pas o-minimal car le coût n'est pas remis à zéro quand une transition discrète est prise. Nous devons donc développer des techniques spécifiques pour ce type de systèmes. Nous montrons que le contrôle optimal et le *model-checking* sont décidables pour les systèmes o-minimaux pondérés. Étant donné que ces problèmes sont indécidables dans le cadre des automates temporisés pondérés, ces résultats attestent que les systèmes o-minimaux sont une classe analysable et expressive de systèmes.

Ces travaux ont fait l'objet de deux publications dans des actes de conférences internationales [BBC06b], [BBC07].

Partie II

Dans cette partie, nous présentons les logiques temporisées que nous étudierons par la suite et nous considérons le problème de contrôle pour des spécifications MTL.

Nous considérons des extensions temporisées de LTL ; la logique MTL étend LTL en ajoutant des contraintes sur les opérateurs temporels comme dans la formule suivante :

$$\mathbf{G}\,(\texttt{problème} \Rightarrow \mathbf{F}_{\leq 5}\,\texttt{alarme})$$

qui exprime que tout problème est suivi, moins de 5 unités de temps plus tard, par une alarme. La logique TPTL utilise des horloges de formule pour exprimer les contraintes temporelles ; une formule de TPTL peut remettre une horloge à zéro, et comparer plus tard la valeur de cette horloge à une constante, la propriété précédente s'exprime par exemple par la formule de TPTL :

$$\mathbf{G}\,(\texttt{problème} \Rightarrow x.\mathbf{F}\,(\texttt{alarme} \wedge x \leq 5))$$

où $x.\varphi$ signifie que l'horloge x est remise à zéro à la position courante, avant d'évaluer φ.

Nous distinguons deux sémantiques, selon que les formules sont évaluées sur des mots temporisés (c'est-à-dire sur une séquence discrète d'observations du système, c'est la *sémantique d'actions*) ou sur les séquences temporisées (c'est-à-dire une observation continue du système, c'est la *sémantique continue*). Les problèmes de satisfaisabilité et de *model-checking* de TPTL ont été prouvés indécidables [AH89] pour les deux sémantiques. Cette preuve d'indécidabilité s'applique à MTL pour la sémantique continue. Cependant les problèmes de satisfaisabilité et de *model-checking* de MTL sont décidables pour la sémantique d'actions sur les mots finis [OW05], même s'ils sont indécidables sur les mots infinis [OW06a].

La preuve d'indécidabilité originale de [AH89] utilise la réduction du problème d'accessibilité d'une machine à deux compteurs ; son principe est naturel et intuitif mais sa formalisation est assez complexe [DP07]. Nous proposons dans le chapitre 5 une méthode différente à partir d'une réduction de l'accessibilité des automates communicants. Cette technique a été utilisée dans [OW05] pour établir la complexité non primitive-récursive de MTL pour la sémantique d'actions. Notre approche permet de montrer que les problèmes de satisfaisabilité et de *model-checking* sont indécidables pour différentes logiques temporisées sur les mots finis ou infinis : TPTL et MTL+Past pour les deux sémantiques, et MTL pour la sémantique continue.

Après avoir introduit ces différentes logiques temporisées et montré l'indécidabilité de certaines, nous considérons le problème de contrôle temporisé. Il s'agit, étant donné un système modélisé par un automate temporisé $\mathcal{P}$ plongé dans un environnement, de déterminer s'il est possible de contrôler $\mathcal{P}$ pour satisfaire une propriété φ, quoi que fasse l'environnement. Les problèmes de contrôle temporisé pour des propriétés d'accessibilité et de sûreté sont décidables [MPS95, AMPS98, HK99]. Pour des propriétés plus complexes, comme celles données par un automate temporisé de Büchi, le problème de contrôle temporisé devient indécidable [DM02]. Cela a amené à introduire des restrictions sur le pouvoir du contrôleur (en fixant son nombre d'horloges et les constantes qu'il utilise), ce problème de contrôle, dit *à ressources fixées*, a été montré décidable [DM02], et l'est même dans un cadre d'observation partielle [BDMP03].

Nous étudions le contrôle temporisé où la spécification est donnée par une formule appartenant aux logiques temporisées exposées au préalable. Comme ce problème est plus difficile que celui de la satisfaisabilité, nous nous restreignons à des logiques dont le problème de satisfaisabilité est décidable. Nous nous plaçons dans le cadre de la sémantique d'actions et considérons les logiques MTL sur les mots finis et Safety-MTL sur les mots infinis (Safety-MTL est une restriction de MTL qui impose intuitivement une borne supérieure finie sur les intervalles contraignant les modalités temporelles). En utilisant l'interaction entre l'environnement et le contrôleur, nous montrons que le problème de contrôle temporisé pour des spécifications MTL sur les mots finis et Safety-MTL sur les mots infinis est indécidable. Nous considérons également le problème de contrôle à ressources fixées et montrons qu'il est décidable pour des spécifications MTL sur les mots finis et Safety-MTL sur les mots infinis. Les méthodes que nous développerons sont très différentes de celles utilisées auparavant pour le contrôle à ressources fixées [DM02, BDMP03] et utilisent des techniques d'ordre partiel pour les configurations temporisées [AN01, OW04, OW05].

Ces travaux ont fait l'objet d'une publication dans des actes de conférences internationales [BBC06a].

Partie III

Dans cette partie, nous donnons des résultats d'expressivité pour MTL et TPTL et introduisons un nouveau formalisme d'automates qui permet d'obtenir de nouvelles caractérisations pour les logiques temporisées, en particulier MTL.

Comme nous l'avons vu dans la partie précédente, toute formule de MTL peut être traduite en une formule équivalente de TPTL. Alur et Henzinger ont conjecturé que TPTL serait strictement plus expressive que MTL [AH92b, AH93, Hen98], en particulier qu'il n'existerait pas de formule de MTL équivalente à la formule de TPTL

$$\mathbf{G}\,(\texttt{problème} \Rightarrow x.\mathbf{F}\,(\texttt{alarme} \wedge \mathbf{F}\,(\texttt{initialisation} \wedge x \leq 5))) \tag{1.1}$$

sans toutefois apporter de preuve de ce résultat. Nous résolvons ce problème pour les deux sémantiques (la sémantique d'actions et la sémantique continue) et montrons que :

- la conjecture est vraie pour les deux sémantiques,
- pour la sémantique d'actions, la formule (1.1) est un bon témoin, c'est-à-dire qu'elle ne peut effectivement pas être exprimée dans MTL,
- pour la sémantique continue, la formule (1.1) *peut* être exprimée dans MTL, mais nous donnons une autre formule de TPTL qui ne peut être exprimée dans MTL, confirmant ainsi la conjecture.

Nos techniques nous permettent également d'obtenir d'autres résultats : pour les deux sémantiques, ajouter des modalités du passé à MTL et MITL augmente strictement leur expressivité (MITL [AFH96] est un fragment de MTL où les égalités sont interdites dans les contraintes) ; MTL est strictement plus expressive pour la sémantique continue que pour la sémantique d'actions ; pour la sémantique continue, MITL ne peut exprimer la formule (1.1), ce qui est contre-intuitif car la formule (1.1) n'utilise pas de contrainte ponctuelle. Nous prouvons enfin que pour la sémantique continue, le fragment de TPTL où la seule modalité **F** est autorisée peut être traduit dans MTL. Cela généralise le fait que la formule (1.1) peut être exprimée dans MTL pour la sémantique continue.

Dans le chapitre 8, nous introduisons un nouveau modèle d'automates temporisé, les automates *input-determined*. Ces automates sont une généralisation de plusieurs modèles introduits depuis les années 90 [AFH96, RS97, DM05]. Ils n'utilisent pas d'horloges explicites, et ne sont pas directement comparables aux automates temporisés, mais ont de fortes propriétés logiques telles que la clôture par complément. Les gardes d'un automate *input-determined* imposent des contraintes uniquement sur le mot d'entrée, leur valeur de vérité ne dépend pas du chemin suivi dans l'automate. Un exemple d'opérateur *input-determined* est l'opérateur $\lhd_a$ de [AFH96] qui mesure le temps depuis la dernière occurrence d'une action a. De même l'opérateur $\Diamond_a$ [DT04, DM05], qui rappelle l'opérateur $\mathbf{F}$ de MTL mesure le temps jusqu'aux prochaines occurrences de l'action a. Pour la sémantique d'actions, l'article [DT04] donne un cadre général pour l'étude d'automates basés sur des opérateurs *input-determined*, qui permet de déduire des caractérisations par logique monadique du second ordre, des résultats d'expressivité, des propriétés de déterminisation et de clôture.

Nous construisons ici un cadre similaire pour la sémantique continue. Nous définissons une version continue des automates *input-determined* paramétrée par un ensemble d'opérateurs. Nous montrons que cette classe d'automates est déterminisable et close pour les opérations booléennes. Tout comme les automates finis, les automates *input-determined* continus admettent une caractérisation par une logique monadique du second ordre ; de plus le fragment du premier ordre de cette logique est expressivement équivalente à une extension *input-determined* naturelle de LTL.

Nous étendons ensuite ces résultats à des automates *input-determined* récursifs. Comme application nous montrons que la logique MTL+Past, qui étend MTL avec des modalités du passé, est expressivement équivalent à une logique du premier ordre associée. La logique MTL+Past peut être vue comme la logique de temps linéaire associée à l'opérateur $\Diamond$ (ainsi qu'à sa version passé $\diamondminus$), et correspond au fragment du premier ordre de la logique monadique du second ordre caractérisant les automates *input-determined* récursifs.

Comme dit ci-dessus, MTL+Past est expressivement équivalente à une logique du premier ordre utilisant des opérateurs *input-determined*. Dans le chapitre 9, nous donnons une représentation de MTL+Past en terme d'automates, établissant une caractérisation similaire à celle de la représentation par automates finis sans compteurs de la logique du premier ordre [Kam68, MP71]. Nous identifions une classe d'automates *input-determined* sans compteurs (paramétrée par un ensemble d'opérateurs) qui caractérise précisément la logique temporelle basée sur ces opérateurs.

Ces résultats donnent une caractérisation en termes d'automates de MTL+Past. Parmi d'autres applications, une telle caractérisation peut être utile pour obtenir des résultats d'expressivité pour certaines logiques. Nous utilisons enfin cette caractérisation pour prouver une propriété des langages temporisés définissables dans MTL+Past : nous montrons que MTL+Past est *stable à l'infini*, c'est-à-dire qu'étant donnée une séquence périodique de mots temporisés uv^iw, la valeur de vérité d'une formule de MTL+Past sur ces modèles finit par se stabiliser sur « vrai » ou « faux ». Ce résultat obtenu grâce à notre caractérisation de MTL+Past en terme d'automates, est une version plus forte d'un résultat de stabilité similaire [PD06], car il est valide même avec les modalités du passé.

Ces travaux ont fait l'objet de deux publications dans des actes de conférences internationales [BCM05], [CDP06].

Ce document de thèse ne regroupe qu'une partie des travaux réalisés ces dernières années. D'autres contributions ont été publiées notamment dans [BCD05, BCKT05, BC06, BC07].

Chapitre 2

Préliminaires

Dans ce chapitre, nous introduisons certaines notions fondamentales que nous utiliserons par la suite. Nous définissons et donnons quelques propriétés des automates finis et des automates finis avec invariants ; pour une étude plus complète des automates finis, consulter par exemple [Sak03]. Nous présentons les deux modèles temporisés que nous utiliserons : les mots temporisés et les séquences temporisées. Nous présentons enfin la sémantique des automates temporisés [AD90, AD94].

2.1 Automates finis

Alphabets et automates finis

Nous notons $\mathbb{N}$ l'ensemble des entier naturels. Un *alphabet* Σ est un ensemble fini de symboles. Nous noterons Σ^* l'ensemble des suites finies d'éléments de Σ, une telle suite sera également appelée *mot* sur Σ. La longueur d'un mot $w = w_0 \cdots w_n \in \Sigma^*$, notée $|w|$ est $n+1$. Le mot vide, noté ε, est de longueur 0. Un *langage* sur Σ est un sous-ensemble de Σ^*.

Définition 2.1.1 *Un* automate fini *est un n-uplet $\mathcal{A} = (\Sigma, Q, I, \delta, F)$ où Σ est un alphabet d'actions, Q un ensemble d'états, $I \subseteq Q$ un ensemble d'états initiaux, $\delta \subseteq Q \times \Sigma \times Q$ une relation de transition et $F \subseteq Q$ un ensemble d'états finaux. Un tel automate est dit* automate fini sur Σ.

Une *exécution* de $\mathcal{A}$ sur un mot $w = w_0 \cdots w_n \in \Sigma^*$ à partir d'un état $q_0 \in Q$ est une suite $\rho = q_0 \xrightarrow{w_0} q_1 \xrightarrow{w_1} \cdots \xrightarrow{w_n} q_{n+1}$ telle que pour tout $0 \leq i \leq n$, $(q_i, w_i, q_{i+1}) \in \delta$. Une telle exécution est acceptante si $q_{n+1} \in F$.

Nous définissons la *fonction de transition généralisée* $\delta^* : Q \times \Sigma^* \to 2^Q$ inductivement par $\delta^*(q, \varepsilon) = \{q\}$ et $\delta^*(q, a \cdot w) = \{q' \in Q \mid \exists q_1 \in Q,\ q_1 \in \delta(q, a) \text{ et } q' \in \delta^*(q_1, w)\}$ pour $a \in \Sigma$ et $w \in \Sigma^*$.

Un état $q \in Q$ est dit *accessible* s'il existe une exécution $\rho = q_0 \xrightarrow{w_0} q_1 \xrightarrow{w_1} \cdots \xrightarrow{w_n} q_{n+1}$ à partir de $q_0 \in I$ telle que $q_{n+1} = q$. Un état $q \in Q$ est dit *co-accessible* s'il existe une exécution $\rho = q_0 \xrightarrow{w_0} q_1 \xrightarrow{w_1} \cdots \xrightarrow{w_n} q_{n+1}$ à partir de $q_0 = q$ telle que $q_{n+1} \in F$. Un automate est dit *accessible* (resp. *co-accessible*) si tous ses états sont accessibles (resp. co-accessibles).

Le langage symbolique[1] accepté par $\mathcal{A}$, noté $L^*_{symb}(\mathcal{A})$, est l'ensemble des mots sur Σ sur lesquels $\mathcal{A}$ a une exécution acceptante à partir d'un état initial. Un langage accepté par un automate fini est dit *régulier*.

Un automate fini $\mathcal{A} = (\Sigma, Q, I, \delta, F)$ est dit *déterministe* si I est un singleton et pour tout $q \in Q$, pour tout $a \in \Sigma$, il y a au plus un état $q' \in Q$ tel que $(q, a, q') \in \delta$. La relation de transition d'un automate fini déterministe peut être vue comme une application partielle de $Q \times \Sigma$ dans Q. Un automate fini $\mathcal{A} = (\Sigma, Q, i, \delta, F)$ est dit *complet* si pour tout $q \in Q, a \in \Sigma$, il y a au moins un état $q' \in Q$ tel que $(q, a, q') \in \delta$.

Soit Σ un alphabet, tout langage régulier L sur Σ peut-être reconnu par un unique (à isomorphisme près) automate fini déterministe complet accessible avec un nombre minimal d'états, qui sera noté $\mathcal{A}_L$.

Automates finis avec invariants

Dans la partie III, nous utiliserons une variante d'automates finis avec des invariants. Un *alphabet partitionné* est un couple d'alphabets (Σ_1, Σ_2) où $\Sigma_1 \cap \Sigma_2 = \emptyset$. Pour des raisons de lisibilité nous noterons parfois un tel alphabet partitionné $\Sigma = \Sigma_1 \cup \Sigma_2$.

Définition 2.1.2 *Un* automate fini avec invariants *est un n-uplet $\mathcal{A} = (\Sigma, Q, i, \delta, F, l)$, où $\Sigma = \Sigma_1 \cup \Sigma_2$ est un alphabet partitionné, Q un ensemble d'états, $i \in Q$ un état initial, $\delta \subseteq Q \times \Sigma_1 \times Q$ une relation de transition, $F \subseteq Q$ un ensemble d'états finaux et $l : Q \longrightarrow \Sigma_2$ une fonction d'étiquetage des états. Un tel automate est dit* automate fini avec invariants sur *Σ.*

Un automate fini avec invariants accepte des mots de $\Sigma_1 \cdot (\Sigma_2 \cdot \Sigma_1)^*$. Une exécution de $\mathcal{A}$ sur un mot $w = w_0 \cdots w_{2n} \in \Sigma_1 \cdot (\Sigma_2 \cdot \Sigma_1)^*$ est une suite $\rho = q_0 \xrightarrow{w_0} q_1 \xrightarrow{w_1} \cdots \xrightarrow{w_{2n}} q_{n+1}$ telle que $q_0 = i$, pour tout $1 \leq j \leq n$, $l(q_j) = w_{2j-1}$ et pour tout $0 \leq j \leq n$, $(q_j, w_{2j}, q_{j+1}) \in \delta$. Une telle exécution est acceptante si $q_{n+1} \in F$. Le langage symbolique d'un automate fini avec invariants $\mathcal{A}$ est l'ensemble des mots de Σ^* sur lesquels $\mathcal{A}$ a une exécution acceptante.

Proposition 2.1.3 *Soit $\Sigma = \Sigma_1 \cup \Sigma_2$ un alphabet partitionné. Les langages acceptés par automates finis avec invariants sur Σ sont les langages réguliers inclus dans $\Sigma_1 \cdot (\Sigma_2 \cdot \Sigma_1)^*$.*

Preuve. Nous raisonnons par double implication.
Implication directe. Soit $\mathcal{A} = (\Sigma, Q, i, \delta, F, l)$ un automate fini avec invariants. Nous construisons un automate fini $\mathcal{A}'$ sur Σ comme suit : pour tout état $q \in Q$ nous ajoutons un état q' et une nouvelle transition $(q, l(q), q')$. Toutes les transitions sortantes de q sont transférées sur q'. Nous ajoutons un nouvel état initial t avec les mêmes transitions sortantes que i. Les états finaux de $\mathcal{A}'$ sont les mêmes que ceux de $\mathcal{A}$.
Implication réciproque. Soit $\mathcal{A} = (\Sigma, Q, i, \delta, F, l)$ un automate fini acceptant un sous-ensemble de $\Sigma_1 \cdot (\Sigma_2 \cdot \Sigma_1)^*$. Nous construisons un automate fini avec invariants reconnaissant le même langage comme suit : les états de $\mathcal{A}'$ sont les transitions de $\mathcal{A}$ qui sont dans $Q \times \Sigma_2 \times Q$. Il y a une transition d'un état (p, a, q) à un état (r, c, t) étiquetée $b \in \Sigma_1$ si (q, b, r) est une transition de $\mathcal{A}$. Les états finaux de $\mathcal{A}'$ sont ceux de la forme (p, a, q) où $p \in F$. Il y a un nouvel état initial i', qui a pour transitions sortantes les $(i', b, (p, c, q))$ telles que $(i, b, p) \in \delta$. Tout état (p, a, q) est étiqueté par a et i' est étiqueté de manière arbitraire. □

[1] Nous utilisons le terme « symbolique » ici car nous considérerons plus tard le langage temporisé pour certains automates.

Automates de Büchi

Soit Σ un alphabet. Nous noterons Σ^ω l'ensemble des suites infinies d'éléments de Σ, et Σ^∞ l'union de Σ^* et Σ^ω.

Un automate fini peut également être interprété pour reconnaître des mots infinis, dans ce cas nous parlons plutôt *d'automate de Büchi.*

Soit $\mathcal{A} = (\Sigma, Q, I, \delta, R)$ un automate de Büchi. Une *exécution* de $\mathcal{A}$ sur un mot $w = w_0 w_1 \cdots \in \Sigma^\omega$ est une suite $\rho = q_0 \xrightarrow{w_0} q_1 \xrightarrow{w_1} \cdots$ telle que $q_0 \in I$ et pour tout $0 \leq i \leq n$, $(q_i, w_i, q_{i+1}) \in \delta$. Une telle exécution est acceptante si $\{i \in \mathbb{N} \mid q_i \in R\}$ est infini.

Le langage symbolique accepté par un automate de Büchi $\mathcal{A}$, noté $L^\omega_{symb}(\mathcal{A})$, est l'ensemble des mots de Σ^ω sur lesquels $\mathcal{A}$ a une exécution acceptante.

2.2 Mots temporisés et séquences temporisées

Nous notons $\mathbb{Q}_{\geq 0}$ l'ensemble des nombres rationnels positifs ou nuls et $\mathbb{R}_{\geq 0}$ l'ensemble des nombres réels positifs ou nuls. L'ensemble de dates $\mathbb{T}$ représente l'un ou l'autre de ces ensembles.

Une *suite de dates* sur $\mathbb{T}$ est une suite finie ou infinie $\tau = (\tau_i)_{i \geq 0} \in \mathbb{T}^\infty$ d'éléments de $\mathbb{T}$ satisfaisant les conditions suivantes :

- *(monotonie)* la suite est croissante : $\forall i \geq 0,\ \tau_i \leq \tau_{i+1}$,
- *(progrès)* si τ est infinie alors $\{\tau_i \mid i \in \mathbb{N}\}$ est non-borné.

Un *mot temporisé* sur Σ est un couple $\sigma = (w, \tau)$ où $w \in \Sigma^\infty$ et τ est une suite de dates sur $\mathbb{T}$ de même longueur que w (deux suites infinies sont considérées comme étant de même longueur). Un mot temporisé peut également être vu comme un élément de $(\Sigma \times \mathbb{T})^\infty$. Nous notons $T\Sigma^*$ l'ensemble des mots temporisés finis sur Σ, et $T\Sigma^\omega$ l'ensemble des mots temporisés infinis sur Σ.

Exemple 2.2.1 *Soit $\sigma = (w, \tau)$ avec $w = (a, b, b, a)$ et $\tau = (0,7; 1,3; 2,87; 2,9)$. Alors σ est un mot temporisé fini sur l'alphabet $\{a, b\}$. Il peut également être vu comme la suite $(a; 0,7)(b; 1,3)(a; 2,87)(a; 2,9)$.*

Soit $\sigma = (w_0, t_0)(w_1, t_1) \cdots (w_n, t_n)$ un mot temporisé fini, la *durée* de σ est définie par $dur(\sigma) = t_n$. Étant donné un mot temporisé fini $\sigma = (w_0, t_0)(w_1, t_1) \cdots (w_n, t_n)$ et un mot temporisé fini ou infini $\sigma' = (w'_0, t'_0)(w'_1, t'_1) \cdots$, nous définissons la concaténation $\sigma \cdot \sigma'$ de manière usuelle par $(w_0, t_0)(w_1, t_1) \cdots (w_n, t_n)(w'_0, t_n + t'_0)(w'_1, t_n + t'_1) \cdots$

Un *intervalle* est un sous-ensemble convexe de $\mathbb{R}_{\geq 0}$. Nous notons $\mathcal{I}_\mathbb{R}$ l'ensemble des intervalles. Deux intervalles I et I' sont dits *adjacents* si $I \cap I' = \emptyset$ et $I \cup I'$ est un intervalle. Nous notons $\mathcal{I}_\mathbb{Q}$ l'ensemble des intervalles dont les bornes sont rationnelles. Nous utilisons la notation classique pour les intervalles, par exemple $]a, b] = \{x \in \mathbb{R}_{\geq 0} \mid a < x \leq b\}$. Nous utilisons les notations $\langle$ et $\rangle$ pour désigner des bornes pouvant être ouvertes ou fermées. Par exemple la notation $\langle a, b[$ désigne l'intervalle $]a, b[$ ou l'intervalle $[a, b[$.

Une *séquence temporisée* sur Σ est un couple $\kappa = (w, I)$ où $w \in (2^\Sigma)^\infty$ et $I = (I_i)_{i \geq 0}$ est une suite d'intervalles non-vides de même longueur que w satisfaisant les conditions suivantes :

- *(adjacence)* pour tout $i \in \mathbb{N}$, I_i et I_{i+1} sont adjacents,
- *(progrès)* si κ est infini, tout réel positif ou nul appartient à un intervalle I_i.

Une séquence temporisée peut également être vue comme un élément de $(2^\Sigma \times \mathcal{I}_\mathbb{R})^\infty$. Nous ne considérerons les séquences temporisées que pour l'ensemble de dates $\mathbb{T} = \mathbb{R}$.

Remarque 2.2.2 *Un mot temporisé peut être vu comme un cas particulier de séquence temporisée. Par exemple, le mot temporisé* $(a; 1, 8)(b; 2.3)$ *correspond à la séquence temporisée* $(\emptyset, [0; 1, 8[)(\{a\}, [1, 8; 1, 8])(\emptyset,]1, 8; 2, 3[)(\{b\}, [2, 3; 2, 3])(\emptyset,]2, 3; +\infty[)$.

2.3 Automates temporisés

Horloges

Nous considérons un ensemble fini X, que nous appellerons *ensemble d'horloges*. Une *valuation* sur X est une fonction $v : X \to \mathbb{T}$. Nous notons $\mathbf{0}$ la valuation qui envoie toute horloge $x \in X$ sur 0. Pour $t \in \mathbb{T}$, la valuation $v + t$ est définie par $(v + t)(x) = v(x) + t$ pour tout $x \in X$. Si Y est un sous-ensemble de X, la valuation $v[Y \leftarrow 0]$ est définie par : pour tout $x \in X$, $(v[Y \leftarrow 0])(x) = 0$ si $x \in Y$ et $(v[Y \leftarrow 0])(x) = v(x)$ sinon.

L'ensemble des *gardes* sur un ensemble d'horloges X, noté $\mathcal{G}(X)$ est défini par la grammaire suivante :

$$g ::= x \sim c \ \mid \ g \wedge g$$

où $\sim \in \{<, \leq, =, \geq, >\}$, $x \in X$, et $c \in \mathbb{Q}_{\geq 0}$.

La satisfaisabilité d'une garde g par une valuation v, notée $v \models g$, est définie par :

$$\begin{cases} v \models x \sim c & \text{si} \quad v(x) \sim c \\ v \models g_1 \wedge g_2 & \text{si} \quad v \models g_1 \text{ et } v \models g_2 \end{cases}$$

L'ensemble des valuations sur X qui satisfont une garde $g \in \mathcal{G}(X)$ est noté $[\![g]\!]_X$, ou simplement $[\![g]\!]$ si l'ensemble X est clair dans le contexte.

Alphabet symbolique

Soit Σ un alphabet et X un ensemble fini d'horloges. Un *alphabet symbolique* Γ basé sur (Σ, X) est un sous ensemble fini de $\mathcal{G}(X) \times \Sigma \times 2^X$.

Un mot symbolique $\gamma = (g_i, b_i, Y_i)_{i \geq 0} \in \Gamma^\infty$ génère un ensemble de mots temporisés, noté $tw(\gamma)$. Nous interprétons l'action symbolique (g, b, Y) par le fait qu'une action b peut se produire si la garde g est satisfaite, et que les horloges de Y sont remises à zéro après l'action. Formellement, soit $\sigma = (a_0, \tau_0)(a_1, \tau_1) \cdots$ un mot temporisé sur Σ, $\sigma \in tw(\gamma)$ si $|\sigma| = |\gamma|$ et s'il existe une suite de valuations $v_0, v_1, \cdots$ sur X telles que $v_0 = \mathbf{0}$, et pour tout $0 \leq i < |\gamma|$, $v_i + \tau_i - \tau_{i-1} \models g_i$ et $v_{i+1} = (v_i + \tau_i - \tau_{i-1})[Y_i \leftarrow 0]$ (en supposant $\tau_{-1} = 0$).

L'opérateur tw est étendu aux langages symboliques L sur Γ de manière usuelle par $tw(L) = \bigcup_{\gamma \in L} tw(\gamma)$.

Automates temporisés

Définition 2.3.1 *Un* automate temporisé *est un n-uplet* $\mathcal{A} = (\Sigma, X, Q, q_0, F, R, \delta)$ *où* Σ *est un alphabet d'actions,* X *un ensemble fini d'horloges,* Q *un ensemble fini d'états,* $q_0 \in Q$ *un état initial,* $F \subseteq Q$ *un ensemble d'états finaux,* $R \subseteq Q$ *un ensemble d'états répétés,* $\delta \subseteq Q \times (\mathcal{G}(X) \times \Sigma \times 2^X) \times Q$ *un ensemble fini de transitions.*

Nous omettons parfois F ou R quand ces ensembles sont vides.

Soit Γ un alphabet symbolique basé sur (Σ, X) tel que $\delta \subseteq Q \times \Gamma \times Q$, le n-uplet $(\Gamma, Q, q_0, \delta, F)$ peut être vu comme un automate fini et le n-uplet $(\Gamma, Q, q_0, \delta, R)$ comme un automate de Büchi.

En tant qu'automate fini, un automate temporisé $\mathcal{A}$ reconnaît un langage symbolique de mots finis sur Γ, noté $L^*_{symb}(\mathcal{A})$. Son langage temporisé est défini par $L^*(\mathcal{A}) = tw(L^*_{symb}(\mathcal{A}))$. En tant qu'automate de Büchi, un automate temporisé $\mathcal{A}$ reconnaît un langage symbolique de mots infinis sur Γ, noté $L^\omega_{symb}(\mathcal{A})$. Son langage temporisé de mots infinis est défini par $L^\omega(\mathcal{A}) = tw(L^\omega_{symb}(\mathcal{A}))$. Nous notons $L_{symb}(\mathcal{A}) = L^*_{symb}(\mathcal{A}) \cup L^\omega_{symb}(\mathcal{A})$ et $L(\mathcal{A}) = L^*(\mathcal{A}) \cup L^\omega(\mathcal{A})$.

Un automate temporisé $\mathcal{A} = (\Sigma, X, Q, q_0, F, \delta)$ est dit *déterministe* si pour toutes transitions distinctes $(q, (g, a, Y), q')$ et $(q, (g', a, Y'), q'')$ de δ, $[\![g]\!] \cap [\![g']\!] = \emptyset$.

Remarque 2.3.2 *La condition de déterminisme pour un automate temporisé est plus forte que le déterminisme de l'automate fini sous-jacent. En effet, les deux transitions* $(q, (g, a, Y), q')$ *et* $(q, (g, a, Y'), q'')$ *pour* $Y \neq Y'$ *donnent lieu à un automate temporisé non-déterministe, mais l'automate fini sous-jacent peut être déterministe.*

Exemple 2.3.3 *Soit* $\Sigma = \{a, b\}$, $X = \{x\}$, $Q = \{q_0, q_1\}$, $F = \{q_0\}$, $R = \emptyset$, *et* $\delta = \{(q_0, (x > 0,8, a, \{x\}), q_1), (q_1, (x > 0,8, b, \{x\}), q_0)\}$.

L'automate temporisé $\mathcal{A} = (\Sigma, X, Q, q_0, F, R, \delta)$ *est représenté sur la figure suivante :*

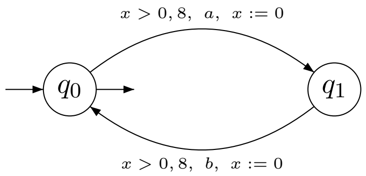


FIG. *2.1 – Exemple d'automate temporisé*

Le langage symbolique de $\mathcal{A}$ *est* $L_{symb}(\mathcal{A}) = ((x > 0,8, a, \{x\})(x > 0,8, b, \{x\}))^*$. *Le langage temporisé de* $\mathcal{A}$ *est* $L(\mathcal{A}) = \{(w, \tau) \mid w \in (ab)^*, |w| = |\tau|, \tau_0 > 0,8$ *et* $\forall\ 0 \leq i < |w|,\ \tau_{i+1} > \tau_i + 0,8\}$.

Première partie

Vérification et contrôle de systèmes o-minimaux

Chapitre 3

Contrôle de systèmes o-minimaux

Les systèmes o-minimaux [LPS00] sont une classe de systèmes hybrides avec une dynamique continue très riche, pouvant par exemple utiliser des polynômes ou la fonction exponentielle, mais dont les transitions discrètes sont limitées, à chaque étape toutes les variables devant être réinitialisées indépendamment de leurs valeurs. Cela permet de découpler les composantes continues et discrètes de tels systèmes. De nombreux travaux sur les systèmes o-minimaux ont été réalisés [Dav99, BMRT04, BM05, KV04, KV06], traitant principalement d'abstractions, de propriétés d'accessibilité et de bisimulations.

Dans ce chapitre, nous nous intéressons au problème de contrôle d'accessibilité pour les systèmes o-minimaux : étant donné un système dont certaines actions sont contrôlables et d'autres incontrôlables (c'est-à-dire appartenant à l'environnement), déterminer s'il est possible de contrôler le système pour atteindre un ensemble d'états gagnants, quoi que fasse l'environnement.

Nous présentons tout d'abord le cadre des jeux non-temporisés et rappelons que la bisimulation est un bon outil pour étudier ces jeux. Nous introduisons alors notre modèle de jeux hybrides dans une structure $\mathcal{M}$, appelés $\mathcal{M}$-jeux. Nous montrons que ces jeux sont fondamentalement différents des jeux classiques (même à un nombre infini d'états) en ce que l'action d'écoulement du temps n'appartient à aucun des deux joueurs ; nous montrons en particulier que la bisimulation à temps abstrait n'est pas un bon outil pour étudier les jeux hybrides (pour deux états bisimilaires, l'un peut être gagnant, et l'autre perdant).

Nous introduisons alors la technique d'encodage des trajectoires par des mots [BMRT04, BM05], nous considérons en particulier l'ensemble des suffixes d'un état. La stabilité par suffixe est une condition plus forte que la bisimilarité à temps abstrait, et nous montrons que cette notion est un bon outil pour étudier le contrôle hybride, cela nous permet de retrouver les résultats de [AMPS98] sur le contrôle temporisé. Nous nous concentrons ensuite sur les jeux o-minimaux et montrons que pour une structure $\mathcal{M}$ décidable et o-minimale, le problème de contrôle est décidable ; nous donnons un algorithme de point-fixe pour calculer l'ensemble des états gagnants, et prouvons que des stratégies gagnantes peuvent être calculées. Nous considérons ensuite un cadre d'observation partielle de la dynamique par le contrôleur : nous proposons un nouveau codage par mots, appelé codage par supermots, qui permet d'obtenir des résultats de décidabilité et de calculabilité similaires à ceux obtenus sous observation totale.

3.1 Jeux finis

Dans cette section, nous rappelons des définitions et résultats concernant les bisimulations de systèmes de transitions (voir [Acz88, Cau95, Hen95]) et les jeux classiques (non temporisés).

3.1.1 Jeux finis classiques

Nous donnons ici les définitions du problème de contrôle sur un graphe fini (aussi appelé jeu fini) et la notion de stratégie (voir [GThW02] pour une vue d'ensemble sur les jeux). Ces définitions sont classiques et seront étendues aux systèmes temporisés par la suite.

Définition 3.1.1 *Un* système de transitions *sur* Σ *est un triplet* $T = (\Sigma, Q, \rightarrow)$ *où* Σ *est un ensemble d'actions (éventuellement infini),* Q *un ensemble d'états (éventuellement infini) et* $\rightarrow \subseteq Q \times \Sigma \times Q$ *une relation de transition.*

Une transition $(q_1, a, q_2) \in \rightarrow$ est aussi notée $q_1 \xrightarrow{a} q_2$. Un système de transitions est dit *fini* si son ensemble d'actions et son ensemble d'états sont finis. Remarquons qu'un automate fini définit de manière canonique un système de transitions.

Définition 3.1.2 *Un* jeu fini *est un automate fini* $(\Sigma, Q, I, \delta, \mathsf{But})$ *où* $\Sigma = \Sigma_c \cup \Sigma_u$ *est un alphabet partitionné correspondant respectivement aux actions contrôlables et incontrôlables.*

Soit $\mathcal{A} = (\Sigma, Q, I, \delta, \mathsf{But})$ un jeu fini. Une action $a \in \Sigma$ est dite *possible* en $q \in Q$ s'il existe $q' \in Q$ tel que $(q, a, q') \in \delta$. Une *exécution* de $\mathcal{A}$ à partir de q_0 est une suite finie ou infinie $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots$; une telle exécution est *gagnante* s'il existe i tel que $q_i \in \mathsf{But}$. Pour une exécution finie $\rho = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n$ nous définissons *dernier*$(\rho) = q_n$. Nous notons $\mathsf{Exec}_f(\mathcal{A})$ l'ensemble des exécutions finies de $\mathcal{A}$.

Nous considérerons des *jeux de contrôle* à deux joueurs : *le contrôleur* et *l'environnement*. Les actions de Σ_c appartiennent au contrôleur et celles de Σ_u à l'environnement. A chaque étape, le contrôleur propose une action contrôlable qui est l'action qu'il veut exécuter ; alors soit cette action, soit une action incontrôlable est prise et l'automate va dans l'état correspondant[1].

Définition 3.1.3 *Une* stratégie[2] *est une application partielle* λ *de* $\mathsf{Exec}_f(\mathcal{A})$ *dans* Σ_c *telle que pour toute exécution* ρ *de* $\mathsf{Exec}_f(\mathcal{A})$*, si* $\lambda(\rho)$ *est définie alors* $\lambda(\rho)$ *est possible en dernier*(ρ)*.*

Soit $\rho = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots$ une exécution, et posons pour tout i, ρ_i le préfixe de longueur i de ρ. L'exécution ρ est *compatible avec la stratégie* λ si pour tout i, $a_{i+1} = \lambda(\rho_i)$ ou $a_{i+1} \in \Sigma_u$. Une exécution ρ est *maximale par rapport à une stratégie* λ si elle est infinie ou si $\lambda(\rho)$ n'est pas défini.

Une stratégie λ est *gagnante à partir d'un état* q si toute exécution maximale par rapport à λ à partir de q compatible avec λ est gagnante.

[1]Il peut y avoir plusieurs états correspondants à une action donnée car nous ne supposons pas que l'automate est déterministe. Nous supposons que l'environnement choisit le prochain état s'il y a plusieurs choix possibles.

[2]Dans le contexte du contrôle, une stratégie est parfois appelée *contrôleur*.

3.1.2 Bisimulation

Nous rappelons la définition de bisimulation sur un système de transitions. Soit $T = (\Sigma, Q, \rightarrow)$ un système de transitions et $\sim$ une relation d'équivalence sur Q. Étant donné $P \subseteq Q$, nous disons que $\sim$ *respecte* P si pour tout $q, q' \in Q$, si $q \in P$ et $q \sim q'$ alors $q' \in P$. Soit $\mathcal{P}$ une partition de Q, nous disons que $\sim$ *respecte* $\mathcal{P}$ si pour tout $P \in \mathcal{P}$, $\sim$ respecte P.

Définition 3.1.4 *Soit $T = (\Sigma, Q, \rightarrow)$ un système de transitions. Une* bisimulation *sur T est une relation d'équivalence $\sim \subseteq Q \times Q$ telle que $\forall q_1, q_1', q_2 \in Q$, $\forall a \in \Sigma$,*

$$\Big(q_1 \sim q_1' \text{ et } q_1 \xrightarrow{a} q_2\Big) \Rightarrow \Big(\exists q_2' \; q_2 \sim q_2' \quad \text{et} \quad q_1' \xrightarrow{a} q_2'\Big)$$

3.1.3 Jeux et bisimulation dans le cadre non temporisé

Dans le cadre non temporisé, la bisimulation est une technique usuelle pour abstraire les jeux : les états bisimilaires peuvent être identifiés comme le précise le théorème suivant.

Théorème 3.1.5 *Soit $\mathcal{A} = (\Sigma, Q, I, \delta, \mathsf{But})$ un jeu fini, $q, q' \in Q$ et $\sim$ une bisimulation qui respecte* But. *Alors il existe une stratégie gagnante à partir de q si, et seulement s'il existe une stratégie gagnante à partir de q'.*

Ce théorème reste vrai pour les jeux discrets à nombre infini d'états [HHM99, dAHM01] et peut être utilisé pour résoudre ces jeux : s'il existe une bisimulation d'index fini, le problème de contrôle peut être réduit à un problème de contrôle sur un graphe fini.

Le problème de contrôle temporisé ne peut pas être vu comme un jeu classique discret à nombre infini d'états à cause de la nature spéciale de l'action d'écoulement du temps, qui n'appartient à aucun des deux joueurs. Il est néanmoins naturel d'essayer d'adapter la technique de bisimulation pour résoudre les problèmes de contrôle temporisé.

3.2 Jeux sur des systèmes dynamiques

3.2.1 Systèmes dynamiques

Soit $\mathcal{M}$ une structure. Nous disons qu'une relation, sous-ensemble ou fonction est *définissable* si elle est définissable par une formule du premier ordre dans $\mathcal{M}$ (voir [Hod97]). Nous notons $\mathsf{Th}(\mathcal{M})$ la théorie de $\mathcal{M}$, c'est-à-dire l'ensemble des énoncés (*i.e.* formules closes du premier ordre) de $\mathcal{M}$ considérés comme vrais. La théorie d'une structure $\mathcal{M}$ est *décidable* s'il existe un algorithme qui, étant donné un énoncé, détermine s'il appartient à $\mathsf{Th}(\mathcal{M})$.

Nous ne considérerons que des structures qui sont des extensions de groupes ordonnés, nous supposons également que $\mathcal{M}$ contient deux symboles de constantes, c'est-à-dire $\mathcal{M} = \langle M, +, 0, 1, <, \cdots \rangle$ et sans perte de généralité nous supposons que $0 < 1$. Nous posons $M^+ = \{\tau \in M \mid \tau \geq 0\}$.

Définition 3.2.1 *Un* système dynamique *est un couple $(\mathcal{M}, \gamma)$ où :*
- *$\mathcal{M} = \langle M, +, 0, 1, <, \cdots \rangle$ est une extension d'un groupe ordonné,*
- *$\gamma : V_1 \times V \rightarrow V_2$ est une fonction définissable dans $\mathcal{M}$ (où $V_1 \subseteq M^{k_1}$, $V \subseteq M$ et $V_2 \subseteq M^{k_2}$)[3].*

[3] Nous utiliserons ces notations dans la suite sans les redéfinir.

La fonction γ *est* la dynamique *du système.*

Quand M est l'ensemble des réels, nous voyons V comme le temps, $V_1 \times V$ comme l'espace-temps, V_1 comme l'espace de départ et V_2 comme l'espace d'arrivée. Nous conservons cette terminologie dans le contexte plus général d'une structure $\mathcal{M}$.

La définition de *système dynamique* prend en compte de nombreux comportements. Nous donnons ci-dessous un premier exemple.

Exemple 3.2.2 *Nous pouvons retrouver la dynamique continue des automates temporisés. Nous posons* $\mathcal{M} = \langle \mathbb{R}, <, +, 0, 1 \rangle$ *et définissons la dynamique* $\gamma : \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ *par* $\gamma(x_1, \cdots, x_n, t) = (x_1 + t, \cdots, x_n + t)$.

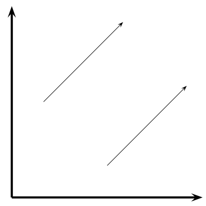


FIG. 3.1 – Dynamique des automates temporisés

Définition 3.2.3 *Soit* $x \in V_1$*, l'ensemble* $\Gamma_x = \{\gamma(x,t) \mid t \in M^+\} \subseteq V_2$ *est appelé* trajectoire *à partir de* x.

Nous définissons un système de transitions associé à un système dynamique ; cette définition est une adaptation à notre contexte de la définition classique de *système de transitions continu* dans le cas de systèmes hybrides (voir [LPS00] par exemple).

Définition 3.2.4 *Soit* $(\mathcal{M}, \gamma)$ *un système dynamique. Il définit un système de transitions* $T_\gamma = (\Sigma, Q, \rightarrow_\gamma)$ *où :*
- *l'ensemble des actions* Σ *est* M^+ *;*
- *l'ensemble des états* Q *est* V_2 *;*
- *la relation de transition* $y_1 \xrightarrow{t}_\gamma y_2$ *est définie par :*

$$\exists x \in V_1,\ \exists t_1, t_2 \in M^+ \text{ tels que } t_1 \leq t_2, \gamma(x, t_1) = y_1,\ \gamma(x, t_2) = y_2 \text{ et } t = t_2 - t_1.$$

3.2.2 $\mathcal{M}$-jeux

Dans cette sous-section, nous définissons les $\mathcal{M}$-automates, qui sont des automates avec des gardes, remises à zéro et dynamiques continues définissables dans la structure $\mathcal{M}$. Nous introduisons alors notre modèle de jeu dynamique qui est un $\mathcal{M}$-automate avec deux ensembles d'actions, un pour chaque joueur ; nous exprimons enfin en terme de stratégie le problème de contrôle dans une classe de $\mathcal{M}$-automates.

Définition 3.2.5 ($\mathcal{M}$-automate) *Un* $\mathcal{M}$-automate $\mathcal{A}$ *est un n-uplet* $(\mathcal{M}, Q, \Sigma, \delta, \gamma)$ *où* $\mathcal{M} = \langle M, +, 0, 1, <, \cdots \rangle$ *est une extension d'un groupe ordonné,* Q *est un ensemble fini d'états,* Σ *un alphabet,* δ *un sous-ensemble fini de transitions* $(q, g, a, R, q') \in Q \times 2^{V_2} \times \Sigma \times (V_2 \to 2^{V_2}) \times Q$ *où* g *et* R *sont définissables dans* $\mathcal{M}$*, et* γ *envoie tout état* $q \in Q$ *sur une dynamique* γ_q : $V_1 \times V \to V_2$.

Nous utilisons une notion générale de remise à zéro : une remise à zéro R est en fait une fonction de V_2 dans 2^{V_2}, qui peut correspondre à une mise à jour non déterministe. Si l'état courant est (q, y), le système ira en (q', y') avec $y' \in R(y)$. Nous appelons *état discret de* $\mathcal{A}$ un élément $q \in Q$, et *dynamique continue* une fonction γ_q pour $q \in Q$.

Un $\mathcal{M}$-automate $\mathcal{A} = (\mathcal{M}, Q, \Sigma, \delta, \gamma)$ définit un système de transitions $T_{\mathcal{A}} = (\Gamma, S, \rightarrow)$ où :

- l'ensemble des actions Γ est $M^+ \cup \Sigma$;
- l'ensemble des états S est $Q \times V_2$;
- la relation de transition est définie par $(q, y) \xrightarrow{e} (q', y')$ si l'une des conditions suivantes est vérifiée :
  - $e \in \Sigma$, et il existe $(q, g, e, R, q') \in \delta$ avec $y \in g$ et $y' \in R(y)$,
  - $e \in M^+$, $q = q'$, et $y \xrightarrow{e}_{\gamma_q} y'$ où γ_q est la dynamique dans l'état q.

Dans la suite, nous nous intéressons aux exécutions formées d'une alternance de transitions continues et discrètes.

Nous aurons également besoin d'une notion de transition plus précise : quand $(q, y) \xrightarrow{\tau} (q, y')$ avec $\tau \in M^+$, cela est dû à un choix d'un $(x, t) \in V_1 \times V$ tel que $\gamma_q(x, t) = y$. Nous disons que $(q, y) \xrightarrow{\tau}_{x,t} (q, y')$ si $\gamma_q(x, t) = y$ et $\gamma_q(x, t + \tau) = y'$. Pour améliorer la lisibilité, nous écrivons parfois $(q, x, t, y) \xrightarrow{\tau} (q, x, t + \tau, y')$ pour $(q, y) \xrightarrow{\tau}_{x,t} (q, y')$. Nous disons qu'une action $(\tau, a) \in M^+ \times \Sigma$ est *possible* dans un état (q, x, t, y) s'il existe (q', x', t', y') et (q'', x'', t'', y'') tels que $(q, x, t, y) \xrightarrow{\tau} (q', x', t', y') \xrightarrow{a} (q'', x'', t'', y'')$. Nous écrivons alors $(q, x, t, y) \xrightarrow{\tau, a} (q'', x'', t'', y'')$. Nous notons $\mathsf{Enb}(q, x, t, y)$ l'ensemble des actions possibles dans (q, x, t, y).

Une *exécution* de $\mathcal{A}$ à partir de (q_0, x_0, t_0, y_0) est une suite finie ou infinie $(q_0, x_0, t_0, y_0) \xrightarrow{\tau_1, a_1} (q_1, x_1, t_1, y_1) \cdots$[4] Une telle exécution est dite *gagnante* s'il existe i tel que $q_i \in \mathsf{But}$. Nous notons $\mathsf{Exec}_f(\mathcal{A})$ l'ensemble des exécutions finies de $\mathcal{A}$. Si ρ est une exécution finie $(q_0, x_0, t_0, y_0) \xrightarrow{\tau_1, a_1} \cdots \xrightarrow{\tau_n, a_n} (q_n, x_n, t_n, y_n)$, nous définissons $dernier(\rho) = (q_n, x_n, t_n, y_n)$.

Définition 3.2.6 ($\mathcal{M}$-jeu) *Un* $\mathcal{M}$-jeu *est un n-uplet* $(\mathcal{M}, Q, \mathsf{But}, \Sigma, \delta, \gamma)$ *où* $\mathsf{But} \subseteq Q$ *est un sous-ensemble d'états gagnants,* $(\mathcal{M}, Q, \Sigma, \delta, \gamma)$ *est un* $\mathcal{M}$*-automate et* $\Sigma = \Sigma_c \cup \Sigma_u$ *est un alphabet partitionné en deux sous-alphabets correspondants aux actions contrôlables et incontrôlables.*

Définition 3.2.7 (Stratégie) *Une* stratégie *est une application partielle* λ *de* $\mathsf{Exec}_f(\mathcal{A})$ *dans* $M^+ \times \Sigma_c$ *telle que pour toute exécution* ρ *de* $\mathsf{Exec}_f(\mathcal{A})$*,* $\lambda(\rho)$ *est possible dans* $dernier(\rho)$*.*

Une stratégie indique ce qui doit être fait pour contrôler le système : à chaque étape, elle indique un délai τ et une action contrôlable qui doit être faite après ce délai. L'action réellement réalisée est alors soit l'action proposée par la stratégie après le délai τ, soit une action incontrôlable après un délai $\tau' \leq \tau$. Remarquons que l'environnement peut avoir à choisir entre différentes arêtes (dans le cas où le jeu est non-déterministe).

Une stratégie λ est *sans mémoire* si pour toutes exécutions finies ρ et ρ', $dernier(\rho) = dernier(\rho')$ implique $\lambda(\rho) = \lambda(\rho')$. Soit $\rho = (q_0, x_0, t_0, y_0) \xrightarrow{\tau_1, a_1} \cdots$ une exécution, et soit pour tout i, ρ_i le préfixe de longueur i de ρ. L'exécution ρ est dite *compatible avec une stratégie* λ si pour tout i, si $\lambda(\rho_i) = (\tau, a)$ alors soit $\tau_{i+1} = \tau$ et $a_{i+1} = a$, soit $\tau_{i+1} \leq \tau$ et $a_{i+1} \in \Sigma_u$. Une exécution ρ est dite *maximale par rapport à une stratégie* λ si elle est infinie ou si $\lambda(\rho)$ n'est pas définie. Une stratégie λ est *gagnante à partir d'un état (q,y)* si pour tout (x, t) tel

[4] Dans le cas d'une exécution finie, celle-ci peut éventuellement finir par une action de délai.

que $\gamma(x,t) = y$, toute exécution maximale par rapport à λ à partir de (q,x,t,y) compatible avec λ est gagnante. L'ensemble des *états gagnants* est l'ensemble des états à partir desquels il existe une stratégie gagnante.

Nous pouvons maintenant définir le problème de contrôle que nous étudierons.

Problème 1 (Problème de contrôle dans une classe $\mathcal{C}$ de $\mathcal{M}$-jeux) *Étant donné un $\mathcal{M}$-jeu $\mathcal{A}$ appartenant à $\mathcal{C}$, et un état initial définissable (q,y), déterminer s'il existe une stratégie gagnante dans $\mathcal{A}$ à partir de (q,y).*

3.2.3 $\mathcal{M}$-jeux et bisimulation

La bisimulation à temps abstrait [Hen95, Dav99, AHLP00] est une abstraction correcte pour les propriétés d'accessibilité dans les systèmes temporisés, et en particulier les $\mathcal{M}$-automates [Bri05]. Nous verrons que cet outil n'est pas suffisant pour traiter les problèmes de contrôle temporisé.

Définition 3.2.8 *Soit $T = (\Gamma, S, \rightarrow)$ un système de transitions, une* bisimulation à temps abstrait *sur T est une relation d'équivalence $\sim \subseteq S \times S$ telle que $\forall q_1, q_1', q_2 \in S$, les deux conditions suivantes sont satisfaites :*

$$\forall a \in \Sigma,\ \left(q_1 \sim q_1' \text{ et } q_1 \xrightarrow{a} q_2\right) \Rightarrow \left(\exists q_2' \in S \text{ tel que } q_2 \sim q_2' \text{ et } q_1' \xrightarrow{a} q_2'\right)$$
$$\forall \tau \in M^+,\ \left(q_1 \sim q_1' \text{ et } q_1 \xrightarrow{\tau} q_2\right) \Rightarrow \left(\exists \tau' \in M^+\ \exists q_2' \in S \text{ tels que } q_2 \sim q_2' \text{ et } q_1' \xrightarrow{\tau'} q_2'\right)$$

Exemple 3.2.9 *Soit $\mathcal{A} = (\mathcal{M}, Q, \mathsf{But}, \Sigma, \delta, \gamma)$ un $\mathcal{M}$-jeu où $\mathcal{M} = \langle \mathbb{R}, <, +, 0, 1, \equiv_2 \rangle$[5], $Q = \{q_1, q_2, q_3\}$, $\mathsf{But} = \{q_2\}$, $\Sigma = \Sigma_c \cup \Sigma_u$ où $\Sigma_c = \{c\}$ (resp. $\Sigma_u = \{u\}$) est l'ensemble des actions contrôlables (resp. incontrôlables). La dynamique dans q_1, $\gamma_{q_1} : \mathbb{R}_{\geq 0} \times \{0,1\} \times \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0} \times \{0,1\}$ est définie par $\gamma_{q_1}(x_1, x_2, t) = (x_1 + t, x_2)$.*

(a) Le $\mathcal{M}$-jeu $\mathcal{A}$ (b) Dynamique dans q_1

FIG. 3.2 – La bisimulation à temps abstrait ne respecte pas les états gagnants

Nous considérons la partition décrite sur la figure 3.2(b). La garde g_C est satisfaite sur les états de C et la garde g_B sur les états de B. Remarquons que cette partition respecte But *et les transitions discrètes.*

Dans ce jeu, le contrôleur peut gagner quand il atteint la pièce C en jouant l'action c, et il perd en entrant dans la pièce B parce qu'il ne peut empêcher l'environnement de jouer l'action u et d'aller dans l'état perdant q_3.

L'état $s_1 = (q_1, (0,1))$ est donc perdant, alors que l'état $s_2 = (q_1, (0,0))$ est gagnant. Or la relation d'équivalence induite par la partition $\{A, B, C\}$ est une bisimulation à temps

[5] $\equiv_2$ dénote la relation « modulo 2 ».

abstrait : les deux états s_1 et s_2 sont donc bisimilaires, mais ne sont pas équivalents pour le jeu. La bisimulation à temps abstrait n'est donc pas correcte pour résoudre le contrôle temporisé, car elle ne peut toujours distinguer les états gagnants des états perdants.

Proposition 3.2.10 *Soit $\mathcal{M}$ une structure et $\mathcal{A}$ un $\mathcal{M}$-jeu. Une partition respectant* But *et induisant une bisimulation à temps abstrait sur $Q \times V_2$ ne respecte pas nécessairement les états gagnants de $\mathcal{A}$.*

3.3 Suffixes et types dynamiques

Dans cette section nous expliquons comment encoder les trajectoires d'un système dynamique par des mots. Cette technique a été introduite dans [BMRT04, BM05] pour étudier les systèmes hybrides o-minimaux. Nous nous intéresserons en particulier à la *partition suffixe* introduite dans [Bri05].

Nous expliquons tout d'abord comment construire les mots associés à une trajectoire. Soit $(\mathcal{M}, \gamma)$ un système dynamique et $\mathcal{P}$ une partition finie de V_2, étant donné $x \in V_1$ nous considérons la trajectoire $\Gamma_x = \{\gamma(x,t) \mid t \in V\}$ et nous lui associons un mot. Nous considérons les ensembles $\{t \in V \mid \gamma(x,t) \in P\}$ pour $P \in \mathcal{P}$. Cela donne lieu à une partition de l'ensemble des temps V. Pour définir un mot sur $\mathcal{P}$ associé à la trajectoire déterminée par x, nous devons définir l'ensemble des intervalles $\mathcal{F}_x = \big\{I \bigm| I$ est un intervalle de temps qui est maximal pour la propriété « $\exists P \in \mathcal{P},\ \forall t \in I,\ \gamma(x,t) \in P$ »$\big\}$. Pour tout x, l'ensemble $\mathcal{F}_x$ est totalement ordonné par l'ordre induit par celui de M. Cela nous permet de définir *le mot sur $\mathcal{P}$ associé à Γ_x*, noté ω_x.

Définition 3.3.1 *Soit $x \in V_1$,* le mot associé à Γ_x *est la fonction $\omega_x : \mathcal{F}_x \to \mathcal{P}$ définie par $\omega_x(I) = P$, où $I \in \mathcal{F}_x$ est tel que $\forall t \in I,\ \gamma(x,t) \in P$.*

Remarque 3.3.2 *Cette définition est en fait une généralisation de la notion classique de mot. En effet elle permet de retrouver les mots finis et infinis mais prend en compte d'autres cas, par exemple des mots indexés par les réels.*

L'ensemble des mots sur $\mathcal{P}$ associés à $(\mathcal{M}, \gamma)$ donne en quelque sorte une description « statique » du système dynamique $(\mathcal{M}, \gamma)$ à travers la partition $\mathcal{P}$ car il indique pour chaque trajectoire la façon dont le système évolue dans la partition $\mathcal{P}$. Pour retrouver l'aspect « dynamique » du système, nous avons besoin de plus d'information, et devons savoir à tout moment quelle partie de la trajectoire a déjà été parcourue, et quelle partie reste à parcourir ; nous introduisons pour cela la notion de suffixe associé à une trajectoire.

Étant donné un point x de l'espace de départ V_1, nous avons associé à x une trajectoire Γ_x et un mot ω_x. Si nous considérons un point (x,t) de l'espace-temps $V_1 \times V$, il correspond à un point $\gamma(x,t)$ de Γ_x. Pour retrouver la position de $\gamma(x,t)$ sur Γ_x depuis ω_x, nous associons à (x,t) un suffixe du mot ω_x noté $\omega_{(x,t)}$. La construction de $\omega_{(x,t)}$ est similaire à la construction de ω_x, nous considérons les ensembles d'intervalles $\mathcal{F}_{(x,t)} = \big\{I \cap \{t' \in V \mid t' \geq t\} \bigm| I \in \mathcal{F}_x\big\}$.

Remarquons qu'étant donné un point (x,t) de l'espace- temps $V_1 \times V$ il existe un unique suffixe $\omega_{(x,t)}$ de ω_x associé à (x,t). Pour un point $y \in V_2$, il peut y avoir plusieurs (x,t) tels que $\gamma(x,t) = y$ et donc plusieurs suffixes associés à y. Le futur d'un mot $y \in V_2$ est non-déterministe, et un seul suffixe $\omega_{(x,t)}$ n'est pas suffisant pour retrouver la dynamique du système de transitions à travers la partition $\mathcal{P}$. Nous introduisons la notion de *type dynamique suffixe* d'un point $y \in V_2$ par rapport à $\mathcal{P}$:

Définition 3.3.3 *Soit $(\mathcal{M},\gamma)$ un système dynamique, $\mathcal{P}$ une partition finie de V_2, y un point de V_2. Le* type dynamique suffixe de y par rapport à $\mathcal{P}$, *noté* $\mathsf{Suf}_{\mathcal{P}}(y)$ *est défini par* $\mathsf{Suf}_{\mathcal{P}}(y) = \{\omega_{(x,t)} \mid \gamma(x,t) = y\}$.

Cela nous permet de définir une relation d'équivalence sur V_2. Nous disons que deux points $y_1, y_2 \in V_2$ ont *mêmes suffixes* si $\mathsf{Suf}_{\mathcal{P}}(y_1) = \mathsf{Suf}_{\mathcal{P}}(y_2)$. Nous notons $\mathsf{Suf}\,(\mathcal{P})$ la partition induite par cette relation d'équivalence. Nous disons qu'une partition $\mathcal{P}$ est *stable par suffixe* si $\mathsf{Suf}(\mathcal{P}) = \mathcal{P}$ (cela implique que si y_1 et y_2 appartiennent à la même pièce de $\mathcal{P}$ alors $\mathsf{Suf}_{\mathcal{P}}(y_1) = \mathsf{Suf}_{\mathcal{P}}(y_2)$).

Soit $y \in V_2$, nous disons qu'il existe un unique suffixe sur $\mathcal{P}$ associé à y si $\mathsf{Suf}_{\mathcal{P}}(y)$ est un singleton. Les exemples suivants illustrent la technique d'encodage par mots.

Exemple 3.3.4 *Nous considérons tout d'abord la dynamique des automates temporisés en dimension deux (voir l'exemple 3.2.2). Nous avons que $\gamma(x_1,x_2,t) = (x_1+t, x_2+t)$. Nous considérons la partition $\mathcal{P} = \{A,B\}$ où $B = [1,2]^2$ et $A = \mathbb{R}^2_{\geq 0} \setminus B$. Les points dont la trajectoire passe par la pièce B mais n'appartenant pas à B ont pour suffixe le mot ABA, les points de la pièce B ont pour suffixe le mot BA, les autres points ont pour suffixe le mot A. La partition suffixe compte donc trois pièces comme décrit sur la figure 3.3.*

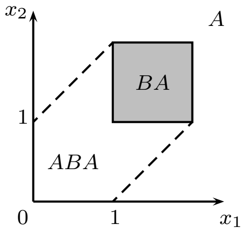


FIG. 3.3 – Suffixes pour la dynamique des automates temporisés

Exemple 3.3.5 *Nous considérons le système dynamique $(\mathcal{M},\gamma)$ où $\mathcal{M} = \langle \mathbb{R}, +, \cdot, 0, 1, <, \sin_{|[0,2\pi]}, \cos_{|[0,2\pi]} \rangle$ [6] et $\gamma : \mathbb{R}^2 \times [0,2\pi] \times \mathbb{R} \to \mathbb{R}^2$ est défini comme suit :*
$\gamma(x_1,x_2,\theta,t) =$

$$\begin{cases} (t.\cos(\theta), t.\sin(\theta)) & \textit{si } (x_1,x_2) = (0,0), \\ (x_1 + t.x_1, x_2 + t.x_2) & \textit{si } (x_1,x_2) \neq (0,0). \end{cases}$$

Nous associons à ce système dynamique la partition $\mathcal{P} = \{A,B,C\}$ où $A = \{(0,0)\}$, $B = \{(\theta\cos(\theta), \theta\sin(\theta)) \mid 0 < \theta \leq 2\pi\}$ et $C = \mathbb{R}^2 \setminus (A \cup C)$. Nous appelons la pièce B la spirale *(voir la figure 3.4). Il y a quatre types dynamiques dans ce système : $\{ACBC\}$ pour le point central $(0,0)$, $\{CBC\}$ pour l'« intérieur » de la spirale, $\{BC\}$ pour la spirale, et $\{C\}$ pour l'« extérieur » de la spirale. Remarquons que même si le système dynamique est infiniment branchant en $(0,0)$, il y a un unique suffixe associé à chaque point y de l'espace d'arrivée.*

Dans les deux exemples précédents, à tout point était associé un unique suffixe ; le prochain exemple permet de retrouver la dynamique continue des automates rectangulaires [HKPV98], et nécessite l'utilisation de type dynamique suffixe (certains points n'ont pas un unique suffixe).

[6] $\sin_{|[0,2\pi]}$ et $\cos_{|[0,2\pi]}$ correspondent aux fonctions sinus et cosinus restreintes au segment $[0,2\pi]$.

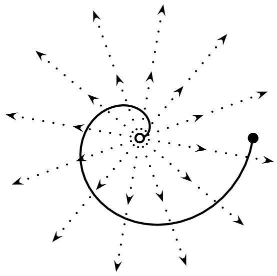


FIG. 3.4 – Le système dynamique de la spirale

Exemple 3.3.6 *Nous considérons le système dynamique* $(\mathcal{M}, \gamma)$ *où* $\mathcal{M} = \langle \mathbb{R}, +, \cdot, 0, 1, < \rangle$ *et* $\gamma : \mathbb{R}^2_{\geq 0} \times [1,2] \times \mathbb{R}_{\geq 0} \to \mathbb{R}^2_{\geq 0}$ *est défini par* $\gamma(x_1, x_2, p, t) = (x_1 + t, x_2 + p \cdot t)$*. Nous associons à ce système dynamique la partition* $\mathcal{P} = \{A, B, C\}$ *où* $B = [2,5] \times [3,4]$*,* $C = [3,5] \times [1,2]$ *et* $A = \mathbb{R}^2_{\geq 0} \setminus (B \cup C)$ *(voir la figure 3.5(a)). Regardons plus précisément les types dynamiques suffixes des points* $y_1 = (1; 2,5)$ *et* $y_2 = (2; 0,5)$*. Nous avons que* $Suf_{\mathcal{P}}(y_1) = \{A, ABA\}$ *et* $Suf_{\mathcal{P}}(y_2) = \{ABA, ACABA\}$*. Même si certains points ont plusieurs suffixes, la partition induite par les types dynamiques suffixes est finie et représentée sur la figure 3.5(b).*

(a) La dynamique

(b) La partition suffixe

FIG. 3.5 – Dynamique rectangulaire

3.4 Résolution des $\mathcal{M}$-jeux

Dans cette section, nous présentons une procédure générale pour calculer l'ensemble des états gagnants d'un $\mathcal{M}$-jeu. Nous montrons que si une partition est stable par suffixe, cette procédure peut être effectuée symboliquement sur les pièces de la partition. Cette procédure n'est pas toujours effective et nous étudierons plus loin des $\mathcal{M}$-structures où chaque étape peut être réalisée algorithmiquement.

3.4.1 Prédécesseurs contrôlables

Comme pour les jeux d'accessibilité classiques [GThW02], un moyen de calculer l'ensemble des états gagnants est de calculer *l'attracteur* de l'ensemble But en itérant l'opérateur des *prédécesseurs contrôlables*.

Soit $\mathcal{A} = (\mathcal{M}, Q, \mathsf{But}, \Sigma, \delta, \gamma)$ un $\mathcal{M}$-jeu. Pour $A \subseteq Q \times V_2$ et $a \in \Sigma$ nous définissons les prédécesseurs contrôlables et incontrôlables comme suit :

$$\mathsf{cPred}(A) = \left\{ (q,y) \in Q \times V_2 \;\middle|\; \begin{array}{l} \exists c \in \Sigma_c,\ c \text{ est possible dans } (q,y), \\ \text{et } \forall (q',y') \in Q \times V_2, \\ (q,y) \xrightarrow{c} (q',y') \Rightarrow (q',y') \in A \end{array} \right\}$$

$$\mathsf{uPred}(A) = \left\{ (q,y) \in Q \times V_2 \;\middle|\; \begin{array}{l} \exists u \in \Sigma_u,\ \exists (q',y') \in Q \times V_2 \text{ tels que} \\ (q,y) \xrightarrow{u} (q',y') \text{ et } (q',y') \in A \end{array} \right\}$$

Comme dans le cadre des jeux temporisés et hybrides [AMPS98, HHM99], nous définissons le prédécesseur temporel *sûr* d'un ensemble A par rapport à un ensemble B : un état (q,y) appartient à $\mathsf{Pred}_t(A,B)$ si en laissant s'écouler le temps, un état de A est finalement atteint, en évitant B. L'opérateur Pred_t est formellement défini comme suit :
$\mathsf{Pred}_t(A,B) =$

$$\left\{ (q,y) \in Q \times V_2 \;\middle|\; \begin{array}{l} \forall (x,t) \text{ tel que } \gamma_q(x,t) = y,\ \exists \tau \in M^+ \\ (q,y) \xrightarrow{\tau}_{x,t} (q',y'),\ (q',y') \in A, \\ \text{et } \mathsf{Post}^{q,x}_{[t,t+\tau]} \subseteq \overline{B} \end{array} \right\}$$

où $\mathsf{Post}^{q,x}_{[t,t+\tau]} = \{(q, \gamma_q(x,t')) \mid t \leq t' \leq t+\tau\}$.

L'opérateur des *prédécesseurs contrôlables* est alors défini par :

$$\pi(A) = A \cup \mathsf{Pred}_t(\mathsf{cPred}(A), \mathsf{uPred}(\overline{A}))$$

Intuitivement, un état (q,y) appartient à $\pi(A)$ s'il appartient déjà à A ou s'il existe un moyen d'attendre un certain temps sans qu'aucune action incontrôlable amène en dehors de A, et à jouer ensuite une action contrôlable qui amène en A.

Nous disons qu'une partition $\mathcal{P}$ est *stable par* π si pour toute pièce $A \in \mathcal{P}$, $\pi(A)$ est une union de pièces de $\mathcal{P}$.

Remarque 3.4.1 *L'opérateur π est définissable dans toute extension d'un groupe ordonné, donc si A est définissable dans $\mathcal{M}$, $\pi(A)$ l'est aussi.*

Nous calculerons l'ensemble des états gagnants en itérant l'opérateur π. En notant $\pi^*(\mathsf{But}) = \bigcup_{k\geq 0} \pi^k(\mathsf{But})$, nous montrerons que le calcul itératif de $\pi^*(\mathsf{But})$ se stabilise, et que l'ensemble des états gagnants du jeu est précisément $\pi^*(\mathsf{But})$. Ceci nous permettra plus tard d'obtenir des résultats d'effectivité et de définissabilité pour les états gagnants et les stratégies gagnantes sous certaines hypothèses sur la structure sous-jacente.

Proposition 3.4.2 *Soit $\mathcal{A} = (\mathcal{M}, Q, \mathsf{But}, \Sigma, \delta, \gamma)$ un $\mathcal{M}$-jeu, et soit $(q,y) \in Q \times V_2$. S'il existe $n \in \mathbb{N}$ tel que $\pi^n(\mathsf{But}) = \pi^{n+1}(\mathsf{But})$ alors $\pi^*(\mathsf{But}) = \pi^n(\mathsf{But})$ est l'ensemble des états gagnants de $\mathcal{A}$.*

Preuve. Nous prouvons tout d'abord que si $(q,y) \in \pi^*(\mathsf{But})$ alors il existe une stratégie gagnante à partir de (q,y). Pour cela, nous définissons une stratégie gagnante sans mémoire pour tout $(q,y) \in \pi^*(\mathsf{But})$. Par abus de notation, nous définissons cette stratégie λ sur les états (q,x,t,y) au lieu des exécutions, ce qui est possible car elle est sans mémoire.

Nous définissons λ sur les ensembles $\bigcup_{0\leq i\leq k} \pi^i(\mathsf{But})$ par récurrence sur k, et montrons que c'est une stratégie gagnante. Si $k=0$, nous posons λ comme non définie, elle est donc gagnante pour les états de But.

Supposons maintenant que λ est déjà définie sur $W = \bigcup_{0\leq i\leq k} \pi^i(\mathsf{But})$ et est gagnante sur ces états. Nous définissons alors λ sur $\pi(W)$. Soit $(q,x,t,y) \in Q \times V_1 \times V \times V_2$: si $(q,y) \in W$, λ est déjà définie ; si $(q,y) \in \pi(W) \setminus W$, nous savons que $(q,y) \in \mathsf{Pred}_t\left(\mathsf{cPred}(W), \mathsf{uPred}(\overline{W})\right)$. Pour tout (x,t) tel que $\gamma_q(x,t) = y$, il existe $\tau \in M^+$ et $c \in \Sigma_c$ avec (τ, c) possible dans (q,y) et $(q,y) \xrightarrow{\tau,c}_{x,t} (q',y')$ implique $(q',y') \in W$ et $\mathsf{Post}^{q,x}_{[t,t+\tau]} \subseteq \overline{\mathsf{uPred}(\overline{W})}$. Nous posons $\lambda(q,x,t,y) = (\tau, c)$ et montrons que ce choix est gagnant.

Soit $\rho = (q,x,t,y) \xrightarrow{\tau_1,a_1} (q_1,x_1,t_1,y_1) \xrightarrow{\tau_2,a_2} \cdots$ une exécution compatible avec λ. Soit $\tau_1 = \tau$ et $a_1 = c$, dans ce cas $(q_1,y_1) \in W$; soit $\tau_1 \leq \tau$ et $a_1 \in \Sigma_u$, dans ce cas $(q,y) \xrightarrow{\tau_1}_{x,t} (q',y') \xrightarrow{a_1} (q_1,y_1)$ avec $(q',y') \notin \mathsf{uPred}(\overline{W})$ donc $(q_1,y_1) \in W$. Dans les deux cas, $(q_1,y_1) \in W$ donc par hypothèse de récurrence ρ est gagnant.

Nous montrons maintenant que s'il existe une stratégie gagnante λ à partir de (q,y) alors $(q,y) \in \pi^*(\mathsf{But})$. Posons $W = \pi^*(\mathsf{But})$, et supposons par l'absurde que $(q,y) \notin W$, nous construisons une exécution compatible avec λ qui n'est pas gagnante. Par hypothèse $\pi(W) = W$ (car $\pi^*(\mathsf{But}) = \pi^n(\mathsf{But}) = \pi^{n+1}(\mathsf{But}) = \pi(\pi^*(\mathsf{But}))$), donc $(q,y) \notin \pi(W)$; il existe donc $(x,t) \in V_1 \times V$ tel que $\gamma_q(x,t) = y$, et pour tout $\tau \in M^+$, $(q,y) \rightarrow^{\tau}_{x,t} (q',y')$ implique $(q',y') \notin \mathsf{cPred}(W)$ ou $\mathsf{Post}^{q,x}_{[t,t+\tau]} \cap \mathsf{uPred}(\overline{W}) \neq \emptyset$. Fixons un tel $(x,t) \in V_1 \times V$, et soit $(\tau, c) = \lambda(q,x,t,y)$.

Il existe (q_1,x_1,t_1,y_1) avec $(q_1,y_1) \notin W$ tel que soit $(q,x,t,y) \xrightarrow{\tau,c} (q_1,x_1,t_1,y_1)$ soit il existe $\tau' \leq \tau$ et $u \in \Sigma_u$ avec $(q,x,t,y) \xrightarrow{\tau',u} (q_1,x_1,t_1,y_1)$. Dans les deux cas, l'exécution construite est compatible avec λ. Comme $(q_1,y_1) \notin W$ nous pouvons appliquer à nouveau cet argument et construire par récurrence une exécution $\rho = (q,x,t,y) \xrightarrow{\tau_1,a_1} (q_1,x_1,t_1,y_1) \xrightarrow{\tau_2,a_2} \cdots$ compatible avec λ telle que pour tout i, $(q_i,x_i,t_i,y_i) \notin W$. Par définition de W, pour tout i, $q_i \notin \mathsf{But}$, ce qui contredit le fait que λ est une stratégie gagnante. $\square$

Nous déduisons maintenant un résultat algorithmique de la proposition 3.4.2. L'ensemble des états gagnants est $\pi^*(\mathsf{But})$ mais cela n'implique pas que nous puissions calculer cet ensemble car beaucoup de $\mathcal{M}$-structures sont intrinsèquement indécidables. Le corollaire suivant établit que si certaines conditions sur la structure et sur π sont satisfaites, alors cette procédure donne une solution algorithmique au problème de contrôle.

Corollaire 3.4.3 *Soit $\mathcal{M}$ une structure telle que $\mathsf{Th}(\mathcal{M})$ est décidable[7]. Soit $\mathcal{C}$ une classe de $\mathcal{M}$-jeux telle que pour tout $\mathcal{A} \in \mathcal{C}$, il existe une partition finie $\mathcal{P}$ de $Q \times V_2$ définissable dans $\mathcal{M}$, respectant* But[8]*, et stable par π. Alors le problème de contrôle dans la classe $\mathcal{C}$ est décidable. De plus si $\mathcal{A} \in \mathcal{C}$, l'ensemble des états gagnants de $\mathcal{A}$ est calculable.*

Preuve. Soit $\mathcal{M}$ une structure et $\mathcal{C}$ une classe d'automates satisfaisant les hypothèses du corollaire, et soit $\mathcal{A} \in \mathcal{C}$. Comme $\mathcal{P}$ est stable par π, $\pi^*(\mathsf{But})$ est une union finie de pièces de $\mathcal{P}$. Il existe donc $n \in \mathbb{N}$ tel que $\pi^*(\mathsf{But}) = \pi^n(\mathsf{But})$. La proposition 3.4.2 montre donc que l'ensemble des états gagnants est $\pi^*(\mathsf{But})$.

Comme π et But sont définissables, nous avons que $\pi^i(\mathsf{But})$ est définissable et comme $\mathsf{Th}(\mathcal{M})$ est décidable nous pouvons tester si $\pi^i(\mathsf{But}) = \pi^{i+1}(\mathsf{But})$, nous pouvons donc calculer une représentation effective de $\pi^*(\mathsf{But})$. Comme $\mathsf{Th}(\mathcal{M})$ est décidable, si un état (q,y) est

[7]Nous rappelons qu'une théorie $\mathsf{Th}(\mathcal{M})$ est décidable s'il existe un algorithme qui peut déterminer si un énoncé (c'est-à-dire une formule sans variable libre) appartient à la théorie (c'est-à-dire est vrai).

[8]C'est-à-dire que But est une union de pièces de $\mathcal{P}$

définissable nous pouvons tester si $(q, y) \in \pi^*(\mathsf{But})$. Le problème de contrôle dans $\mathcal{C}$ est donc décidable. $\square$

3.4.2 Stabilité de $\mathsf{Suf}(\mathcal{P})$ par π

Dans la sous-section 3.2.3, nous avons donné un contre-exemple montrant que la bisimulation à temps abstrait n'est pas toujours correcte pour résoudre les problèmes de contrôle. La principale raison est que la partition induite par une bisimulation à temps abstrait n'est pas toujours stable par l'opérateur π.

Nous présentons maintenant une condition suffisante pour qu'une partition soit stable par l'opérateur π : si une partition est stable par cPred, uPred et par suffixe[9], elle est correcte pour résoudre les problèmes de contrôle.

Proposition 3.4.4 *Soit $\mathcal{A}$ un $\mathcal{M}$-jeu, $\mathcal{P}$ une partition de $Q \times V_2$ et π l'opérateur des prédécesseurs contrôlables. Si $\mathcal{P}$ respecte* But*, est stable par* cPred*,* uPred *et par suffixe, alors $\mathcal{P}$ est stable par l'opérateur π.*

Preuve. Nous fixons un état discret q de l'automate et prenons $y_1, y_2 \in V_2$ tels qu'il existe $A \in \mathcal{P}$ avec $(q, y_1) \in A$ et $(q, y_2) \in A$. Nous montrons que si $(q, y_1) \in \pi(X)$ pour $X \in \mathcal{P}$ alors $(q, y_2) \in \pi(X)$.

Comme $(q, y_1) \in \pi(X)$, pour tout $(x, t) \in V_1 \times V$ tel que $\gamma_q(x, t) = y_1$, il existe $\tau_1 \in M^+$ et y_1' tels que $y_1 \xrightarrow{\tau_1}_{x,t} y_1'$, $y_1' \in \mathsf{cPred}(X)$, et $\mathsf{Post}^{q,x}_{[t,t+\tau_1]} \subseteq \overline{\mathsf{uPred}(\overline{X})}$.

Soit $\omega_{(x,t)}(y_1)$ un suffixe de y_1 associé à x et t. En terme de mots, les trois conditions précédentes signifient qu'il existe un préfixe de $\omega_{(x,t)}(y_1)$ dont la dernière lettre est dans $\mathsf{cPred}(X)$ et sans occurrence de lettres de $\mathsf{uPred}(\overline{X})$ (cela a un sens car par hypothèse $\mathsf{cPred}(X)$ et $\mathsf{uPred}(\overline{X})$ sont des unions de pièces de $\mathcal{P}$). Notons $\omega^p_{(x,t)}(y_1)$ ce préfixe.

Comme $\mathcal{P} = \mathsf{Suf}(\mathcal{P})$, et (q, y_1) et (q, y_2) appartiennent à la même pièce de $\mathcal{P}$, nous avons que $\mathsf{Suf}_{\mathcal{P}}(y_1) = \mathsf{Suf}_{\mathcal{P}}(y_2)$. Soit $(x, t) \in V_1 \times V$ tel que $\gamma_q(x, t) = y_2$ et soit $\omega_{(x,t)}(y_2)$ le suffixe de y_2 associé à x et t. Comme $\mathsf{Suf}_{\mathcal{P}}(y_1) = \mathsf{Suf}_{\mathcal{P}}(y_2)$, il existe (x', t') tel que $\gamma_q(x', t') = y_1$ et $\omega_{(x,t)}(y_2) = \omega_{(x',t')}(y_1)$, donc le préfixe $\omega^p_{(x',t')}(y_1)$ est un préfixe de $\omega_{(x,t)}(y_2)$.

Nous pouvons donc trouver $y_2' \in \mathsf{cPred}(X)$ et $\tau_2 \in M^+$ tels que $y_2 \xrightarrow{\tau_2}_{x,t} y_2'$, $y_2' \in \mathsf{cPred}(X)$, et $\mathsf{Post}^{q,x}_{[t,t+\tau_2]} \subseteq \overline{\mathsf{uPred}(\overline{X})}$. Donc $(q, y_2) \in \pi(X)$. $\square$

Comme corollaire immédiat de cette proposition et du corollaire 3.4.3, nous obtenons le résultat de décidabilité suivant.

Corollaire 3.4.5 *Soit $\mathcal{M}$ une structure telle que* $\mathsf{Th}(\mathcal{M})$ *est décidable. Soit $\mathcal{C}$ une classe de $\mathcal{M}$-jeux telle que pour tout $\mathcal{A} \in \mathcal{C}$, il existe une partition finie $\mathcal{P}$ de $Q \times V_2$ définissable dans $\mathcal{M}$, respectant* But*, stable par* cPred*,* uPred *et par suffixe. Alors le problème de contrôle dans la classe $\mathcal{C}$ est décidable, et si $\mathcal{A} \in \mathcal{C}$, l'ensemble des états gagnants est calculable.*

[9]Nous étendons de manière naturelle la notion de partition stable par suffixe (voir page 34) pour une partition d'un $\mathcal{M}$-jeu respectant les états discrets.

Remarque 3.4.6 *La stabilité par suffixe est une condition plus forte que la bisimulation à temps abstrait [Bri05], et nous montrons ici que c'est l'un des bons outils pour résoudre les problèmes de contrôle. Dans l'exemple 3.2.9 la partition $\mathcal{P}$ est une bisimulation à temps abstrait qui n'est pas stable par suffixe. En effet $s_1,\ s_2 \in A$ mais $Suf_{\mathcal{P}}(s_1) \neq Suf_{\mathcal{P}}(s_2)$.*

Remarque 3.4.7 *Les résultats de cette section permettent de retrouver les résultats sur le contrôle des automates temporisés de [AMPS98]. En effet, considérons la partition finie classique pour les automates temporisés induite par le graphe des régions [AD94]. Appelons $\mathcal{P}_R$ cette partition, et remarquons que $\mathcal{P}_R$ est définissable dans la structure $\langle \mathbb{R}, <, +, 0, 1\rangle$. $\mathcal{P}_R$ est stable par* cPred *et* uPred*. Par l'exemple 3.2.2 la dynamique continue des automates temporisés est définissable dans $\langle \mathbb{R}, <, +, 0, 1\rangle$. Nous pouvons donc encoder les trajectoires continues des automates temporisés par des mots. Il est alors facile de voir que $\mathsf{Suf}(\mathcal{P}_R) = \mathcal{P}_R$. Par le corollaire 3.4.5 nous obtenons la décidabilité et le calcul des états gagnants dans les jeux temporisés [AMPS98].*

Corollaire 3.4.8 *Le problème de contrôle dans la classe des automates temporisés est décidable. De plus l'ensemble des états gagnants $\pi^*(\mathsf{But})$ est calculable.*

3.5 Jeux o-minimaux

Dans cette section, nous nous intéressons au cas particulier des jeux o-minimaux (c'est-à-dire les $\mathcal{M}$-jeux où $\mathcal{M}$ est une structure o-minimale et des restrictions sont faites sur les remises à zéro) [LPS00].

Nous rappelons quelques définitions et résultats concernant la o-minimalité [PS86] (pour une étude plus complète se référer à [vdD98]). Nous considérerons alors des structures o-minimales dont la théorie est décidable pour obtenir des résultats de décidabilité et de calculabilité dans le cadre du contrôle.

Définition 3.5.1 *Une extension d'un groupe ordonné $\mathcal{M} = \langle M, <, \cdots\rangle$ est* o-minimale *si tout sous-ensemble définissable de M est une union finie de points et d'intervalles ouverts.*

Dans une structure o-minimale, les sous-ensembles définissables sont donc les plus simples possibles : ce sont ceux qui sont définissables dans $\langle M, <\rangle$.

Exemple 3.5.2 *Il y a beaucoup d'exemples de structures o-minimales comme le groupe ordonné des rationnels $\langle \mathbb{Q}, <, +, 0, 1\rangle$, le corps ordonné des réels $\langle \mathbb{R}, <, +, \cdot, 0, 1\rangle$, le corps des réels muni de la fonction exponentielle.*

Le principal théorème que nous utiliserons sur les structures o-minimales est un cas particulier du théorème de finitude uniforme [KPS86, Théorème 0.3(a)], (voir aussi [vdD98, Corollaire 3.6, p. 60]) :

Théorème 3.5.3 ([KPS86]) *Soit $\mathcal{M} = \langle M, <, \cdots\rangle$ une structure o-minimale et m un entier, soit $S \subseteq M^m \times M$ un sous-ensemble définissable. Pour $x \in M^m$, soit $S_x = \{y \in M \mid (x, y) \in S\}$. Alors il existe un entier N_S tel que pour tout $x \in M^m$, l'ensemble $S_x \subseteq M$ est une union d'au plus N_S points ou intervalles ouverts définissables.*

3.5.1 Généralités sur les jeux o-minimaux

Définition 3.5.4 *Soit $\mathcal{A}$ un $\mathcal{M}$-automate (resp. $\mathcal{M}$-jeu), nous disons que $\mathcal{A}$ est* o-minimal *si la structure $\mathcal{M}$ est o-minimale et si toutes les transitions (q, g, a, R, q') de $\mathcal{A}$ appartiennent*[10] *à $Q \times 2^{V_2} \times \Sigma \times 2^{V_2} \times Q$.*

Remarquons que dans un jeu o-minimal $\mathcal{A}$, les gardes, remises à zéro et dynamiques sont définissables dans la structure o-minimale sous-jacente. Nous notons $\mathcal{P}_{\mathcal{A}}$ la partition la plus grossière de l'espace d'états $S = Q \times V_2$ qui respecte But et toutes les gardes et remises à zéro de $\mathcal{A}$. Notons que $\mathcal{P}_{\mathcal{A}}$ est une partition finie et définissable de S.

La forte condition sur les remises à zéro implique que $\mathcal{P}_{\mathcal{A}}$ est stable par cPred et uPred et permet de découpler les composantes continues et discrètes d'un système hybride o-minimal [LPS00]. Remarquons également que tout raffinement de $\mathcal{P}_{\mathcal{A}}$ est stable par cPred et uPred.

Les jeux o-minimaux sont en particulier des systèmes hybrides o-minimaux (comme définis dans [BM05, Bri06]). Nous utiliserons les résultats suivants sur ces systèmes.

Théorème 3.5.5 ***[Bri06]*** *Soit $\mathcal{A}$ un automate o-minimal, alors :*
- *pour tout $y \in V_2$, alors $\mathsf{Suf}_{\mathcal{P}_{\mathcal{A}}}(y)$ est composé d'un ensemble fini de mots finis sur $\mathcal{P}_{\mathcal{A}}$,*
- *la partition $\mathsf{Suf}(\mathcal{P}_{\mathcal{A}})$ est finie et définissable,*
- *s'il existe un unique suffixe sur $\mathcal{P}_{\mathcal{A}}$ associé à tout $y \in V_2$ alors $\mathsf{Suf}(\mathcal{P}_{\mathcal{A}})$ est une bisimulation à temps abstrait.*

3.5.2 Jeux o-minimaux avec un unique suffixe

Dans cette sous-section, nous appliquons les résultats généraux de la section 3.4 au cas particulier des jeux o-minimaux satisfaisant la condition d'unicité du suffixe. Nous prouvons tout d'abord que toute partition induisant une bisimulation à temps abstrait est stable par suffixe :

Lemme 3.5.6 *Soit $\mathcal{A}$ un jeu o-minimal et $\mathcal{P}$ une partition induisant une bisimulation à temps abstrait. S'il existe un unique suffixe sur $\mathcal{P}$ associé à tout $(q, y) \in Q \times V_2$ alors $\mathcal{P} = \mathsf{Suf}(\mathcal{P})$.*

Preuve. Nous fixons un état discret $q \in Q$ et pour améliorer la lisibilité nous notons y l'état (q, y). Nous notons également $y \rightarrow_\gamma y'$ pour $\exists\ \tau \geq 0,\ y \xrightarrow{\tau}_\gamma y'$.

Soit $A_1 \in \mathcal{P}$ et $y_1, y_1' \in A_1$. Montrons que $\mathsf{Suf}_{\mathcal{P}}(y_1) = \mathsf{Suf}_{\mathcal{P}}(y_1')$. Dans le cadre des jeux o-minimaux, le suffixe associé à un point est un mot fini, soit donc $\omega_1 = A_1 ... A_n$ le suffixe unique associé à y_1, nous pouvons construire la suite de transitions suivante :

$$y_1 \rightarrow_\gamma y_2 \rightarrow_\gamma ... \rightarrow_\gamma y_n,$$

avec $y_i \in A_i$ pour tout $1 \leq i \leq n$. Comme y_1 et y_1' sont bisimilaires, nous pouvons construire une suite similaire de transitions

$$y_1' \rightarrow_\gamma y_2' \rightarrow_\gamma ... \rightarrow_\gamma y_n',$$

avec $y_i' \in A_i$ pour tout $1 \leq i \leq n$.

Montrons maintenant que l'hypothèse d'unicité du suffixe implique qu'il existe $x \in V_1$ et $t_1,...,t_n \in M$ avec $t_1 \leq ... \leq t_n$ tels que $\gamma(x, t_1) = y_1'$, et $\gamma(x, t_i) \in A_i$ pour tout $1 \leq i \leq n$;

[10] Il s'agit du cas particulier où les remises à zéro du $\mathcal{M}$-jeu sont des fonctions constantes.

ce qui montre que ω_1 est un sous-mot de ω_1' (l'unique suffixe associé à y_1'). Nous pouvons clairement trouver x, t_1, t_2 avec $t_1 \leq t_2$, $y_1' = \gamma(x, t_1) \in A_1$ et $\gamma(x, t_2) \in A_2$ (car $y_1' \rightarrow_\gamma y_2'$). Supposons par l'absurde qu'il existe x, t_1, t_2 tels que $t_1 \leq t_2$, $\gamma(x, t_1) \in A_1$, $\gamma(x, t_2) \in A_2$ et $\gamma(x, t_3) \notin A_3$ pour tout $t_3 \geq t_2$. Par l'hypothèse d'unicité du suffixe, l'unique suffixe associé à y_2' ne contient pas la lettre A_3. Cela contredit l'existence de la transition $y_2' \rightarrow_\gamma y_3'$ où $y_3' \in A_3$. Nous pouvons donc trouver t_3 satisfaisant les conditions requises. En répétant cet argument nous trouvons les autres t_i.

De même, ω_1' est un sous-mot de ω_1. Donc $\omega_1 = \omega_1'$ car ce sont des mots finis. Nous avons montré que $\mathsf{Suf}_\mathcal{P}(y_1) = \mathsf{Suf}_\mathcal{P}(y_1')$, ce qui achève la preuve du lemme. □

Proposition 3.5.7 *Soit $\mathcal{A}$ un jeu o-minimal, $\mathcal{P}$ une partition induisant une bisimulation à temps abstrait et respectant* But*, les gardes et les remises à zéro de $\mathcal{A}$. S'il existe un unique suffixe sur $\mathcal{P}$ associé à tout $(q, y) \in Q \times V_2$ alors $\mathcal{P}$ est stable par π.*

Preuve. En appliquant le théorème 3.5.5 et le lemme 3.5.6, nous obtenons que $\mathsf{Suf}(\mathcal{P}_\mathcal{A})$ est stable par suffixe, et en utilisant la proposition 3.4.4 nous obtenons que celle-ci est stable par π. □

Remarque 3.5.8 *Le théorème 3.5.5 assure que la partition suffixe* $\mathsf{Suf}(\mathcal{P}_\mathcal{A})$ *est une partition satisfaisant les hypothèses de la proposition ci-dessus si la condition d'unicité du suffixe est vérifiée.*

Remarque 3.5.9 *La proposition 3.5.7 ne contredit pas la proposition 3.2.10 qui montre que la bisimulation à temps abstrait n'est pas en général un outil correct pour résoudre les problèmes de contrôle. En effet, l'exemple de la figure 3.2(b) satisfait la condition d'unicité du suffixe mais n'est pas o-minimal. Il est également possible de construire un exemple qui est o-minimal mais ne satisfait pas la condition d'unicité du suffixe (voir l'exemple 3.5.10). Les hypothèses d'o-minimalité et d'unicité du suffixe sont donc nécessaires dans la proposition 3.5.7.*

Exemple 3.5.10 *Considérons le jeu $\mathcal{A}$ de l'exemple 3.2.9. Nous définissons un jeu o-minimal $\mathcal{A}_o$ à partir de $\mathcal{A}$. La structure o-minimale sous-jacente $\mathcal{M}$ est $\langle \mathbb{R}, <, +, 0, 1\rangle$. Le jeu o-minimal $\mathcal{A}_o$ a le même ensemble d'états discrets, le même ensemble* But*, le même ensemble d'actions et le même automate fini sous-jacent que $\mathcal{A}$ (c'est-à-dire que la figure 3.2(a) représente aussi $\mathcal{A}_o$). Les deux différences entre $\mathcal{A}$ et $\mathcal{A}_o$ sont les gardes et la dynamique continue dans q_1. Définissons tout d'abord les gardes. Nous avons que $g_B = \{(y, 0) \mid y > 0\} \cup \{(0, 1)\}$ et $g_C = \{(0, 0)\} \cup \{(y, 1) \mid y > 0\}$. La dynamique continue dans q_1 est représentée sur la figure 3.6. $\gamma_{q_1} : \mathbb{R} \times \{0, 1\}^2 \times \mathbb{R}_{\geq 0} \rightarrow \mathbb{R} \times \{0, 1\}$ est définie par :*

$$\gamma_{q_1}(x_1, x_2, p, t) = \begin{cases} (x_1 + t, x_2) & \text{si } p = 0 \\ (x_1, x_2) & \text{si } p = 1 \text{ et } x_1 \leq 0 \\ (x_1, x_2) & \text{si } p = 1,\ x_1 > 0 \text{ et } t = 0 \\ (0, x_2) & \text{si } p = 1,\ x_1 > 0 \text{ et } t > 0 \end{cases}$$

g_B, g_C et γ_{q_1} sont clairement définissables dans $\mathcal{M}$. Comme dans l'exemple 3.2.9, nous pouvons facilement vérifier que la relation d'équivalence induite par la partition $\mathcal{P} = \{A, B, C\}$ (où

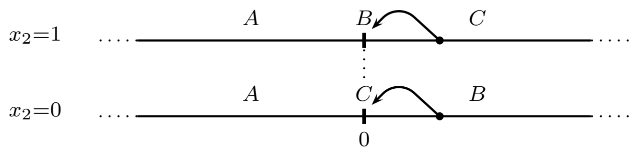


FIG. 3.6 – Un jeu o-minimal où la bisimulation à temps abstrait n'est pas correcte pour le contrôle

$B = g_B$, $C = g_C$ et $A = V_2 \setminus (g_B \cup g_C)$) est une bisimulation à temps abstrait sur $\mathcal{A}_o$. Les états $s_1 = (q_1, (-1,0))$ et $s_2 = (q_1, (-1,1))$ sont bisimilaires alors que s_1 est gagnant et pas s_2. Remarquons que cet exemple ne satisfait pas l'hypothèse d'unicité du suffixe : par exemple les points $(x_1, 1)$ avec $x_1 > 0$ ont $\{C, CB\}$ pour ensemble de suffixes.

Remarque 3.5.11 *L'hypothèse d'unicité du suffixe de la proposition 3.5.7 est satisfaite par les comportements continus décrits dans [LPS00] (où il y a une unique trajectoire passant par chaque point). Des systèmes plus complexes entrent dans ce cadre, par exemple la dynamique de la spirale (exemple 3.5.14) qui est un système infiniment branchant avec un unique suffixe.*

3.5.3 Relâcher l'hypothèse d'unicité du suffixe

Dans la sous-section précédente, sous l'hypothèse d'unicité du suffixe et en appliquant la proposition 3.4.4, nous avons montré que $\mathsf{Suf}(\mathcal{P}_\mathcal{A})$ est stable par π. Nous montrerons que nous pouvons relâcher l'hypothèse d'unicité du suffixe et obtenir la stabilité de $\mathsf{Suf}(\mathcal{P}_\mathcal{A})$ par π. À partir de maintenant et dans le reste de ce chapitre nous relâchons l'hypothèse d'unicité du suffixe ; ce cadre est plus général et nous ne pouvons plus appliquer la proposition 3.4.4 car la partition $\mathsf{Suf}(\mathcal{P}_\mathcal{A})$ n'est pas stable par suffixe en général (même si le système est o-minimal), c'est le cas par exemple pour la sémantique rectangulaire décrite sur la figure 3.5(a).

Le théorème 3.5.5 assure que la partition $\mathsf{Suf}(\mathcal{P}_\mathcal{A})$ est finie ; même si elle n'est pas toujours une bisimulation à temps abstrait (comme sur la figure 3.5(b)) ; nous montrons que $\mathsf{Suf}(\mathcal{P}_\mathcal{A})$ est stable par π.

Proposition 3.5.12 *Soit $\mathcal{A}$ un jeu o-minimal. La partition suffixe $\mathsf{Suf}(\mathcal{P}_\mathcal{A})$ est finie et stable par π.*

Preuve. Nous fixons un état discret q de $\mathcal{A}$, et soit $y_1, y_2 \in V_2$ tels que $\mathsf{Suf}_{\mathcal{P}_\mathcal{A}}(y_1) = \mathsf{Suf}_{\mathcal{P}_\mathcal{A}}(y_2)$, c'est-à-dire que y_1 et y_2 appartiennent à la même pièce de $\mathsf{Suf}(\mathcal{P}_\mathcal{A})$. Soit $X \in \mathsf{Suf}(\mathcal{P}_\mathcal{A})$, nous montrons que $y_1 \in \pi(X)$ si, et seulement si $y_2 \in \pi(X)$. Nous montrons uniquement que si $y_1 \in \pi(X)$ alors $y_2 \in \pi(X)$, l'autre sens est similaire.

Rappelons la définition de l'opérateur des prédécesseurs contrôlables :

$$\pi(X) = X \cup \mathsf{Pred}_t(\mathsf{cPred}(X), \mathsf{uPred}(\overline{X}))$$

Condidérons tout d'abord le cas simple où $y_1 \in X$. Comme $\mathsf{Suf}_{\mathcal{P}_\mathcal{A}}(y_1) = \mathsf{Suf}_{\mathcal{P}_\mathcal{A}}(y_2)$ et $X \in \mathsf{Suf}(\mathcal{P}_\mathcal{A})$, nous avons clairement que $y_2 \in X$ et donc que $y_2 \in \pi(X)$.

Considérons maintenant le cas où $y_1 \in \pi(X) \setminus X$. Nous avons que pour tout $(x_1, t_1) \in V_1 \times V$ tel que $\gamma_q(x_1, t_1) = y_1$, il existe $\tau_1 \in M^+$ et y_1' tels que $y_1 \xrightarrow{\tau_1}_{x_1,t_1} y_1'$ avec $y_1' \in \mathsf{cPred}(X)$ et $\mathsf{Post}^{q,x}_{[t_1, t_1+\tau_1]} \subseteq \overline{\mathsf{uPred}(\overline{X})}$.

Par la définition des transitions d'un jeu o-minimal, nous avons que $\mathsf{cPred}(X)$ est une union de pièces de $\mathcal{P}_\mathcal{A}$, et de même pour $\mathsf{uPred}(\overline{X})$.

Soit $\omega_{(x_1,t_1)}(y_1)$ le suffixe de y_1 associé à x_1 et t_1. En terme de mots, cette condition signifie qu'il existe un préfixe $\omega_{(x_1,t_1)}(y_1)$ dont la dernière lettre est dans $\mathsf{cPred}(X)$ et sans occurrence de lettres de $\mathsf{uPred}(\overline{X})$ (ce qui a un sens car $\mathsf{cPred}(X)$ et $\mathsf{uPred}(\overline{X})$ sont des unions de pièces de $\mathcal{P}_\mathcal{A}$). Appelons $\omega^p_{(x_1,t_1)}(y_1)$ ce préfixe. Comme $\mathsf{Suf}_{\mathcal{P}_\mathcal{A}}(y_1) = \mathsf{Suf}_{\mathcal{P}_\mathcal{A}}(y_2)$, il existe $(x_2,t_2) \in V_1 \times M^+$ tel que $\gamma_q(x_2,t_2) = y_2$ et $\omega_{(x_1,t_1)}(y_1) = \omega_{(x_2,t_2)}(y_2)$. En particulier, le préfixe $\omega^p_{(x_1,t_1)}(y_1)$ est un préfixe de $\omega_{(x_2,t_2)}(y_2)$. Nous pouvons donc trouver $\tau_2 \in M^+$ et $y_2' \in \mathsf{cPred}(X)$ tels que $y_2 \xrightarrow{\tau_2}_{x,t} y_2'$ avec $y_2' \in \mathsf{cPred}(X)$ et $\mathsf{Post}^{q,x}_{[t,t+\tau_2]} \subseteq \overline{\mathsf{uPred}(\overline{X})}$. Donc $y_2 \in \pi(X)$, ce qui achève la preuve de la proposition. □

3.5.4 Synthèse de stratégies gagnantes

Nous montrons maintenant qu'étant donné un jeu o-minimal $\mathcal{A}$ définissable dans une structure $\mathcal{M}$, nous pouvons construire une stratégie *définissable* (dans la même structure $\mathcal{M}$) à partir de l'ensemble des états gagnants. L'effectivité de cette construction sera discutée dans la sous-section 3.5.5.

Théorème 3.5.13 *Soit $\mathcal{A}$ un jeu o-minimal. Il existe une stratégie gagnante sans mémoire définissable pour tout $(q,y) \in \pi^*(\mathsf{But})$.*

Preuve. Par le théorème 3.5.5, la partition $\mathsf{Suf}(\mathcal{P}_\mathcal{A})$ est finie. Par la proposition 3.5.12, elle est stable par π, il existe donc $n \in \mathbb{N}$ tel que $\pi^*(\mathsf{But}) = \pi^n(\mathsf{But})$. Alors par la proposition 3.4.2, $\pi^*(\mathsf{But})$ est l'ensemble des états gagnants.

Soit $(q,y) \in \pi^*(\mathsf{But})$, nous savons qu'il existe une stratégie gagnante à partir de (q,y). Nous donnons maintenant une stratégie définissable. En suivant la preuve de la proposition 3.4.2, nous construisons cette stratégie par récurrence sur le nombre d'itérations de π. Soit $W = \cup_{0\leq i\leq k}\pi^i(\mathsf{But})$ et supposons que nous ayons déjà défini une stratégie sans mémoire sur toute pièce de W, considérons maintenant $\pi(W) \setminus W$.

Par la proposition 3.5.12, nous savons que $\pi(W) \setminus W$ est une union finie de pièces de $\mathsf{Suf}(\mathcal{P}_\mathcal{A})$. Soit P une de ces pièces. Par le théorème 3.5.5, P correspond à un nombre fini de mots finis sur $\mathcal{P}_\mathcal{A}$. Donc pour $(q,y) \in P$ nous avons que

$$\mathrm{Suf}_{\mathcal{P}_\mathcal{A}}(y) = \{\omega_1, ..., \omega_l\}$$

où les ω_i sont des mots finis sur $\mathcal{P}_\mathcal{A}$.

Soit $(x,t) \in V_1 \times V$ tel que $\gamma_q(x,t) = y$, nous avons que $\omega_{(x,t)} = \omega_i$ pour $\omega_i \in \mathrm{Suf}_{\mathcal{P}_\mathcal{A}}(y)$.

Comme $(q,y) \in \pi(W) \setminus W$, le mot $\omega_{(x,t)}$ doit contenir un préfixe dont la dernière lettre est inclue dans $\mathsf{cPred}(W)$ et sans occurrence de lettre de $\mathsf{uPred}(\overline{W})$ (ce qui a un sens car $\mathsf{cPred}(W)$ et $\mathsf{uPred}(\overline{W})$ sont des unions de pièces de $\mathcal{P}$). Soit A la dernière lettre de ce préfixe. Comme $A \subseteq \mathsf{cPred}(W)$, il existe une action contrôlable $c \in \Sigma_c$ telle que pour tout $(q,y) \in A$, une transition étiquetée par c est possible et toutes ces transitions amènent en W. La stratégie pour (q,x,t,y) sera d'exécuter l'action c après un certain délai. Nous expliquons maintenant comment choisir ce délai.

Soit (q, x, t, y) tel que $(q, y) \in P$ et $\gamma_q(x, t) = y$. Considérons $\mathsf{Temps}(x, t)$ le sous-ensemble de M^+ défini comme suit :

$$\mathsf{Temps}(x, t) = \{\tau \in M^+ \mid \gamma(x, t + \tau) \in A\}.$$

Cet ensemble est définissable car A est définissable.

Par o-minimalité, $\mathsf{Temps}(x, t)$ est une union finie de points et d'intervalles ouverts. Notons I le point ou l'intervalle le plus à gauche. Remarquons que I est définissable. Si I a un minimum m, nous définissons $\lambda(q, x, t, y) = (m, c)$. Sinon deux cas sont possibles. Si I est borné alors il est de la forme $]m, m'[$ ou $]m, m']$ dans ce cas nous définissons[11] $\lambda(q, x, t, y) = (\frac{1}{2}(m + m'), c)$. Enfin si I n'a pas de minimum et est non-borné, il est de la forme $]m, +\infty[$, nous définissons alors $\lambda(q, x, t, y) = (m + 1, c)$. Nous pouvons résumer la définition de λ sur P de la manière suivante :

$$\lambda(q, x, t, y) = \begin{cases} \big(\min(I), c\big) & \text{si } \varphi_1(x, t) \\ \big(\frac{1}{2}\big(\inf(I) + \sup(I)\big), c\big) & \text{si } \varphi_2(x, t) \\ \big(\inf(I) + 1, c\big) & \text{sinon} \end{cases}$$

où $\varphi_1(x, t)$ est une formule qui est vraie si, et seulement si I a un minimum et $\varphi_2(x, t)$ est une formule qui est vraie si, et seulement si I n'a pas de minimum et est borné. λ est donc clairement définissable sur P.

Comme il existe un nombre fini de $P \in \mathsf{Suf}(\mathcal{P}_\mathcal{A})$ par le théorème 3.5.5 et un nombre fini de suffixes pour chaque $P \in \mathsf{Suf}(\mathcal{P}_\mathcal{A})$, nous pouvons conclure que λ est définissable. □

Nous illustrons maintenant le théorème 3.5.13 sur deux exemples.

Exemple 3.5.14 *Considérons à nouveau l'automate de l'exemple 3.2.9. Nous définissons à partir de $\mathcal{A}$ un jeu o-minimal $\mathcal{A}_s$ relatif à l'exemple de la spirale (voir l'exemple 3.3.5). La structure o-minimale sous-jacente[12] $\mathcal{M}$ est $\langle \mathbb{R}, +, \cdot, 0, 1, <, \sin_{|[0,2\pi]}, \cos_{|[0,2\pi]} \rangle$. Le jeu o-minimal $\mathcal{A}_s$ a le même ensemble d'états, le même* But*, le même ensemble d'actions et le même automate fini sous-jacent $\mathcal{A}$ (c'est-à-dire que la figure 3.2(a) représente aussi $\mathcal{A}_s$). Les deux différences entre $\mathcal{A}$ et $\mathcal{A}_s$ sont les gardes et les dynamiques continues. Définissons tout d'abord les gardes : g_B peut être franchie sur les états de la pièce B (c'est-à-dire les points de la spirale) et g_C sur les états de la pièce C (les points différents de l'origine n'étant pas sur la spirale). La dynamique continue dans q_1 est celle décrite dans l'exemple 3.3.5 (les dynamiques continues dans q_2 et q_3 ne jouent aucun rôle). Clairement g_B, g_C et γ_{q_1} sont définissables dans $\mathcal{M}$.*

La stratégie au point $(0, 0)$ donnée par le théorème 3.5.13 est $\lambda(0, 0, \theta, t) = (\frac{\theta}{2}, c)$ où c est l'action qui mène à l'état gagnant q_2.

Exemple 3.5.15 *Pour la dynamique des automates temporisés (décrite dans l'exemple 3.2.2), les stratégies définissables correspondent aux stratégies réalisables obtenues dans [BCFL04].*

3.5.5 Résultats de décidabilité

Le théorème 3.5.13 est un résultat existentiel. Il assure qu'étant donné un jeu o-minimal, il existe une stratégie sans mémoire définissable pour tout $y \in \pi^*(\mathsf{But})$. La conclusion de la sous-

[11] Rappelons que tout groupe ordonné o-minimal est sans torsion et divisible (voir [PS86]), ce qui implique qu'il existe un unique y satisfaisant $y + y = m + m'$, que nous notons $\frac{1}{2}(m + m')$.

[12] Cette structure est o-minimale (voir [vdD96]).

section précédente est qu'étant donné un jeu o-minimal il existe une stratégie sans mémoire définissable pour tout $y \in \pi^*(\mathsf{But})$.

Le théorème 3.5.13 ne permet pas de déduire la décidabilité du problème de contrôle dans une $\mathcal{M}$-structure dans le cas général. En effet, cela dépend de la décidabilité de $\mathsf{Th}(\mathcal{M})$.

Théorème 3.5.16 *Soit $\mathcal{M}$ une structure o-minimale telle que $\mathsf{Th}(\mathcal{M})$ est décidable et $\mathcal{C}$ une classe de $\mathcal{M}$-jeux. Alors le problème de contrôle dans la classe $\mathcal{C}$ est décidable. De plus si $\mathcal{A} \in \mathcal{C}$, l'ensemble des états gagnants $\pi^*(\mathsf{But})$ est calculable et une stratégie gagnante sans mémoire définissable peut être calculée pour tout $(q, y) \in \pi^*(\mathsf{But})$.*

Preuve. Par le théorème 3.5.5, pour tout $\mathcal{A} \in \mathcal{C}$, $\mathsf{Suf}(\mathcal{P}_{\mathcal{A}})$ est une partition définissable respectant But ; la proposition 3.5.12 assure que cette partition est stable par π. Les hypothèses du corollaire 3.4.3 sont donc satisfaites et nous obtenons que le problème de contrôle dans la classe $\mathcal{C}$ est décidable et que les états gagnants et stratégies d'un jeu $\mathcal{A} \in \mathcal{C}$ sont calculables.
$\square$

Remarque 3.5.17 *Les structures $\langle \mathbb{R}, <, +, 0, 1 \rangle$ et $\langle \mathbb{R}, <, +, \cdot, 0, 1 \rangle$ sont des exemples de structures o-minimales avec une théorie décidable.*

Remarque 3.5.18 *En fait le théorème 3.5.16 peut être prouvé pour une classe plus large que les systèmes o-minimaux, la condition que toute variable est remise à zéro après chaque transition n'est pas nécessaire : il est suffisant qu'il existe une partition stable par suffixe et par* cPred *et* uPred *; si cette condition est satisfaite (et la dynamique de chaque état o-minimale) alors les remises à zéro peuvent être quelconques.*

Les automates temporisés entrent dans ce cadre. Le théorème 3.5.16 donne donc un moyen de calculer des stratégies gagnantes pour les jeux temporisés.

Remarque 3.5.19 *Notons que les résultats de décidabilité que nous avons obtenus pour les jeux o-minimaux reposent uniquement sur le calcul d'un attracteur basé sur une partition finie de l'espace d'états. Nous pouvons donc appliquer les algorithmes classiques pour les problèmes de sûreté, ou des conditions de victoire* à la Büchi. *De manière plus générale nous pouvons résoudre les jeux o-minimaux pour des conditions de parité, voir [GThW02] pour plus de détails sur ces jeux.*

Remarque 3.5.20 *Remarquons que nous n'avons pas traité de façon spécifique les comportements zénon. En effet, dans le cadre considéré, si l'environnement a une stratégie qui empêche le jeu d'atteindre* But *en bloquant le temps, nous considérons que le contrôleur perd le jeu. Dans le contexte des automates temporisés, une solution élégante a été proposée dans [AFH+03]. A cause de la forte condition sur les remises à zéro, cette méthode ne peut être facilement appliquée aux jeux o-minimaux.*

3.6 Contrôle hybride avec observation partielle de la dynamique

Jusqu'à maintenant, nous avons supposé qu'à partir d'un point donné, l'environnement choisit une trajectoire continue suivie par le jeu, et que le contrôleur réagit en fonction de la trajectoire choisie. Dans cette section, nous considérons une observation partielle de la dynamique : la trajectoire n'est pas connue par le contrôleur, et sa stratégie ne peut dépendre que du point de départ. Ce cadre permet notamment de modéliser naturellement des horloges où les pentes des horloges appartiennent à un petit intervalle autour de 1 [Pur98, ALM05].

Notons que l'observation partielle que nous considérons ne concerne que la dynamique du système et non pas les actions effectuées, ce qui est une hypothèse différente de celle faite dans le cadre des systèmes finis dans [AVW03] ou des systèmes temporisés dans [BDMP03].

3.6.1 Stratégies sous observation partielle

Dans cette sous-section, nous adaptons toutes les notions que nous avons déjà définies au cadre de l'observation partielle.

Soit $\rho = (q_0, x_0, t_0, y_0) \xrightarrow{\tau_1, a_1} \cdots \xrightarrow{\tau_n, a_n} (q_n, x_n, t_n, y_n)$ une exécution finie. L'*observation* de ρ, notée $\mathrm{obs}(\rho)$, est la suite $(q_0, y_0) \xrightarrow{\tau_1, a_1} \cdots \xrightarrow{\tau_n, a_n} (q_n, y_n)$.

Définition 3.6.1 (Stratégie sous observation partielle) *Une stratégie λ est dite* sous observation partielle *si pour toutes les exécutions finies ρ, ρ', si $obs(\rho) = obs(\rho')$ alors $\lambda(\rho) = \lambda(\rho')$.*

Toutes les autres notions, comme les stratégies sans mémoire, compatibilité, stratégies gagnantes, états gagnants, *etc...* s'étendent naturellement à ce nouveau contexte.

Problème 2 (Problème de contrôle sous observation partielle) *Étant donné un $\mathcal{M}$-jeu $\mathcal{A} \in \mathcal{C}$, et un état initial définissable (q, y), déterminer s'il existe une stratégie gagnante sous observation partielle dans $\mathcal{A}$ à partir de (q, y).*

Exemple 3.6.2 *Nous considérons à nouveau l'exemple de la spirale (exemple 3.5.14). Rappelons que le comportement discret du système est représenté sur la figure 3.2(a) et sa dynamique définie dans l'exemple 3.3.5.*

Nous avons montré que sous observation totale ce $\mathcal{M}$-jeu a une stratégie gagnante en $(0,0)$ donnée par $\lambda(0,0,\theta,t) = (\frac{\theta}{2}, c)$. Notons que cette stratégie dépend de la trajectoire prise (car elle dépend de θ).

Il n'existe pas de stratégie gagnante sous observation partielle pour ce jeu : une telle stratégie ne peut dépendre que du point courant, et dans cet exemple, pour toute action (τ, a) proposée par le contrôleur en $(0,0)$, il existe une trajectoire qui atteint un mauvais *état (c'est-à-dire un point sur la spirale) avant τ.*

3.6.2 Un nouveau codage symbolique des trajectoires : les supermots

Dans la section 3.4, nous avons prouvé que la partition suffixe était un bon outil pour étudier le problème de contrôle (le problème 1). Cependant, dans le cas du contrôle sous observation partielle (le problème 2), la partition suffixe n'est plus un bon outil, comme illustré par l'exemple 3.6.3. Pour résoudre le problème 2, nous introduirons donc un autre codage symbolique des trajectoires.

Exemple 3.6.3 *Nous considérons le système dynamique $(\mathcal{M}, \gamma)$ où $\mathcal{M} = \langle \mathbb{R}, +, \cdot, 0, 1, < \rangle$ et $\gamma : \mathbb{R}^2_{\geq 0} \times [1,2]^2 \times \mathbb{R}_{\geq 0} \to \mathbb{R}^2_{\geq 0}$ est définie par*

$$\gamma(x_1, x_2, p_1, p_2, t) = \begin{cases} (x_1 + p_1 \cdot t, x_2 + p_2 \cdot t) & \textit{si } x_1 \leq 2 \\ (x_1 + t, x_2 + 2t) & \textit{sinon} \end{cases}$$

Nous associons à ce système dynamique la partition $\mathcal{P} = \{A, B\}$ où $B = [3,5] \times [0, +\infty[$ et $A = \mathbb{R}^2 \setminus B$ (voir la figure 3.7). Considérons les types dynamiques suffixes des points $y_1 = (0,0)$ et $y_2 = (8,0)$. Nous avons que $Suf_{\mathcal{P}}(y_1) = Suf_{\mathcal{P}}(y_2) = \{ABA\}$.

À partir de ce système dynamique, nous construisons un $\mathcal{M}$-jeu simple où la pièce B est la garde d'une transition contrôlable (étiquetée a) menant à But*. Dans ce jeu, il est facile de voir que y_2 est un état gagnant alors que y_1 ne l'est pas. En effet, une stratégie gagnante à partir de y_2 est d'attendre 2 unités de temps et d'effectuer alors l'action a. À partir de y_1, nous savons qu'après τ unités de temps, le système sera dans le carré $[\tau, 2\tau]^2$. Pour tout $\tau \in \mathbb{R}_{\geq 0}$, nous avons que $[\tau, 2\tau]^2 \nsubseteq B$ (comme décrit sur la figure 3.7). En particulier, à partir de y_1 il est impossible de garantir que la transition sera possible après τ unités de temps et donc il n'existe pas de stratégie gagnante (sous observation partielle) à partir de y_1. Cela montre que la partition suffixe n'est pas un bon outil pour étudier la problème de contrôle sous observation partielle.*

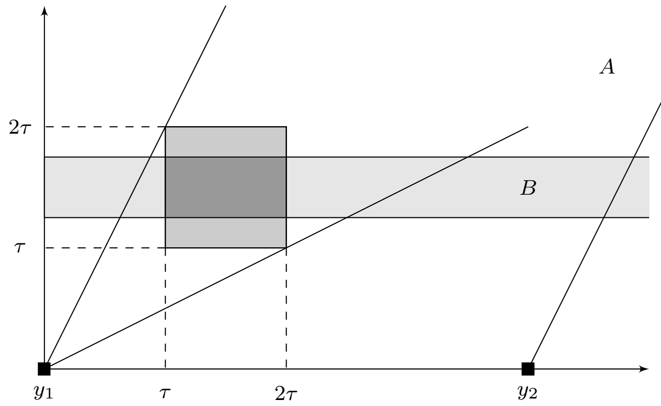


FIG. 3.7 – La partition suffixe ne respecte pas les états gagnants sous observation partielle

Supermots.

Nous décrivons tout d'abord un nouveau codage symbolique des trajectoires, nous discuterons de sa définissabilité (dans le cas o-minimal) par la suite. Soit $(\mathcal{M}, \gamma)$ un système dynamique, y un point de V_2 et $\mathcal{P}$ une partition de V_2. Comme plusieurs trajectoires passent par le point y, il existe plusieurs y' tels que $y \xrightarrow{\tau} y'$, pour $\tau \in M^+$. En particulier, en partant de y, il est possible d'être dans plusieurs pièces de $\mathcal{P}$ après τ unités de temps. Pour capturer ce comportement, nous associons un mot ω_y sur $2^{\mathcal{P}}$ à tout point $y \in V_2$. Pour définir le mot sur $2^{\mathcal{P}}$ associé à $y \in V_2$, nous devons introduire quelques définitions supplémentaires.

Définition 3.6.4 *Soit y un point de V_2 et τ un temps de M^+.*

$$\mathcal{F}_y(\tau) = \{P \in \mathcal{P} \mid \exists x \in M^{k_1}\ \exists t \in M\ \gamma(x,t) = y \text{ et } \gamma(x, t+\tau) \in P\}.$$

L'ensemble $\mathcal{F}_y(\tau)$ représente l'ensemble des pièces qui sont potentiellement atteintes après τ unités de temps à partir de y.

Définition 3.6.5 *Soit y un point de V_2.*

$$\mathcal{F}_y = \big\{ I \mid I \text{ est un intervalle de temps maximal pour la propriété } \exists S \in 2^{\mathcal{P}}\ \forall \tau \in I \ \ \mathcal{F}_y(\tau) = S \big\}$$

Pour tout $y \in V_2$, l'ensemble $\mathcal{F}_y$ est exactement l'ensemble des composantes connexes des ensembles $\{\tau \in M^+ \mid \mathcal{F}_y(\tau) = S\}$, pour $S \in 2^{\mathcal{P}}$.

Nous pouvons maintenant définir le mot ω_y associé à un point $y \in V_2$.

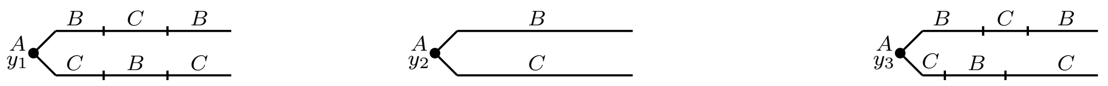
Définition 3.6.6 *Soit $(\mathcal{M}, \gamma)$ un système dynamique, y un point de V_2, et $\mathcal{P}$ une partition de V_2.* Le supermot associé à *y est la fonction $\mathsf{Sup}_{\mathcal{P}}(y) : \mathcal{F}_y \to 2^{\mathcal{P}}$ définie par :*

$$\mathsf{Sup}_{\mathcal{P}}(y)(I) = S \qquad \text{où } I \in \mathcal{F}_y \text{ est tel que } \forall \tau \in I \ \ \mathcal{F}_y(\tau) = S.$$

Si $(\mathcal{M}, \gamma)$ est un système dynamique et $\mathcal{P}$ une partition finie de V_2, nous notons $\mathsf{Sup}(\mathcal{P})$ la partition induite par les supermots.

Nous illustrons cette nouvelle notion sur un exemple.

Exemple 3.6.7 *Considérons les trois systèmes dynamiques représentés sur la figure 3.8. Dans les trois cas, le système dynamique consiste en deux trajectoires à partir du point y_i. La différence entre les trois systèmes réside dans la façon dont la partition $\mathcal{P} = \{A, B, C\}$ est franchie. Les supermots associés aux y_i sont les suivants : pour les deux premiers systèmes dynamiques nous avons $\mathsf{Sup}_{\mathcal{P}}(y_1) = \mathsf{Sup}_{\mathcal{P}}(y_2) = \{A\}\{B, C\}$, et pour le dernier $\mathsf{Sup}_{\mathcal{P}}(y_3) = \{A\}\{B, C\}\{B\}\{B, C\}\{C\}\{B, C\}$.*



(a) $\{A\}\{B,C\}$ (b) $\{A\}\{B,C\}$ (c) $\{A\}\{B,C\}\{B\}\{B,C\}\{C\}\{B,C\}$

FIG. 3.8 – Les suffixes et les supermots ne sont pas comparables

Remarquons que les notions de type dynamique suffixe et supermot sont incomparables. Nous avons que $\mathsf{Sup}_{\mathcal{P}}(y_1) = \mathsf{Sup}_{\mathcal{P}}(y_2) \neq \mathsf{Sup}_{\mathcal{P}}(y_3)$. Considérons les types dynamiques suffixes de ces points :

$$\mathsf{Suf}(y_1) = \{ABCB, ACBC\} \ ; \ \mathsf{Suf}(y_2) = \{AB, AC\} \ ; \ \mathsf{Suf}(y_3) = \{ABCB, ACBC\}.$$

Les supermots peuvent donc distinguer y_1 et y_3, mais pas y_1 et y_2, alors que les types dynamiques suffixes peuvent distinguer y_1 et y_2, mais pas y_1 et y_3.

Définissabilité.

Soit $(\mathcal{M}, \gamma)$ un système dynamique et $\mathcal{P}$ une partition finie de V_2. Nous avons expliqué comment associer un mot sur $2^{\mathcal{P}}$ à un point y de V_2. Nous expliquons maintenant pourquoi dans le cas des systèmes dynamiques o-minimaux le codage par supermots peut être réalisé de façon définissable.

Soit $(\mathcal{M},\gamma)$ un système dynamique o-minimal et $\mathcal{P}$ une partition finie et définissable de V_2. Remarquons tout d'abord que pour un sous-ensemble $S \in 2^{\mathcal{P}}$, nous pouvons facilement écrire une formule du premier ordre $\varphi_S(y,\tau)$ qui est vraie si, et seulement si $\mathcal{F}_y(\tau) = S$.

En effet si, $S = \{A_1, \cdots, A_n\}$, il suffit de poser :

$$\begin{aligned}\varphi_S(y,\tau) \ &= \exists x_1\ \exists t_1\ \cdots\ \exists x_n\ \exists t_n \bigwedge_{1\leq i\leq n} \big(\gamma(x_i,t_i) = y\ \wedge\ \gamma(x_i,t_i+\tau) \in A_i\big) \\ &\qquad \wedge\ \forall x\ \forall t\ \big(\gamma(x,t) = y\big) \Rightarrow \big(\gamma(x,t+\tau) \in A_1 \cup \cdots \cup A_n\big).\end{aligned}$$

Or pour tout $y \in V_2$, l'ensemble $\mathcal{F}_y$ est composé exactement des composantes connexes des ensembles $\{\tau \in M^+ \mid \varphi_S(y,\tau)\}$, pour $S \in 2^{\mathcal{P}}$. La technique pour prouver que ces ensembles sont définissables est la même utilisée pour montrer que $\mathcal{F}_x$ est définissable (voir [BM05, Bri06]).

Résultats de finitude.

Nous montrons maintenant que dans le cadre des systèmes dynamiques o-minimaux, seul un nombre fini de supermots finis est nécessaire pour encoder l'ensemble des trajectoires.

Proposition 3.6.8 *Soit $(\mathcal{M},\gamma)$ un système dynamique o-minimal et $\mathcal{P}$ une partition finie et définissable de V_2. Alors il existe un nombre fini de supermots finis associés à $(\mathcal{M},\gamma)$ par rapport à $\mathcal{P}$.*

Preuve. Soit $S \in 2^{\mathcal{P}}$, considérons tout d'abord l'ensemble

$$\mathcal{F}_y(S) = \big\{\tau \in M^+ \mid \mathcal{F}_y(\tau) = S\big\} = \big\{\tau \in M^+ \mid \varphi_S(y,\tau)\big\}.$$

Comme nous l'avons remarqué, l'ensemble $\mathcal{F}_y(S)$ est un sous-ensemble définissable de M. Par o-minimalité, c'est donc une union finie de points et d'intervalles ouverts, qui a donc nécessairement un nombre fini de composantes connexes. Par définition de $\mathcal{F}_y$ nous avons l'égalité suivante :

$$|\mathcal{F}_y| = \sum_{S\in 2^{\mathcal{P}}} \Big(\text{nombre de composantes connexes de } \mathcal{F}_y(S)\Big).$$

Comme $\mathcal{P}$ est finie, nous déduisons que $\mathcal{F}_y$ est fini.

En utilisant la finitude uniforme (voir le théorème 3.5.3), nous obtenons qu'il existe $N \in \mathbb{N}$ tel que pour tout $y \in V_2$, $\big|\mathcal{F}_y\big| \leq N$. En terme de codage de mots, cela signifie qu'il y a uniquement un nombre fini de supermots associés aux points de V_2. En effet la taille des supermots $\mathsf{Sup}_{\mathcal{P}}(y)$ est uniformément bornée par N. □

La proposition précédente montre directement que la partition $\mathsf{Sup}(\mathcal{P})$ est finie. De plus cette partition est définissable comme le précise la proposition suivante.

Proposition 3.6.9 *Soit $(\mathcal{M},\gamma)$ un système dynamique o-minimal, et $\mathcal{P}$ une partition finie et définissable de V_2. La partition $\mathsf{Sup}(\mathcal{P})$ est finie et définissable.*

Preuve. La finitude de $\mathsf{Sup}(\mathcal{P})$ est une conséquence de la proposition 3.6.8. La définissabilité de $\mathsf{Sup}(\mathcal{P})$ provient de la définissabilité des $\mathcal{F}_y$ et suit la preuve du lemme 12.2.6 de [Bri06]. □

3.6.3 Prédécesseurs contrôlables

Soit $\mathcal{A} = (\mathcal{M}, Q, \mathsf{But}, \Sigma, \delta, \gamma)$ un $\mathcal{M}$-jeu. Remarquons que les opérateurs définis dans la sous-section 3.4.1 ne permettent pas de calculer l'ensemble des états gagnants. En effet, sous observation partielle le choix de la stratégie doit être indépendant de la trajectoire suivie. Nous définissons donc de nouveaux prédécesseurs contrôlables qui prennent en compte cette spécificité ; nous expliquerons dans la remarque 3.6.10 la différence entre les anciens et les nouveaux opérateurs.

Pour $A \subseteq Q \times V_2$, $a \in \Sigma_c$ et $u \in \Sigma_u$ nous devons raffiner la notion de prédécesseurs contrôlables discrets. Pour tout $a \in \Sigma_c \cup \Sigma_u$, nous définissons

$$a\mathsf{Pred}(A) = \left\{ (q,y) \in Q \times V_2 \;\middle|\; \begin{array}{l} a \text{ est possible dans } (q,y), \\ \text{et } \forall (q',y') \in Q \times V_2, \\ \Big((q,y) \xrightarrow{a} (q',y') \Rightarrow (q',y') \in A\Big) \end{array} \right\}.$$

Notons que les opérateurs cPred et uPred, définis page 36, correspondent à $\mathsf{cPred}(A) = \bigcup_{a\in\Sigma_c} a\mathsf{Pred}(A)$, et $\mathsf{uPred}(A) = \bigcup_{u\in\Sigma_u} u\mathsf{Pred}(A)$.

La notion de prédécesseur temporel sûr doit être adaptée pour prendre en compte le fait que la stratégie du contrôleur ne peut dépendre de la trajectoire. Nous définissons le *prédécesseur temporel sûr uniforme* d'un ensemble A par rapport à un ensemble B : un état (q,y) appartient à $\widetilde{\mathsf{Pred}}_t(A,B)$ s'il existe un délai τ tel que en laissant s'écouler τ unités de temps, un état de A est atteint, en évitant B. Formellement l'opérateur $\widetilde{\mathsf{Pred}}_t$ est défini par :

$$\widetilde{\mathsf{Pred}}_t(A,B) = \left\{ (q,y) \in Q \times V_2 \;\middle|\; \begin{array}{l} \exists \tau \in M^+,\ \forall (x,t) \in V_1 \times V \text{ tel que } \gamma_q(x,t) = y, \\ (q,y) \xrightarrow{\tau}_{x,t} (q',y') \text{ implique } (q',y') \in A \\ \text{et } \mathsf{Post}^{q,x}_{[t,t+\tau]} \subseteq \overline{B} \end{array} \right\}.$$

L'opérateur des *prédécesseurs contrôlables* est alors défini par :

$$\widetilde{\pi}(A) = A \cup \bigcup_{a\in\Sigma_c} \widetilde{\mathsf{Pred}}_t(a\mathsf{Pred}(A), \mathsf{uPred}(\overline{A})).$$

Remarque 3.6.10 *Considérons les différences entre cet opérateur $\widetilde{\pi}$ et l'opérateur π défini page 36 :*

$$\pi(A) = A \cup \mathsf{Pred}_t\left(\bigcup_{a\in\Sigma_c} a\mathsf{Pred}(A), \bigcup_{u\in\Sigma_u} u\mathsf{Pred}(\overline{A})\right)$$

et

$$\widetilde{\pi}(A) = A \cup \bigcup_{a\in\Sigma_c} \widetilde{\mathsf{Pred}}_t\left(a\mathsf{Pred}(A), \bigcup_{u\in\Sigma_u} u\mathsf{Pred}(\overline{A})\right).$$

L'opérateur Pred_t a été remplacé par $\widetilde{\mathsf{Pred}}_t$, et la position de l'union sur les actions contrôlables a également changé. Ce changement exprime que sous observation partielle, l'action contrôlable qui doit être effectuée doit être choisie indépendamment de la trajectoire. L'exemple 3.6.11 illustre cette différence.

Exemple 3.6.11 *Considérons le $\mathcal{M}$-jeu $\mathcal{A}$ représenté sur la figure 3.9(a) où $\mathsf{But} = \{q_2, q_3\}$ et $c_1, c_2 \in \Sigma_c$ sont des actions contrôlables distinctes. La dynamique dans q_1 est décrite sur la* *figure 3.9(b). Il y a une stratégie gagnante (sous observation totale) dans $\mathcal{A}$ à partir de y ; en effet, selon la trajectoire suivie, le contrôleur va jouer soit (τ, c_1), soit (τ, c_2), pour un certain $\tau \in \mathbb{R}_{\geq 0}$. Cependant il n'y a pas de stratégie gagnante sous observation partielle à partir de y. Même s'il existe $\tau \in \mathbb{R}_{\geq 0}$ tel qu'une action contrôlable gagnante sera possible après τ unités de temps, il est impossible de dire laquelle de ces actions doit être prise* a priori.

(a) Le $\mathcal{M}$-jeu $\mathcal{A}$ (b) Dynamique dans q_1

FIG. 3.9 – Jeu gagnant sous observation totale mais pas sous observation partielle

La proposition suivante établit la correction du nouvel opérateur de prédécesseurs contrôlables pour calculer l'ensemble des états gagnants sous observation partielle.

Proposition 3.6.12 *Soit $\mathcal{A} = (\mathcal{M}, Q, \mathsf{But}, \Sigma, \delta, \gamma)$ un $\mathcal{M}$-jeu. S'il existe $n \in \mathbb{N}$ tel que $\widetilde{\pi}^n(\mathsf{But}) = \widetilde{\pi}^{n+1}(\mathsf{But})$ alors $\widetilde{\pi}^*(\mathsf{But}) = \widetilde{\pi}^n(\mathsf{But})$ est l'ensemble des états gagnants de $\mathcal{A}$.*

Preuve. Nous prouvons tout d'abord que si $(q, y) \in \widetilde{\pi}^*(\mathsf{But})$ alors il existe une stratégie gagnante sous observation partielle à partir de (q, y). Nous définissons une stratégie gagnante sans mémoire à partir de tout $(q, y) \in \widetilde{\pi}^*(\mathsf{But})$. Par abus de notation, nous définissons la stratégie λ sur les états (q, y) plutôt que sur les exécutions, ce qui est possible car elle est sans mémoire.

Nous définissons λ sur les ensembles $\bigcup_{0 \leq i \leq k} \widetilde{\pi}^i(\mathsf{But})$ par récurrence sur k, et montrons que c'est une stratégie gagnante. Si $k = 0$, nous posons λ non-définie, c'est donc une stratégie gagnante pour tout état de But.

Supposons maintenant λ déjà définie sur $W = \bigcup_{0 \leq i \leq k} \widetilde{\pi}^i(\mathsf{But})$ et gagnante sur ces états. Nous définissons maintenant λ sur $\widetilde{\pi}(W)$. Soit $(q, y) \in Q \times V_2$: si $(q, y) \in W$, λ est déjà définie ; si $(q, y) \in \widetilde{\pi}(W) \backslash W$, nous savons qu'il existe $a \in \Sigma_c$ avec $(q, y) \in \widetilde{\mathsf{Pred}}_t\left(a\mathsf{Pred}(W), \mathsf{uPred}(\overline{W})\right)$. Il existe $\tau \in M^+$ avec (τ, a) possible[13] dans (q, y) tel que pour tout (x, t) si $\gamma_q(x, t) = y$, alors $(q, y) \xrightarrow{\tau, a}_{x,t} (q', y')$ avec $(q', y') \in W$ et $\mathsf{Post}^{q,x}_{[t,t+\tau]} \subseteq \overline{\mathsf{uPred}(\overline{W})}$. Nous posons $\lambda(q, y) = (\tau, a)$ et montrons que c'est un choix gagnant.

Nous montrons par récurrence sur k que λ est gagnante pour tout état de $W = \bigcup_{0 \leq i \leq k} \widetilde{\pi}^i(\mathsf{But})$. C'est immédiat pour $k = 0$. Supposons le résultat vrai pour k, et soit $(q, y) \in \widetilde{\pi}(W)$. Soit $\rho = (q, x, t, y) \xrightarrow{\tau_1, a_1} (q_1, x_1, t_1, y_1) \xrightarrow{\tau_2, a_2} \cdots$ une exécution compatible avec λ. Nous avons que soit $\tau_1 = \tau$ et $a_1 = a$, dans ce cas $(q_1, y_1) \in W$, soit $\tau_1 \leq \tau$ et $a_1 \in \Sigma_u$, dans ce cas $(q, y) \xrightarrow{\tau_1}_{x,t} (q', y') \xrightarrow{a_1} (q_1, y_1)$ avec $(q', y') \notin \mathsf{uPred}(\overline{W})$ donc $(q_1, y_1) \in W$. Dans les deux cas $(q_1, y_1) \in W$ donc par hypothèse de récurrence ρ est gagnant.

Nous montrons maintenant que s'il existe une stratégie gagnante sous observation partielle λ à partir de (q, y) alors $(q, y) \in \widetilde{\pi}^*(\mathsf{But})$. Posons $W = \widetilde{\pi}^*(\mathsf{But})$, et supposons par l'absurde que

[13] Nous disons que $(\tau, a) \in M^+ \times \Sigma$ est possible dans (q, y) s'il existe $(x, t) \in V_1 \times V$ tel que $\gamma(x, t) = y$ et (τ, a) est possible dans (q, x, t, y).

$(q, y) \notin W$, nous construisons une exécution non-gagnante compatible avec λ. Par hypothèse $\widetilde{\pi}(W) = W$ donc $(q, y) \notin \widetilde{\pi}(W)$, il vient que pour tout $a \in \Sigma_c$, pour tout $\tau \in M^+$ il existe $(x, t) \in V_1 \times V$ tel que $\gamma_q(x, t) = y$, et $(q, y) \to^{\tau}_{x,t} (q', y')$ implique $(q', y') \notin a\mathsf{Pred}(W)$ ou $\mathsf{Post}^{q,x}_{[t,t+\tau]} \cap \mathsf{uPred}(\overline{W}) \neq \emptyset$. Soit $(\tau, a) = \lambda(q, y)$ (comme λ est une stratégie sous observation partielle, elle ne dépend pas de x et t) et soit $(x, t) \in V_1 \times M^+$ comme définis précédemment.

Il existe (q_1, x_1, t_1, y_1) avec $(q_1, y_1) \notin W$ tel que soit $(q, x, t, y) \xrightarrow{\tau,a} (q_1, x_1, t_1, y_1)$, soit il existe $\tau' \leq \tau$ et $u \in \Sigma_u$ avec $(q, x, t, y) \xrightarrow{\tau',u} (q_1, x_1, t_1, y_1)$. Dans les deux cas, l'exécution construite est compatible avec λ. Comme $(q_1, y_1) \notin W$ nous pouvons appliquer à nouveau cet argument et construire par récurrence une exécution $\rho = (q, x, t, y) \xrightarrow{\tau_1,a_1} (q_1, x_1, t_1, y_1) \xrightarrow{\tau_2,a_2} \cdots$ compatible avec λ et telle que pour tout i, $(q_i, x_i, t_i, y_i) \notin W$. Par définition de W, pour tout i, $q_i \notin \mathsf{But}$, ce qui contredit que λ est une stratégie gagnante. $\square$

3.6.4 Résolution du contrôle o-minimal sous observation partielle

La proposition suivante est l'équivalent de la proposition 3.5.12 sous l'hypothèse d'observation partielle.

Proposition 3.6.13 *Soit $\mathcal{A}$ un jeu o-minimal, et $\mathcal{P}_\mathcal{A}$ la partition correspondant à ses gardes et remises à zéro (voir page 40). La partition des supermots $\mathsf{Sup}(\mathcal{P}_\mathcal{A})$ est finie et stable par $\widetilde{\pi}$.*

Preuve. Nous fixons un état discret q de l'automate et prenons $y_1, y_2 \in V_2$ ayant le même supermot, c'est-à-dire $\mathsf{Sup}_\mathcal{P}(y_1) = \mathsf{Sup}_\mathcal{P}(y_2)$; en particulier (q, y_1) et (q, y_2) appartiennent à la même pièce de $\mathsf{Sup}\,(\mathcal{P}_\mathcal{A})$. Soit $X \in \mathsf{Sup}\,(\mathcal{P}_\mathcal{A})$, nous montrons que $(q, y_1) \in \pi(X)$ si, et seulement si $(q, y_2) \in \pi(X)$. Nous montrons uniquement que si $(q, y_1) \in \pi(X)$ alors $(q, y_2) \in \pi(X)$, l'autre sens est identique.

Rappelons la définition de l'opérateur des prédécesseurs contrôlables :

$$\widetilde{\pi}(X) = X \cup \bigcup_{a \in \Sigma_c} \widetilde{\mathsf{Pred}}_t(a\mathsf{Pred}(X), \mathsf{uPred}(\overline{X})).$$

Considérons tout d'abord le cas simple où $(q, y_1) \in X$, comme $\mathsf{Sup}_{\mathcal{P}_\mathcal{A}}(y_1) = \mathsf{Sup}_{\mathcal{P}_\mathcal{A}}(y_2)$ et $X \in \mathsf{Sup}\,(\mathcal{P}_\mathcal{A})$, nous avons que $(q, y_2) \in X$ et donc $(q, y_2) \in \widetilde{\pi}(X)$.

Considérons maintenant le cas où $(q, y_1) \in \widetilde{\pi}(X) \setminus X$. Alors il existe $a \in \Sigma_c$ et $t'_1 \in M^+$ tels que pour tout $(x_1, t_1) \in V_1 \times M^+$, si $\gamma_q(x_1, t_1) = y_1$ et $y_1 \xrightarrow{\tau_1}_{x_1,t_1} y'_1$, alors $y'_1 \in a\mathsf{Pred}(X)$ et $\mathsf{Post}^{q,x}_{[t_1,t_1+\tau_1]} \subseteq \overline{\mathsf{uPred}(\overline{X})}$. Par la définition des transitions dans un jeu o-minimal nous avons que $a\mathsf{Pred}(X)$ est une union de pièces de $\mathcal{P}_\mathcal{A}$, de même pour $\mathsf{uPred}(\overline{X})$.

La condition précédente signifie qu'il existe un préfixe de $\mathsf{Sup}_\mathcal{P}(y)$ sans occurrence de lettres de $\mathsf{uPred}(\overline{X})$ et tel que tout élément de sa dernière lettre est un sous-ensemble de $a\mathsf{Pred}(X)$ (cela a un sens car $a\mathsf{Pred}(X)$ et $\mathsf{uPred}(\overline{X})$ sont des unions de pièces de $\mathcal{P}_\mathcal{A}$). Appelons $\omega^p_{y_1}$ ce préfixe. Supposons que $\omega^p_{y_1} = S_1 S_2 \cdots S_K$, avec $S_i \in 2^{\mathcal{P}_\mathcal{A}}$, les conditions précédentes peuvent être exprimées comme suit :

- pour tout $A \in S_k$, $A \subseteq a\mathsf{Pred}(X)$,
- pour tout $1 \leq i \leq k$, pour tout $A \in S_i$, $A \cap \mathsf{uPred}(\overline{X}) = \emptyset$.

Comme $\mathsf{Sup}_{\mathcal{P}_\mathcal{A}}(y_1) = \mathsf{Sup}_{\mathcal{P}_\mathcal{A}}(y_2)$, nous pouvons trouver $\tau_2 \in M^+$ tel que pour tout $(x_2, t_2) \in V_1 \times V$ tel que $\gamma_q(x_2, t_2) = y_2$, $y_2 \xrightarrow{\tau_2}_{x_2,t_2} y'_2$ implique $y'_2 \in a\mathsf{Pred}(X)$ et $\forall 0 \leq$

$\tau' \leq \tau_2\ \gamma_q(x, t+\tau') \notin \mathsf{uPred}(\overline{X})$. Donc $(q, y_2) \in \widetilde{\pi}(X)$, ce qui achève la preuve de la proposition. $\square$

Nous avons donc montré le point essentiel de la résolution des jeux o-minimaux sous observation partielle : la partition des supermots est finie et stable par $\widetilde{\pi}$. Nous pouvons alors appliquer les techniques de la section 3.5 pour obtenir le résultat suivant.

Théorème 3.6.14 *Soit $\mathcal{M}$ une structure o-minimale telle que $\mathsf{Th}(\mathcal{M})$ est décidable et $\mathcal{C}$ une classe de $\mathcal{M}$-jeux. Alors le problème de contrôle sous observation partielle dans la classe $\mathcal{C}$ est décidable. De plus si $\mathcal{A} \in \mathcal{C}$, l'ensemble des états gagnants $\pi^*(\mathsf{But})$ est calculable et une stratégie gagnante sans mémoire définissable peut être effectivement calculée pour tout $(q, y) \in \pi^*(\mathsf{But})$.*

Chapitre 4

Systèmes o-minimaux pondérés

Pour exprimer des propriétés quantitatives sur les systèmes temporisés, le modèle des automates temporisés a été étendu avec des fonctions de coût. Dans ce cadre, le problème d'atteignabilité optimale est décidable [ALP01, BFH$^+$01, BBBR06]. Pour des propriétés plus riches, comme celles exprimées par une formule de WCTL, une extension pondérée de CTL avec des modalités sur les coûts, le problème de *model-checking* devient indécidable [BBR04]. De même, dans le cadre du contrôle, le problème de contrôle optimal (déterminer le meilleur coût possible pour atteindre un état, quoi que fasse l'environnement) est indécidable dans le modèle des automates temporisés pondérés [BBR05, BBM06].

Dans ce chapitre, nous adoptons une démarche similaire et enrichissons la classe des systèmes o-minimaux avec des fonctions de coût. Un système o-minimal pondéré est un système hybride, non nécessairement o-minimal, en particulier il n'admet pas nécessairement de bisimulation finie à temps abstrait (voir la remarque 4.1.7) ; nous développons donc des techniques spécifiques aux systèmes o-minimaux pondérés.

Nous considérons tout d'abord le problème de contrôle optimal. Dans le chapitre 3, nous avions utilisé la partition suffixe pour résoudre le contrôle optimal pour les systèmes o-minimaux classiques ; avec l'ajout de fonctions de coût, cette outil n'est plus adapté (deux états ayant les mêmes suffixes peuvent être l'un gagnant et l'autre perdant pour un objectif de coût). Nous tirons parti des fortes conditions sur les transitions discrètes des systèmes o-minimaux pour borner *a priori* la durée du jeu (en nombre de transitions franchies) et utilisons des techniques de définissabilité pour montrer que pour une structure décidable et o-minimale, le problème de contrôle optimal est décidable pour les systèmes o-minimaux pondérés.

Nous considérons ensuite le problème de *model-checking* de WCTL. Pour une formule φ de WCTL, nous construisons inductivement une partition de l'espace d'états respectant φ (c'est-à-dire que la valeur de vérité de φ sur une pièce est uniforme), et nous montrons que cette partition est finie et définissable (c'est-à-dire qu'il existe une formule décrivant l'ensemble des états d'une pièce). La construction d'une telle partition permet de déduire la décidabilité du *model-checking* de WCTL si la structure sous-jacente est décidable.

4.1 Automates et jeux o-minimaux pondérés

Dans cette section, nous définissons les automates et jeux o-minimaux pondérés, qui étendent les modèles du chapitre 3 avec des fonctions de coût. Ces fonctions de coût donnent une information quantitative et permettent de mesurer les performances des systèmes. Ces modèles sont respectivement inspirés des automates temporisés pondérés [ALP01, BFH⁺01] et des jeux temporisés pondérés [ABM04, BCFL04].

4.1.1 Définitions

Nous considérons des fonctions de coût **positives** et **croissantes**. Remarquons que les fonctions de coût pour les automates temporisés pondérés [ALP01, BFH⁺01] satisfont ces hypothèses.

Soit $\mathcal{M}$ une structure, *une fonction de coût sur $\mathcal{M}$ positive et croissante* est une fonction $\mathsf{Coût} : Q \times V_1 \times V \times M^+ \to M^+$ définissable dans $\mathcal{M}$ telle que pour tout $q \in Q$, $x \in V_1$, $t \in V$ et $\tau_1,\ \tau_2 \in M^+$, si $\tau_1 \leq \tau_2$ alors $\mathsf{Coût}(q, x, t, \tau_1) \leq \mathsf{Coût}(q, x, t, \tau_2)$.

Définition 4.1.1 (Automates et jeux o-minimaux pondérés)
Un $\mathcal{M}$-automate (resp. $\mathcal{M}$-jeu) pondéré *est un $\mathcal{M}$-automate (resp. $\mathcal{M}$-jeu) o-minimal muni d'une fonction de coût sur $\mathcal{M}$ positive et croissante.*

La sémantique d'un automate (resp. jeu) pondéré $(\mathcal{H}, \mathsf{Coût})$ est celle de l'automate o-minimal sous-jacent. La fonction de coût ne modifie donc pas le comportement de l'automate ; elle associe à tout mouvement de l'automate une valeur positive, qui correspond au coût de cette étape. Ce coût s'étend naturellement à une exécution de l'automate : soit $\rho = (q_0, x_0, t_0, y_0) \xrightarrow{\tau_1, a_1} \cdots \xrightarrow{\tau_p, a_p} (q_p, x_p, t_p, y_p)$ une exécution finie de $\mathcal{H}$. Le *coût*[1] *de ρ* est $\mathsf{Coût}(\rho) = \sum_{i=1}^{p} \mathsf{Coût}(q_{i-1}, x_{i-1}, t_{i-1}, \tau_i)$. Donnons un exemple d'automate o-minimal pondéré.

Exemple 4.1.2 ([BLM07]) *L'automate o-minimal pondéré de la figure 4.1 modélise un processus de réparation. La réparation d'un problème a un certain coût, capturé par la fonction* $\mathsf{Coût}$. *Dès qu'un problème survient (modélisé par la transition pb) la valeur du coût augmente avec une pente de 1, jusqu'à ce qu'une réparation soit effectuée par une des transitions rp_1 (réparation peu chère mais longue) ou rp_2 (réparation chère mais rapide).*

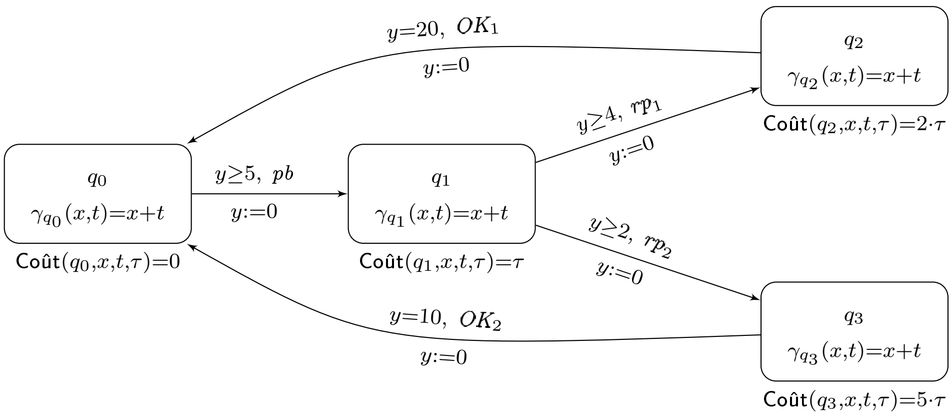


FIG. 4.1 – Un problème de réparation

[1] Si ρ finit avec une action de délai, c'est-à-dire qu'il y a une transition supplémentaire $(q_p, x_p, t_p, y_p) \xrightarrow{\tau_{p+1}} (q_{p+1}, x_{p+1}, t_{p+1}, y_{p+1})$, il faut alors ajouter le terme $\mathsf{Coût}(q_p, x_p, t_p, \tau_{p+1})$ dans $\mathsf{Coût}(\rho)$.

4.1.2 Problèmes associés

Nous définissons maintenant les deux problèmes auxquels nous allons nous intéresser dans ce chapitre : le *problème de contrôle optimal* et le *problème de* model-checking *de* WCTL .

Le problème de contrôle optimal

Le problème de contrôle d'atteignabilité optimale a été étudié en premier lieu dans le cadre des automates temporisés pondérés [ABM04, BCFL04]. Cependant il a été prouvé dans [BBR05, BBM06] que ce problème est indécidable pour les automates temporisés pondérés.

Dans notre cadre de jeux o-minimaux pondérés, le *problème de contrôle optimal* consiste à calculer le coût optimal pour le contrôleur pour atteindre But quoi que fasse l'environnement. Pour prendre en compte la fonction de coût, nous définissons le *coût d'une stratégie depuis un état* et le *coût optimal depuis un état.*

Définition 4.1.3 (Coût d'une stratégie depuis un état) *Soit* $(\mathcal{H}, \mathsf{Coût})$ *un jeu o-minimal pondéré, s un état de* V_2 *et* λ *une stratégie. Le* coût $\mathsf{Coût}(s,\lambda)$ de λ à partir de *s est défini par :*

$$\mathsf{Coût}(s,\lambda) = \sup\big\{\mathsf{Coût}(\rho) \;\mid\; \rho \textit{ est une exécution à partir de } s \textit{ compatible avec } \lambda\big\}.$$

Le contrôleur cherche à minimiser le coût, mais l'environnement n'est pas nécessairement coopératif, ce qui explique la présence du supremum sur les exécutions compatibles avec la stratégie du contrôleur.

Définition 4.1.4 (Coût optimal depuis un état) *Soit* $(\mathcal{H}, \mathsf{Coût})$ *un jeu o-minimal pondéré et s un état de* V_2*. Le* coût optimal $\mathsf{CoûtOpt}(s)$ associé à *s est défini par :*

$$\mathsf{CoûtOpt}(s) = \inf\big\{\mathsf{Coût}(s,\lambda) \;\mid\; \lambda \textit{ est une stratégie gagnante}\big\}.$$

Une stratégie gagnante à partir de s est *optimale* si $\mathsf{Coût}(s,\lambda) = \mathsf{CoûtOpt}(s)$.

Problème 3 (Problème de contrôle optimal) *Étant donné un jeu o-minimal pondéré* $(\mathcal{H}, \mathsf{Coût})$*, un état définissable s et une constante définissable*[2] *c, déterminer s'il existe une stratégie gagnante* λ *à partir de s telle que* $\mathsf{Coût}(s,\lambda) \leq c$.

Problème 4 (Calcul du coût optimal) *Étant donné un jeu o-minimal pondéré* $(\mathcal{H}, \mathsf{Coût})$ *et un état définissable s, calculer le coût optimal* $\mathsf{CoûtOpt}(s)$.

Remarque 4.1.5 *Il existe une stratégie gagnante optimale à partir de s si, et seulement si l'infimum peut être remplacé par un minimum dans la définition de* $\mathsf{OptCost}(s)$*. Si nous savons résoudre les problèmes 3 et 4, nous pouvons également déterminer s'il existe une stratégie gagnante optimale en demandant s'il existe une stratégie* λ *telle que* $\mathsf{Coût}(s,\lambda) \leq \mathsf{OptCost}(s)$*. Dans [BBR05, BBM06], il a été montré que la variante du problème 3 pour les automates temporisés pondérés à plus de trois horloges est indécidable.*

[2] Nous rappelons qu'un état ou une constante sont définissables s'ils sont définissables par une formule du premier ordre dans la structure sous-jacente.

Le problème de *model-checking* de WCTL.

La logique WCTL (pour *Weighted Computational Tree Logic*) a été proposée dans le contexte des systèmes temporisés pondérés comme une extension de CTL avec des contraintes de coût sur les modalités [BBR04, BBM06, BLM07]. Dans notre cadre, nous définissons pour toute structure $\mathcal{M}$ la logique $\mathsf{WCTL}_{\mathcal{M}}$ sur Σ inductivement comme suit :

$$\varphi \ ::= \ a \ \mid \ \varphi \vee \varphi \ \mid \ \neg\varphi \ \mid \ \mathbf{E}\,\varphi\,\mathbf{U}_{\sim c}\,\varphi \ \mid \ \mathbf{A}\,\varphi\,\mathbf{U}_{\sim c}\,\varphi$$

où $a \in \Sigma$, $\sim \in \{<, \leq, =, \geq, >\}$ et c est une constante définissable dans $\mathcal{M}$.

Soit $(\mathcal{H}, \mathsf{Coût})$ un $\mathcal{M}$-automate o-minimal. Nous supposons par la suite que $\mathcal{H}$ est sans blocage, c'est-à-dire qu'il existe une transition discrète à partir de tout état $(q, y) \in Q \times V_2$.

Pour définir la sémantique de $\mathsf{WCTL}_{\mathcal{M}}$, nous aurons besoin de la notion de *position* dans une exécution. Soit $\rho = (q_0, x_0, t_0, y_0) \xrightarrow{\tau_1, a_1} (q_1, x_1, t_1, y_1) \xrightarrow{\tau_2, a_2} \cdots$ une exécution de $\mathcal{H}$. Une *position* p de ρ est un couple $(i, \tau) \in \mathbb{N} \times M^+$ tel que $\tau \leq \tau_{i+1}$. Nous définissons l'état $\rho[(i, \tau)] = (q_i, \gamma_{q_i}(x_i, t_i + \tau))$ et l'exécution $\rho_{\leq (i,\tau)} = (q_0, x_0, t_0, y_0) \xrightarrow{\tau_1, a_1} \cdots \xrightarrow{\tau_{i-1}, a_{i-1}} (q_i, x_i, t_i, y_i) \xrightarrow{\tau_i} (q_i, x_i, t_i + \tau_i, \gamma_{q_i}(x_i, t_i + \tau))$; nous définissons de même l'exécution $\rho_{\geq (i,\tau)}$. Remarquons que les positions d'une exécution sont totalement ordonnées de manière naturelle.

La sémantique de $\mathsf{WCTL}_{\mathcal{M}}$ est définie pour tout état $(q, y) \in Q \times V_2$ de $(\mathcal{H}, \mathsf{Coût})$ de la manière suivante :

$$\begin{array}{rcl}
(q,y) \models a & \Leftrightarrow & (q,y) \xrightarrow{a} (q',y') \text{ pour } (q',y') \in Q \times V_2 \ ^{3} \\
(q,y) \models \neg\varphi & \Leftrightarrow & (q,y) \not\models \varphi \\
(q,y) \models \varphi_1 \vee \varphi_2 & \Leftrightarrow & (q,y) \models \varphi_1 \text{ ou } (q,y) \models \varphi_2 \\
(q,y) \models \mathbf{E}\,\varphi_1\,\mathbf{U}_{\sim c}\,\varphi_2 & \Leftrightarrow & \text{il existe une exécution infinie } \rho \text{ de } \mathcal{H} \text{ à partir de } (q,y) \\
 & & \text{telle que } \rho \models \varphi_1\,\mathbf{U}_{\sim c}\,\varphi_2 \\
(q,y) \models \mathbf{A}\,\varphi_1\,\mathbf{U}_{\sim c}\,\varphi_2 & \Leftrightarrow & \text{toute exécution infinie } \rho \text{ dans } \mathcal{H} \text{ à partir de } (q,y) \\
 & & \text{satisfait } \varphi_1\,\mathbf{U}_{\sim c}\,\varphi_2 \\
\rho \models \varphi_1\,\mathbf{U}_{\sim c}\,\varphi_2 & \Leftrightarrow & \text{il existe } p \geq 0 \text{ une position de } \rho \text{ telle que} \\
 & & \rho[p] \models \varphi_2, \text{ et pour toute position } 0 \leq p' < p \text{ de } \rho, \\
 & & \rho[p'] \models \varphi_1 \vee \varphi_2, \text{ et } \mathsf{Coût}(\rho_{\leq p}) \sim c \ ^{4}
\end{array}$$

Nous utilisons les raccourcis classiques : $\top$ pour $a \vee \neg a$, $\mathbf{E}\,\mathbf{F}_{\sim c}\,\varphi$ (resp. $\mathbf{A}\,\mathbf{F}_{\sim c}\,\varphi$) pour $\mathbf{E}\,\top\,\mathbf{U}_{\sim c}\,\varphi$ (resp. $\mathbf{A}\,\top\,\mathbf{U}_{\sim c}\,\varphi$) et $\mathbf{A}\,\mathbf{G}_{\sim c}\,\varphi$ (resp. $\mathbf{E}\,\mathbf{G}_{\sim c}\,\varphi$) pour $\neg\mathbf{E}\,\mathbf{F}_{\sim c}\,\neg\varphi$ (resp. $\neg\mathbf{A}\,\mathbf{F}_{\sim c}\,\neg\varphi$).

L'exemple suivant reprend le cadre du problème de réparation de l'exemple 4.1.2 et illustre le type de propriétés pouvant être exprimées dans WCTL.

Exemple 4.1.6 ([BLM07]) *Une propriété qui peut être exprimée dans* WCTL *est «* chaque occurrence d'un problème peut être réparée avec un coût total inférieur à 55 *». Cette propriété peut être exprimée par la formule suivante :*

$$\mathbf{A}\,\mathbf{G}\,\big(pb \ \Rightarrow \ \mathbf{E}\,\mathbf{F}_{\leq 55}\,(OK_1 \vee OK_2)\big).$$

Il est facile de vérifier que cette formule est vraie pour tout état de l'automate de la figure 4.1.

[3] Cela peut également être vu comme des propositions atomiques.

[4] Comme par exemple dans [Ras99], nous demandons aux positions intermédiaires de satisfaire $\varphi_1 \vee \varphi_2$ pour avoir une interprétation naturelle de la satisfaction quand φ_2 est vraie sur un intervalle ouvert à gauche.

Problème 5 (*Model-checking* de WCTL) *Étant donné un $\mathcal{M}$-automate o-minimal pondéré $(\mathcal{H}, \mathsf{Coût})$, φ une formule de $\mathsf{WCTL}_{\mathcal{M}}$ et $(q, y) \in Q \times V_2$ un état définissable de $\mathcal{H}$, déterminer si $(q, y) \models \varphi$.*

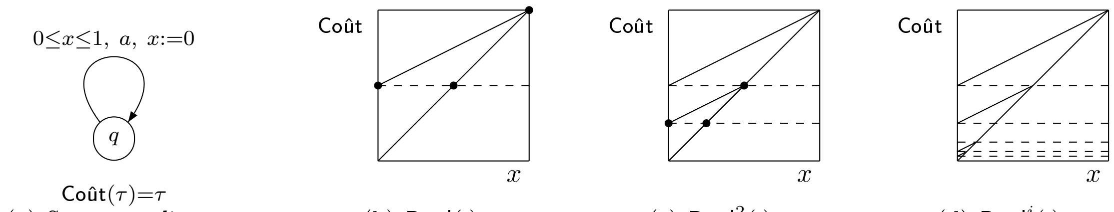
Les résultats classiques sur les systèmes o-minimaux ne s'appliquent pas pour les problèmes ci-dessus : en particulier, les automates o-minimaux pondérés n'admettent pas de bisimulation finie comme le précise la remarque suivante.

Remarque 4.1.7 *Un $\mathcal{M}$-automate o-minimal pondéré $(\mathcal{H}, \mathsf{Coût})$ peut être vu comme un $\mathcal{M}$-automate $\mathcal{H}_c$ avec une variable supplémentaire (la variable de coût). Cependant il est facile de voir que $\mathcal{H}_c$ n'est pas un automate o-minimal ; en effet la variable de coût n'est jamais remise à zéro. En particulier, les automates o-minimaux pondérés (vus comme des $\mathcal{M}$-automates avec une variable supplémentaire) n'admettent pas toujours de bisimulation finie. Nous donnons un tel exemple ci-dessous. Considérons l'automate o-minimal de la figure 4.2(a) où la dynamique $\gamma : \mathbb{R}_{\geq 0} \times \{1, 2\} \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ est définie par $\gamma_q(x, p, t) = x + p \cdot t$ et la fonction de coût par $\mathsf{Coût}(q, x_1, x_2, p, t, \tau) = \tau$.*

Montrons maintenant que l'automate o-mininal pondéré de la figure 4.2(a) n'admet pas de bisimulation finie à temps abstrait. Nous montrons que l'algorithme classique de bisimulation [PT87, Hen95] ne termine pas. L'algorithme de bisimulation fonctionne intuitivement comme suit : étant données deux pièces P_1 et P_2 de la partition initiale, l'algorithme calcule $P_1 \cap \mathsf{Pred}(P_2)$ (où $\mathsf{Pred}(P_2)$ est l'ensemble des prédécesseurs[5] de P_2). Si le nouvel ensemble $P_1 \cap \mathsf{Pred}(P_2)$ raffine la pièce P_1, alors nous créons une nouvelle partition où la pièce P_1 est remplacée par les deux pièces $P_1 \cap \mathsf{Pred}(P_2)$ et $P_1 \setminus \mathsf{Pred}(P_2)$. L'algorithme de bisimulation itère cette opération jusqu'à obtenir une partition stable par $\mathsf{Pred}(\cdot)$, dans ce cas la partition obtenue est une bisimulation à temps abstrait. Si cet algorithme itératif ne termine pas, c'est qu'il n'existe pas de bisimulation finie à temps abstrait.

Dans cet exemple, l'espace d'états à considérer est $(\mathbb{R}_{\geq 0})^2$. Supposons que dans la partition initiale le point $\{(1, 1)\}$ et la droite $x = 0$ soient isolés.

Appliquons maintenant l'algorithme de bisimulation à l'exemple de la figure 4.2(a). D'après la figure 4.2(b), nous remarquons que $\mathsf{Pred}_t(\{(1, 1)\})$ isole le point $(0, \frac{1}{2})$ et $\mathsf{Pred}_a(\{(0, \frac{1}{2})\})$ isole le point $(\frac{1}{2}, \frac{1}{2})$. Sur la figure 4.2(c), nous répétons cette opération pour isoler le point $(\frac{1}{4}, \frac{1}{4})$. Ce processus ne termine pas, ce qui prouve qu'il n'existe pas de bisimulation finie à temps abstrait pour cet automate (voir la figure 4.2(d)).



(a) Structure discrète (b) $\mathsf{Pred}(\cdot)$ (c) $\mathsf{Pred}^2(\cdot)$ (d) $\mathsf{Pred}^i(\cdot)$

FIG. 4.2 – Un système simple

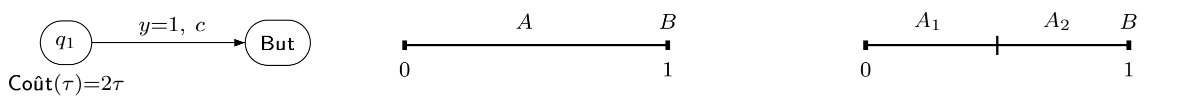
[5] $\mathsf{Pred}(P)$ est soit $\mathsf{Pred}_t = \{y \in V_2 \mid \exists y' \in P \text{ tel que } y \xrightarrow{t} y'\}$ ou $\mathsf{Pred}_a = \{y \in V_2 \mid \exists y' \in P \text{ tel que } y \xrightarrow{a} y'\}$.

4.2 Problème de contrôle optimal

Dans cette section, nous montrons que le problème 3 est décidable.

Remarque 4.2.1 *Dans le chapitre 3, pour prouver la décidabilité du problème de contrôle, nous avons utilisé le fait que l'équivalence par suffixe était suffisante pour distinguer les états gagnants des états perdants. Dans le cadre du contrôle optimal, ce n'est pas le cas comme en atteste l'exemple suivant. Considérons le jeu o-minimal pondéré* $(\mathcal{H}, \mathsf{Coût})$ *décrit sur la figure 4.3(a), où la dynamique dans* q_1 *est définie par* $\gamma(x,t) = x + t$ *et la fonction de coût dans* q_1 *est* $\mathsf{Coût}(q_1, x, t, \tau) = 2\tau$*. Dans cet exemple, la partition de* $[0,1]$ *respectant les gardes et remises à zéro consiste à isoler le point* $\{1\}$*, c'est-à-dire* $\mathcal{P}_{\mathcal{H}} = \{A, B\}$ *où* $A = [0,1[$ *et* $B = \{1\}$ *(voir la figure 4.3(b)). Il est facile de voir Suf(*$\mathcal{P}_{\mathcal{H}}$*)* $= \mathcal{P}_{\mathcal{H}}$*.*

Supposons que nous voulions atteindre But *avec un coût inférieur ou égal à* 1*. Il est facile de voir que cet objectif est réalisable à partir du point* $s = (q_1, 0.6)$ *mais pas du point* $s' = (q_1, 0)$*. Cependant les points* s *et* s' *ont le même suffixe sur* $\mathcal{P}_{\mathcal{H}}$*. La partition correcte pour ce jeu est donnée sur la figure 4.3(c).*



(a) Le $\mathcal{M}$-jeu $\mathcal{A}$ (b) La partition suffixe dans q_1 (c) La partition correcte dans q_1

FIG. 4.3 – La partition suffixe n'est pas suffisante

Nous montrons maintenant un point clé : nous pouvons nous restreindre aux stratégies qui franchissent chaque transition au plus une fois. La preuve de la proposition 4.2.4 repose sur le fait que les fonctions de coût sont positives ou nulles.

Définition 4.2.2 *Soit* $(\mathcal{H}, \mathsf{Coût})$ *un jeu o-minimal pondéré. Nous disons qu'une exécution* $(q_0, x_0, t'_0, y_0) \xrightarrow{\tau_1, a_1} \cdots$ traverse la transition $e = (q, g, a, R, q')$ à l'étape i *si* $q = q_{i-1}$*,* $q' = q_i$*,* $a = a_i$*,* $\gamma_{q_{i-1}}(x_{i-1}, t'_{i-1} + \tau_i) \in g$ *et* $y_i \in R$*.*

Définition 4.2.3 *Nous disons qu'une exécution* $\rho' = s'_0 \to s'_1 \to \cdots \to s'_k$ étend une exécution $\rho = s_0 \to s_1 \to \cdots \to s_l$ *si* $l \leq k$ *et pour tout* $0 \leq i \leq l$ $s'_i = s_i$*.*

Proposition 4.2.4 *Soit* $(\mathcal{H}, \mathsf{Coût})$ *un jeu o-minimal pondéré et* (q, y) *un état de* $\mathcal{H}$*. S'il existe une stratégie gagnante* λ *à partir de* (q, y) *alors il existe une stratégie gagnante* λ_{nouv} *avec* $\mathsf{Coût}((q,y), \lambda_{nouv}) \leq \mathsf{Coût}((q,y), \lambda)$ *telle que toute exécution compatible avec* λ_{nouv} *à partir de* (q, y) *franchit chaque transition au plus une fois.*

Preuve. L'intuition de la preuve est la suivante : pour une transition fixée e, nous assurons qu'il existe une stratégie gagnante (de coût $c \leq \mathsf{Coût}((q,y), \lambda)$) franchissant e au plus une fois. Considérons l'ensemble des exécutions à partir de (q, y) compatibles avec λ comme un arbre (infiniment branchant) : tout chemin de cet arbre est fini et atteint But, mais il peut franchir e plusieurs fois. Nous construisons une nouvelle stratégie λ_{nouv} qui court-circuite λ dans le sens suivant : elle simule λ jusqu'à ce que e soit franchie pour la première fois, elle passe alors directement à un descendant dans l'arbre à partir duquel tous les chemins ne franchissent plus e (c'est possible car λ est gagnante). Une telle stratégie franchit e au plus une fois, son coût est plus petit que celui de λ car la fonction de coût est positive et les exécutions qui sont compatibles avec λ_{nouv} sont « plus courtes » que celles compatibles avec λ.

Sans perte de généralité nous supposons qu'il n'y a pas de transition sortant des états de But. Nous fixons un état (q_0, y_0), une stratégie gagnante λ à partir de (q_0, y_0) et une transition e. Nous montrons qu'il existe une stratégie gagnante λ_{nouv} telle que toute exécution compatible avec λ_{nouv} franchit e au plus une fois.

La construction de λ_{nouv} est basée sur le lemme suivant :

Lemme 4.2.5 *Soit ρ une exécution compatible avec λ franchissant e et $(q, x, t, y) \in R(e)$. Alors il existe une exécution $r^{ext}(\rho, (q, x, t, y))$ (notée de manière plus concise r^{ext}) étendant ρ, compatible avec λ, telle que $dernier(r^{ext}) = (q, x, t, y)$ et pour tout ρ' étendant r^{ext} compatible avec λ, ρ' ne franchit e à aucune étape $i > |r^{ext}|$.*

Preuve.[Preuve du lemme] Supposons par l'absurde qu'il existe ρ et $(q, x, t, y) \in R(e)$ ne satisfaisant pas le lemme.

Alors pour toute exécution r^{ext} étendant ρ, compatible avec λ, telle que $dernier(r^{ext}) = (q, x, t, y)$, il existe ρ' étendant r^{ext} compatible avec λ, et franchissant e strictement après $|r^{ext}|$.

En considérant $r^{ext} = \rho$ nous pouvons trouver ρ_1 avec $dernier(\rho_1) = (q, x, t, y)$ étendant ρ, compatible avec λ, et franchissant e strictement après $|\rho|$. En considérant alors $r^{ext} = \rho_1$ nous trouvons une exécution similaire ρ_2 franchissant e strictement après $|\rho_1|$. Nous construisons itérativement une exécution infinie compatible avec λ franchissant e infiniment souvent. C'est une contradiction car λ est une stratégie gagnante et But n'a pas de transition sortante. $\square$

Soit $\rho = (q_0, x_0, t'_0, y_0) \xrightarrow{\tau_1, a_1} \cdots$ une exécution, la stratégie $\lambda_{nouv}(\rho)$ est définie par :
- si ρ ne franchit pas e alors $\lambda_{nouv}(\rho) = \lambda(\rho)$,
- sinon soit i le plus petit entier tel que ρ franchit e à l'étape i. Soit ρ_{nouv} l'exécution $r^{ext}(\rho, (q_i, x_i, t'_i, y_i)) \xrightarrow{\tau_{i+1}, a_{i+1}} (q_{i+1}, x_{i+1}, t'_{i+1}, y_{i+1}) \xrightarrow{\tau_{i+2}, a_{i+2}} \cdots$ Alors $\lambda_{nouv}(\rho) = \lambda(\rho_{nouv})$.

Informellement λ_{nouv} réagit comme λ jusqu'à ce que e soit franchie, et réagit alors comme λ *en supposant* que r^{ext} a été lu. λ_{nouv} satisfait les propriétés suivantes :

Lemme 4.2.6 *Soit ρ une exécution compatible avec λ_{nouv} :*
- *ρ franchit e au plus une fois,*
- *si ρ est maximale par rapport à λ_{nouv} alors elle est gagnante,*
- *il existe ρ_0 compatible avec λ tel que $\mathsf{Coût}(\rho) \leq \mathsf{Coût}(\rho_0)$,*
- *si e' est une transition et ρ franchit e' k fois alors il existe ρ_1 compatible avec λ franchissant e' k' fois avec $k' \geq k$.*

Preuve.[Preuve du lemme] Soit $\rho = (q_0, x_0, t'_0, y_0) \xrightarrow{\tau_1, a_1} \cdots$ une exécution compatible avec λ_{nouv}. Si ρ ne franchit pas e alors le lemme est trivial. Sinon soit ρ_{nouv} comme dans la définition de λ_{nouv}. La preuve des quatre points du lemme est une conséquence du fait que ρ_{nouv} soit compatible avec λ. $\square$

Le quatrième point du lemme 4.2.6 assure que nous pouvons appliquer cette procédure successivement à toutes les transitions et ainsi trouver une stratégie gagnante qui franchit chaque transition au plus une fois, ce qui achève la preuve de la proposition 4.2.4. □

Remarque 4.2.7 *Remarquons que la proposition 4.2.4 n'est pas vérifiée dans le cadre du contrôle optimal pour les automates temporisés.*

Nous donnons maintenant un algorithme de calcul en arrière pour trouver le coût optimal, basé sur la formulation du problème donnée dans [LMM02, ABM04]. La terminaison de cet algorithme repose sur la proposition 4.2.4.

Étant donné $(q, y) \in Q \times V_2$ et $n \in \mathbb{N}$ nous définissons tout d'abord $c_n(q, y)$ le coût optimal pour atteindre But en moins de n étapes à partir de (q, y). Formellement nous avons que :

- $c_0(q, y) = 0$ si $q \in \mathsf{But}$, $+\infty$ si $q \notin \mathsf{But}$.
- $c_{n+1}(q, y) = \sup_{\gamma_q(x,t)=y} \inf_{(\tau,a)\in\mathsf{Enb}(q,x,t,y)} \max \left\{ \begin{array}{l} \mathsf{Coût}_{c_n}^{\tau,a}(q, x, t, y) \\ \sup\{\mathsf{Coût}_{c_n}^{\tau',u}(q, x, t, y) \mid {\scriptstyle (\tau',u)\in\mathsf{Enb}(q,x,t,y),\tau'\leq\tau}\} \end{array} \right.$

  où $(\tau, e) \in \mathsf{Enb}(q, x, t, y)$ ssi $e \in \mathsf{Enb}(q, x, t + \tau, \gamma_q(x, t + \tau))$,
  et $\mathsf{Coût}_c^{\tau,e}(q, x, t, y) = \mathsf{Coût}(q, x, t, y, \tau) + \sup\{c(q', y') \mid (q, x, t, y) \xrightarrow{\tau,a} (q', x', t', y')\}$.

Définition 4.2.8 *Une stratégie λ est n-*bornée *à partir de (q, y) si toute exécution compatible avec λ à partir de (q, y) est de longueur au plus n.*

Lemme 4.2.9 *Pour tout $\varepsilon > 0$, pour tout $(q, y) \in Q \times V_2$ tel que $c_n(q, y) < +\infty$, il existe une stratégie définissable n-bornée λ à partir de (q, y) telle que $\mathsf{Coût}((q, y), \lambda) \leq c_n(q, y) + \varepsilon$.*

Preuve. Nous montrons ce lemme par récurrence sur n. Pour $n = 0$, si $c_0(q, y) < +\infty$, alors $(q, y) \in \mathsf{But}$, il suffit donc de prendre λ non-définie.

Supposons le résultat vrai pour $n - 1$; soit $\varepsilon > 0$ et (q_0, y_0) tel que $c_n(q_0, y_0) < +\infty$. Nous construisons une stratégie n-bornée λ à partir de (q_0, y_0) telle que $\mathsf{Coût}((q_0, y_0), \lambda) \leq c_n(q_0, y_0) + \varepsilon$. Comme nous avons supposé le résultat vrai pour $n - 1$, pour tout (q', y') tel que $c_{n-1}(q', y') < +\infty$, il existe une stratégie $(n - 1)$-bornée à partir de (q', y'), notée $\lambda^{q',y'}$, telle que $\mathsf{Coût}((q', y'), \lambda^{q',y'}) \leq c_{n-1}(q', y') + \frac{\varepsilon}{2}$.
Soit (τ, a) tel que

$$\max \left\{ \begin{array}{l} \mathsf{Coût}_{c_n}^{\tau,a}(q_0, x_0, t_0, y_0) \\ \sup\{\mathsf{Coût}_{c_n}^{\tau',u}(q_0, x_0, t_0, y_0) \mid {\scriptstyle (\tau',u)\in\mathsf{Enb}(q_0,x_0,t_0,y_0),\tau'\leq\tau}\} \end{array} \right\} \leq c_n(q, y) + \frac{\varepsilon}{2}.$$

Nous construisons maintenant la stratégie λ en utilisant (τ, a) et les stratégies $\lambda^{q',y'}$. Soit $\rho = (q_0, x_0, t_0, y_0) \xrightarrow{\tau_1,a_1} \cdots \xrightarrow{\tau_m,a_m} (q_m, x_m, t_m, y_m)$, nous définissons $\lambda(\rho)$ comme suit :
- si $|\rho| = 1$ alors $\lambda(\rho) = (\tau, a)$,
- si $|\rho| \geq 2$ alors $\lambda(\rho) = \lambda^{q_1,y_1}(\rho_{\geq(1,0)})$.

Nous montrons maintenant que λ est une stratégie n-bornée à partir de (q, y) et que $\mathsf{Coût}((q, y), \lambda) \leq c_n(q, y) + \varepsilon$.

Soit $\rho = (q_0, x_0, t_0, y_0) \xrightarrow{\tau_1,a_1} \cdots \xrightarrow{\tau_m,a_m} (q_m, x_m, t_m, y_m)$ une exécution compatible avec λ. L'exécution $\rho_{\geq(1,0)}$ est compatible avec λ^{q_1,y_1}, elle est donc gagnante et nous avons que $\sum_{i=1}^{m-1} \mathsf{Coût}(q_i, x_i, t_i, \tau_i) \leq c_{n-1}(q_1, y_1) + \frac{\varepsilon}{2}$. Comme ρ est compatible avec λ, en considérant la première étape nous déduisons que $\mathsf{Coût}(q_0,x_0,t_0,\tau_0)+c_{n-1}(q_1,y_1)\leq c_n(q,y)+\frac{\varepsilon}{2}$. Donc $\mathsf{Coût}(\rho)=\sum_{i=0}^{m-1}\mathsf{Coût}(q_i,x_i,t_i,\tau_i)\leq c_n(q,y)+\varepsilon$.

Notons que λ est définissable. En effet λ^{q_1,y_1} est définissable par hypothèse de récurrence. De plus il existe $a\in\Sigma_c$ tel que l'ensemble $\mathsf{Temps}_{q_0,x_0,t_0,y_0,\varepsilon,a}$ défini par

$$\left\{\tau \mid \max\left\{\begin{array}{l}\mathsf{Coût}^{\tau,a}_{c_n}(q_0,x_0,t_0,y_0)\\ \sup\{\mathsf{Coût}^{\tau',u}_{c_n}(q_0,x_0,t_0,y_0)\mid{\scriptstyle (\tau',u)\in\mathsf{Enb}(q_0,x_0,t_0,y_0),\tau'\leq\tau}\}\end{array}\right\}\leq c_n(q,y)+\frac{\varepsilon}{2}\right\}$$

n'est pas vide. Alors en utilisant la *sélection de courbe* d'extensions o-minimales de groupes ordonnés (voir [vdD98, chap.6]), nous pouvons choisir de manière définissable un délai τ dans cet ensemble. $\square$

Lemme 4.2.10 *Si λ est une stratégie n-bornée à partir de (q,y) alors* $\mathsf{Coût}((q,y),\lambda)\geq c_n(q,y)$.

Preuve. Nous montrons ce lemme par récurrence sur n. Il est immédiat pour $n=0$.

Supposons le lemme vrai pour $n-1$ et soit λ une stratégie n-bornée à partir de (q_0,y_0), nous montrons que $\mathsf{Coût}((q_0,y_0),\lambda)\geq c_n(q_0,y_0)$. Soit (x_0,t_0) tel que $\gamma_{q_0}(x_0,t_0)=y_0$, nous devons trouver (τ,a) tel que

$$\mathsf{Coût}((q_0,y_0),\lambda)\geq\max\left\{\begin{array}{l}\mathsf{Coût}^{\tau,a}_{c_{n-1}}(q_0,x_0,t_0,y_0)\\ \sup\{\mathsf{Coût}^{\tau',u}_{c_{n-1}}(q_0,x_0,t_0,y_0)\mid{\scriptstyle (\tau',u)\in\mathsf{Enb}(q_0,x_0,t_0,y_0),\tau'\leq\tau}\}\end{array}\right\}.$$

Pour montrer que (τ,a) a la propriété requise, nous considérons une stratégie λ_1 à partir de (q_1,y_1) qui réagit comme λ si la première étape est (τ,a). Soit $\rho=(q_1,x_1,t_1,y_1)\xrightarrow{\tau_2,a_2}\cdots\xrightarrow{\tau_m,a_m}(q_m,x_m,t_m,y_m)$, $\lambda_1(\rho)$ est défini comme suit :

- si $(q_1,y_1)\neq(q',y')$ alors $\lambda_1(\rho)=\perp$ (c'est-à-dire que λ_1 n'est pas définie),
- si $(q_1,y_1)=(q',y')$ alors $\lambda_1(\rho)=\lambda((q_0,x_0,t_0,y_0)\xrightarrow{\tau,a}\rho)$.

λ_1 est une stratégie $(n-1)$-bornée à partir de (q',y'). Donc par hypothèse de récurrence $\mathsf{Coût}((q',y'),\lambda_1)\geq c_{n-1}(q',y')$. Il vient que pour tout $\varepsilon>0$ il existe une exécution ρ_ε compatible avec λ_1 telle que $\mathsf{Coût}(\rho_\varepsilon)\geq c_{n-1}(q',y')-\varepsilon$. Alors l'exécution $\rho=(q_0,x_0,t_0,y_0)\xrightarrow{\tau,a}\rho_\varepsilon$ est compatible avec λ, donc $\mathsf{Coût}((q_0,y_0),\lambda)\geq\mathsf{Coût}(q_0,x_0,t_0,\tau)+\mathsf{Coût}(\rho_\varepsilon)\geq\mathsf{Coût}^{\tau,a}_{c_{n-1}}(q_0,x_0,t_0,y_0)-\varepsilon$.

Cela est vrai pour tout $\varepsilon>0$ donc $\mathsf{Coût}((q_0,y_0),\lambda)\geq\mathsf{Coût}^{\tau,a}_{c_{n-1}}(q_0,x_0,t_0,y_0)$. De même nous montrons que pour tout $(\tau',u)\in\mathsf{Enb}(q_0,x_0,t_0,y_0)$, pour tout $\tau'\leq\tau$, nous avons que $\mathsf{Coût}((q_0,y_0),\lambda)\geq\mathsf{Coût}^{\tau',u}_{c_{n-1}}(q_0,x_0,t_0,y_0))$.

Nous avons donc que

$$\mathsf{Coût}((q_0,y_0),\lambda)\geq\max\left\{\begin{array}{l}\mathsf{Coût}^{\tau,a}_{c_{n-1}}(q_0,x_0,t_0,y_0)\\ \sup\{\mathsf{Coût}^{\tau',u}_{c_{n-1}}(q_0,x_0,t_0,y_0)\mid{\scriptstyle (\tau',u)\in\mathsf{Enb}(q_0,x_0,t_0,y_0),\tau'\leq\tau}\}\end{array}\right\}.$$

C'est ce que nous devions montrer, ceci achève donc la preuve du lemme. $\square$

Théorème 4.2.11 *Soit $\mathcal{M}=\langle M,+,0,1,\cdots\rangle$ une structure o-minimale telle que* $\mathsf{Th}(\mathcal{M})$ *est décidable. Le problème de contrôle optimal pour les $\mathcal{M}$-jeux o-minimaux est décidable.*

Preuve. Soit $(\mathcal{H}, \mathsf{Coût})$ un jeu o-minimal pondéré et s un état de $\mathcal{H}$. Les lemmes 4.2.9 et 4.2.10 montrent que $c_k(s) = \inf\{\mathsf{Coût}(s, \lambda) \mid \lambda \text{ est une stratégie } k\text{-bornée}\}$.

Soit n le nombre de transitions de $\mathcal{H}$. La proposition 4.2.4 montre que pour toute stratégie gagnante à partir de s il existe une stratégie gagnante $(n+1)$-bornée à partir de s avec un coût inférieur. Donc $\mathsf{CoûtOpt}(s) = \inf\{\mathsf{Coût}(s, \lambda) \mid \lambda \text{ est une stratégie } (n+1)\text{-bornée}\} = c_{n+1}(s)$. Remarquons que $c_{n+1}(s)$ est calculable car $\mathsf{Th}(\mathcal{M})$ est décidable.

Nous pouvons de plus décider si le coût optimal peut être réalisé par une stratégie : par la proposition 4.2.4 il est suffisant d'énumérer toutes les stratégies $(n+1)$-bornées en utilisant les τ_i comme des paramètres et de vérifier si le coût d'une de ces stratégies est égal à $c_{n+1}(s)$. □

Remarque 4.2.12 *La preuve du théorème 4.2.11 montre que s'il existe une stratégie assurant un coût donné c, alors pour tout $\varepsilon > 0$, nous pouvons construire une stratégie sans mémoire et définissable assurant un coût $c + \varepsilon$.*

Remarque 4.2.13 *Remarquons qu'il existe des exemples de jeux o-minimaux où (i) il n'existe pas de stratégie optimale ou (ii) le coût optimal ne peut être réalisé qu'avec de la mémoire. Ces exemples sont directement inspirés du cadre des automates temporisés pondérés à une horloge [BLMR06].*

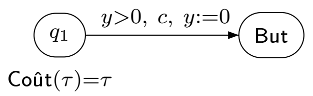


Fig. 4.4 – Pas de stratégie optimale

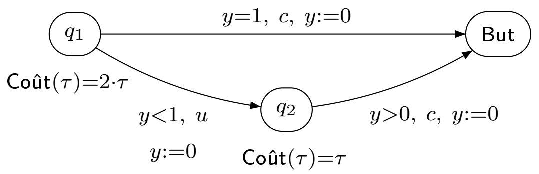


Fig. 4.5 – L'optimalité requiert de la mémoire

Sur la figure 4.4 nous avons un exemple d'un jeu o-minimal pondéré où le coût optimal est 0 *mais où aucune stratégie ne peut assurer ce coût optimal.*

Sur la figure 4.5 nous avons un exemple d'un jeu o-minimal pondéré où le coût optimal est 2 *mais où aucune stratégie sans mémoire ne peut assurer ce coût. En effet l'environnement peut amener le jeu en q_2 avec un coût $\mathsf{Coût}(\rho)$ arbitrairement proche de 2, le contrôleur doit alors réagir en jouant une action (τ, c) avec $\mathsf{Coût}(\rho) + \tau \leq 2$, ce qui est impossible sans mémoire.*

4.3 *Model-checking* de WCTL

Notre but dans cette section est de montrer le résultat de décidabilité ci-dessous. Les techniques que nous utiliserons pour prouver ce résultat sont en partie inspirées de la preuve de décidabilité de WCTL sur les automates temporisés pondérés à une horloge [BLM07].

Théorème 4.3.1 *Soit $\mathcal{M} = \langle M, +, 0, 1, <, \cdots \rangle$ une structure o-minimale telle que $\mathcal{M}$ est archimédienne*[6] *et $\mathsf{Th}(\mathcal{M})$ est décidable. Alors le* model-checking *de $\mathsf{WCTL}_{\mathcal{M}}$ est décidable.*

[6] Une structure est *archimédienne* si pour tout $(a, b) \in M^2$ tel que $a > 0$, il existe un entier n tel que $n \cdot a > b$.

Si $\mathcal{P}$ et $\mathcal{P}'$ sont deux partitions, nous écrivons $\mathcal{P} \sqcap \mathcal{P}'$ pour la partition jointe, c'est-à-dire la partition la plus grossière qui raffine à la fois $\mathcal{P}$ et $\mathcal{P}'$. Remarquons que si $\mathcal{P}$ et $\mathcal{P}'$ sont toutes deux finies et définissables alors $\mathcal{P} \sqcap \mathcal{P}'$ est finie et définissable.

Soit $\mathcal{H} = (\mathcal{M}, Q, \Sigma, \delta, \gamma)$ un $\mathcal{M}$-automate o-minimal. Soit $\mathcal{P}$ une partition de $S = Q \times V_2$ et φ une formule de $\mathsf{WCTL}_{\mathcal{M}}$. $\mathcal{P}$ *est une partition pour* φ si pour tout $P \in \mathcal{P}$, soit tous les états $(q, y) \in P$ satisfont φ, soit aucun état $(q, y) \in P$ ne satisfait φ.

Étant donné un $\mathcal{M}$-automate o-minimal $\mathcal{H}$, nous rappelons que $\mathcal{P}_{\mathcal{H}}$ est la partition la plus grossière respectant les gardes et remises à zéro de $\mathcal{H}$. $\mathcal{P}_{\mathcal{H}}$ est une partition pour les propositions atomiques $a \in \Sigma$. Pour toute formule de $\mathsf{WCTL}_{\mathcal{M}}$, nous construisons inductivement un raffinement (fini et définissable) $\mathcal{P}_\varphi$ de $\mathcal{P}_{\mathcal{H}}$ tel que $\mathcal{P}_\varphi$ est une partition pour φ.

Nous procédons en trois étapes : à partir de partitions pour φ et ψ nous construisons une partition pour $\mathbf{E}\,\varphi\,\mathbf{U}\,\psi$ et $\mathbf{A}\,\varphi\,\mathbf{U}\,\psi$, puis pour $\mathbf{E}\,\varphi\,\mathbf{U}_{\sim c}\,\psi$ et enfin pour $\mathbf{A}\,\varphi\,\mathbf{U}_{\sim c}\,\psi$.

4.3.1 Partition pour $\mathbf{E}\,\varphi\,\mathbf{U}\,\psi$ et $\mathbf{A}\,\varphi\,\mathbf{U}\,\psi$

Nous réalisons la première étape avec le lemme suivant, qui applique la construction d'une partition finie et définissable (la partition suffixe) induisant une bisimulation, et qui est donc correcte pour les formules de CTL [AHLP00, HMR05].

Lemme 4.3.2 *Soit $\mathcal{M} = \langle M, +, 0, 1, <, \cdots \rangle$ une structure o-minimale telle que $\mathcal{M}$ est archimédienne et $\mathsf{Th}(\mathcal{M})$ est décidable. Soit $(\mathcal{H}, \mathsf{Coût})$ un $\mathcal{M}$-automate o-minimal pondéré, soit φ et ψ deux formules de $\mathsf{WCTL}_{\mathcal{M}}$. Supposons que nous ayons construit deux partitions finies et définissables $\mathcal{P}_\varphi$ et $\mathcal{P}_\psi$ pour φ et ψ respectivement, alors nous pouvons construire une partition finie et définissable pour les formules $\mathbf{E}\,\varphi\,\mathbf{U}\,\psi$ et $\mathbf{A}\,\varphi\,\mathbf{U}\,\psi$.*

Preuve. $\mathcal{P}_\varphi \sqcap \mathcal{P}_\psi$ est une partition pour les deux formules φ et ψ. Par définition, la partition suffixe $\mathsf{Suf}(\mathcal{P}_\varphi \sqcap \mathcal{P}_\psi)$ raffine la partition $\mathcal{P}_\varphi \sqcap \mathcal{P}_\psi$. Par [Bri06, Theorem 12.6.14], $\mathsf{Suf}(\mathcal{P}_\varphi \sqcap \mathcal{P}_\psi)$ est une bisimulation à temps abstrait sur $\mathcal{H}$ pour φ et ψ. C'est donc une partition pour toute formule de CTL où φ et ψ sont utilisées comme des propositions atomiques (voir par exemple [AHLP00, HMR05]) ; c'est en particulier une partition pour $\mathbf{E}\,\varphi\,\mathbf{U}\,\psi$ et $\mathbf{A}\,\varphi\,\mathbf{U}\,\psi$.

Remarquons que nous avons prouvé que si $(q, y) \models \mathbf{E}\,\varphi\,\mathbf{U}\,\psi$ alors il existe un témoin (c'est-à-dire une exécution finie satisfaisant $\varphi\,\mathbf{U}\,\psi$) de longueur au plus $|\mathsf{Suf}(\mathcal{P}_\varphi \sqcap \mathcal{P}_\psi)|$ car cela correspond à un problème d'accessibilité dans l'abstraction. □

4.3.2 Partition pour $\mathbf{E}\,\varphi\,\mathbf{U}_{\sim c}\,\psi$

La construction d'une partition (finie et définissable) pour $\mathbf{E}\,\varphi\,\mathbf{U}_{\sim c}\,\psi$ est plus complexe. Notons $\mathcal{P}$ la partition $\mathsf{Suf}(\mathcal{P}_\varphi \sqcap \mathcal{P}_\psi)$. Soient q et q' deux états discrets de $\mathcal{H}$. Soient P et P' deux pièces de $\mathcal{P}$. Nous donnons tout d'abord une formule définissant l'ensemble des coûts possibles des chemins d'un état (ou d'une pièce de $\mathcal{P}$) à une pièce de $\mathcal{P}$:

- $\theta^{\varphi\vee\psi}_{(q,P)\to(q',P')}(c, y)$ exprime qu'il est possible d'aller d'un état (q, y) avec $y \in P$ à un autre état (q', y') avec $y' \in P'$ par une étape continue suivie d'une action discrète, avec un coût c, en satisfaisant constamment $\varphi \vee \psi$.
- $\theta^{\varphi\vee\psi}_{(q,P\rightsquigarrow P')}(c, y)$ exprime qu'il est possible d'aller d'un état (q, y) avec $y \in P$ à un autre état (q, y') avec $y' \in P'$ par une étape continue (et pas d'action discrète), avec un coût c, en satisfaisant constamment $\varphi \vee \psi$.

Les formules $\theta^{\varphi\vee\psi}_{(q,P)\to(q',P')}(c,y)$ (resp. $\theta^{\varphi\vee\psi}_{(q,P\rightsquigarrow P'')}$) sont définies dans $\mathcal{M}$ comme suit :

$$\begin{aligned}\theta^{\varphi\vee\psi}_{(q,P)\to(q',P')}(c,y) \;=\; & \exists x\in V_1\ \exists t\in V\ \exists\tau\in M^+\exists a\in\Sigma\ \exists y'\in V_2\\ & \gamma_q(x,t)=y\ \wedge\ (q,y)\xrightarrow{\tau,a}(q',y')\ \wedge\ (q,y)\in P\ \wedge\ (q',y')\in P'\ \wedge\\ & \quad \forall\tau'<\tau,\ (q,\gamma(x,t+\tau'))\models\varphi\vee\psi\ \ \wedge\ \mathsf{Co\hat{u}t}((q,x,t,y,\tau))=c.\end{aligned}$$

$$\begin{aligned}\theta^{\varphi\vee\psi}_{(q,P\rightsquigarrow P')}(c,y) \;=\; & \exists x\in V_1\ \exists t\in V\ \exists\tau\in M^+\ \exists y'\in V_2\\ & \gamma_q(x,t)=y\ \wedge\ (q,y)\xrightarrow{\tau}(q,y')\ \wedge\ (q,y)\in P\ \wedge\ (q',y')\in P'\ \wedge\\ & \quad \forall\tau'<\tau,\ (q,\gamma(x,t+\tau'))\models\varphi\vee\psi\ \ \wedge\ \mathsf{Co\hat{u}t}((q,x,t,y,\tau))=c.\end{aligned}$$

À partir des formules précédentes, nous définissons les ensembles définissables suivants :

$$\begin{aligned}\kappa^{\varphi\vee\psi}_{(q,P)\to(q',P')} &= \{c\in M^+\ \mid\ \exists y\in P \text{ tel que } \theta^{\varphi\vee\psi}_{(q,P)\to(q',P')}(c,y)\}\ ;\\ \kappa^{\varphi\vee\psi}_{(q,P\rightsquigarrow P')} &= \{c\in M^+\ \mid\ \exists y\in P \text{ tel que } \theta^{\varphi\vee\psi}_{(q,P\rightsquigarrow P')}(c,y)\}\ ;\\ \lambda^{\varphi\vee\psi}_{(q,P)\to(q',P')}(y) &= \{c\in M^+\ \mid\ \theta^{\varphi\vee\psi}_{(q,P)\to(q',P')}(c,y)\}\ ;\\ \lambda^{\varphi\vee\psi}_{(q,P\rightsquigarrow P')}(y) &= \{c\in M^+\ \mid\ \theta^{\varphi\vee\psi}_{(q,P\rightsquigarrow P')}(c,y)\}.\end{aligned}$$

Nous construisons alors un graphe fini qui abstrait la partie dynamique de $\mathcal{H}$ et restreint aux parties de $\mathcal{H}$ satisfaisant la formule $\mathbf{E}\,\varphi\,\mathbf{U}\,\psi$. Les arêtes de ce graphe sont étiquetées par un poids (en fait un ensemble définissable) qui représente l'ensemble des coûts des chemins de $\mathcal{H}$ vérifiant la formule $\mathbf{E}\,\varphi\,\mathbf{U}\,\psi$. Formellement, nous construisons un graphe fini pondéré (définissable) $\mathcal{G}_{\varphi,\psi}=(V,E)$ comme suit :

– son ensemble d'états V est

$$\{(q,P),(q,P,\mathsf{init})\ \mid\ (q,P)\models\varphi\vee\psi\}^{\,7}\quad\cup\quad\{(q,P,\mathsf{final})\ \mid\ (q,P)\models\psi\},$$

– ses arêtes E sont

$$\begin{aligned}&\{(q,P,\mathsf{init})\xrightarrow{\lambda^{\varphi\vee\psi}_{(q,P)\to(q',P')}(y)}(q',P')\}\cup\{(q,P,\mathsf{init})\xrightarrow{\lambda^{\varphi\vee\psi}_{(q,P\rightsquigarrow P'')}(y)}(q,P'',\mathsf{final})\}\\ \cup\quad&\{(q,P,\mathsf{init})\xrightarrow{[0]}(q,P,\mathsf{final})\}\cup\{(q,P)\xrightarrow{\kappa^{\varphi\vee\psi}_{(q,P)\to(q',P')}}(q',P')\}\\ \cup\quad&\{(q,P)\xrightarrow{\kappa^{\varphi\vee\psi}_{(q,P)\to(q',P')}}(q',P',\mathsf{final})\}\cup\{(q,P)\xrightarrow{\kappa^{\varphi\vee\psi}_{(q,P\rightsquigarrow P'')}}(q,P'',\mathsf{final})\}.\end{aligned}$$

Soit $\xi=(q_0,P_0,\mathsf{init})\xrightarrow{\lambda(y)}(q_1,P_1)\xrightarrow{\kappa_1}\cdots\xrightarrow{\kappa_n}(q_n,P_n,\mathsf{final})$ une exécution finie de $\mathcal{G}_{\varphi,\psi}$. Nous définissons $K_\xi(y)$ l'ensemble de tous les coûts possibles de l'exécution ξ :

$$K_\xi(y)=\{\lambda(y)+c_2+\cdots+c_n\mid c_i\in\kappa_i\}.$$

Les propositions suivantes montrent que le graphe $\mathcal{G}_{\varphi,\psi}$ peut être utilisé pour réaliser le *model-checking* de la formule $\mathbf{E}\,\varphi\,\mathbf{U}_{\sim c}\,\psi$.

[7] C'est un abus de notation : $(q,P)\models\varphi\vee\psi$ signifie que pour tout $y\in P$, $(q,y)\models\varphi\vee\psi$, ce qui est équivalent au fait qu'il existe $y\in P$ tel que $(q,y)\models\varphi\vee\psi$.

Proposition 4.3.3 *Soit ξ une exécution finie de $\mathcal{G}_{\varphi,\psi}$ telle que $\xi = (q_0, P_0, \mathit{init}) \xrightarrow{\lambda(y)} (q_1, P_1) \xrightarrow{\kappa_1} \cdots \xrightarrow{\kappa_{n-1}} (q_{n-1}, P_{n-1}) \xrightarrow{\kappa_n} (q_n, P_n, \mathit{final})$, et soit $y_0 \in P_0$. Alors pour tout $\zeta \in K_\xi(y_0)$, il existe une exécution réelle ρ de $\mathcal{H}$ telle que $\mathsf{Coût}(\rho) = \zeta$, pour toute position p sur ρ, $\rho[p] \models \varphi \vee \psi$, et dernier$(\rho) \models \psi$. La réciproque est également vraie.*

Preuve. Considérons tout d'abord le cas particulier des exécutions de longueur 1. Supposons que $\xi = (q_0, P_0, \mathsf{init}) \xrightarrow{\lambda(y)} (q_n, P_n, \mathsf{final})$. Deux cas sont possibles : soit $\lambda(y) = [0]$, soit $\lambda(y) = \lambda^{\varphi\vee\psi}_{(q,P\rightsquigarrow P'')}(y)$. Dans le premier cas l'exécution ρ est simplement (q_0, y_0). Dans le second cas, l'exécution $(q_0, y_0) \xrightarrow{\tau} (q_0, y_0'')$ est facilement obtenue à partir de la définition de $\lambda^{\varphi\vee\psi}_{(q,P\rightsquigarrow P'')}(y)$.

Considérons maintenant le cas général. Soit ξ l'exécution suivante :

$$\xi = (q_0, P_0, \mathsf{init}) \xrightarrow{\lambda(y)} (q_1, P_1) \xrightarrow{\kappa_1} \cdots \xrightarrow{\kappa_{n-1}} (q_{n-1}, P_{n-1}) \xrightarrow{\kappa_n} (q_n, P_n, \mathsf{final}),$$

et soit ζ un point de $K_\xi(y_0)$. Nous construisons une exécution réelle $\rho = (q_0, y_0) \xrightarrow{\tau_1, a_1} (q_1, y_1) \xrightarrow{\tau_2, a_2} \cdots \xrightarrow{\tau_n, a_n} (q_n, y_n)$ de $\mathcal{H}$ telle que $\mathsf{Coût}(\rho) = \zeta$, pour toute position p sur ρ, $\rho[p] \models \varphi \vee \psi$, et $(q_n, y_n) \models \psi$.

Comme $\zeta \in K_\xi(y_0)$, il existe $c_2, \cdots, c_n \in M^+$ tels que $c_i \in \kappa_i$ et $\zeta = \lambda(y_0) + c_2 + \cdots + c_n$. Comme $(q_0, P_0, \mathsf{init}) \xrightarrow{\lambda(y_0)} (q_1, P_1)$, il existe un délai τ_1 tel que $(q_0, y_0) \xrightarrow{\tau_1, a_1} (q_1, P_1)$, et $\mathsf{Coût}((q_0, y_0) \xrightarrow{\tau_1, a_1} (q_1, P_1)) = \lambda(y_0)$. Notons que $\lambda(y_0)$ assure que toutes les positions intermédiaires menant à (q_1, P_1) satisfont $\varphi \vee \psi$. Comme $(q_1, P_1) \xrightarrow{\kappa_2} (q_2, P_2)$ et $c_2 \in \kappa_2$, il existe un point $(q_1, y_1) \in P_1$ et un délai τ_2 tels que $(q_1, y_1) \xrightarrow{\tau_2, a_2} (q_2, P_2)$, et $\mathsf{Coût}((q_1, y_1) \xrightarrow{\tau_2, a_2} (q_2, P_2)) = c_2$. Ici aussi toutes les positions intermédiaires menant à (q_2, P_2) satisfont $\varphi \vee \psi$.

Nous avons construit une exécution $(q_0, y_0) \xrightarrow{\tau_1, a_1} (q_1, y_1) \xrightarrow{\tau_2, a_2} (q_2, P_2)$. Nous itérons cette construction et construisons une exécution réelle $(q_0, y_0) \xrightarrow{\tau_1, a_1} (q_1, y_1) \cdots \xrightarrow{\tau_{n-1}, a_{n-1}} (q_{n-1}, P_{n-1})$ telle que toutes les positions intermédiaires menant à (q_{n-1}, P_{n-1}) satisfont $\varphi \vee \psi$. Pour la transition finale $(q_{n-1}, P_{n-1}) \xrightarrow{\kappa_n} (q_n, P_n, \mathsf{final})$, deux cas sont possibles : κ_n est soit de la forme $\kappa^{\varphi\vee\psi}_{(q,P)\to(q',P')}$, soit de la forme $\kappa^{\varphi\vee\psi}_{(q,P\rightsquigarrow P')}$. Dans le premier cas, comme $(q_{n-1}, P_{n-1}) \xrightarrow{\kappa_n} (q_n, P_n)$ et $c_n \in \kappa_n$, par définition de $\kappa_n = \kappa^{\varphi\vee\psi}_{(q_{n-1},P_{n-1})\to(q_n,P_n)}$, nous pouvons trouver $(q_{n-1}, y_{n-1}) \in P_{n-1}$, $(q_n, y_n) \in P_n$ et un délai τ_n tels que $(q_{n-1}, y_{n-1}) \xrightarrow{\tau_n, a_n} (q_n, P_n)$, $\mathsf{Coût}((q_{n-1}, y_{n-1}) \xrightarrow{\tau_n, a_n} (q_n, y_n)) = c_n$, (q_n, y_n) satisfait ψ et toutes les positions intermédiaires menant à (q_n, y_n) satisfont $\varphi \vee \psi$. Dans le second cas, en utilisant la définition de $\kappa_n = \kappa^{\varphi\vee\psi}_{(q_{n-1},P_{n-1})\rightsquigarrow(q_n,P_n)}$, nous pouvons construire une transition $(q_{n-1}, y_{n-1}) \xrightarrow{\tau_n} (q_n, P_n)$ satisfaisant les propriétés demandées. Cela achève la preuve de cette implication.

La réciproque peut être prouvée d'une manière similaire. □

A partir de la proposition précédente nous déduisons le résultat suivant.

Proposition 4.3.4 *Soit $q_0 \in Q$, $P_0 \in \mathcal{P}$ et $y_0 \in P_0$. Alors $(q_0, y_0) \models \mathbf{E}\,\varphi\,\mathbf{U}_{\sim c}\,\psi$ si, et seulement s'il existe une exécution $\xi = (q_0, P_0, \mathit{init}) \xrightarrow{\lambda(y)} (q_1, P_1) \xrightarrow{\kappa_1} \cdots \xrightarrow{\kappa_n} (q_n, P_n, \mathit{final})$ de $\mathcal{G}_{\varphi,\psi}$, et $\zeta \in M$ tels que $\zeta \in K_\xi(y_0)$ et $\zeta \sim c$.*

Nous prouvons, sous réserve que la structure sous-jacente soit archimédienne, que nous pouvons borner la longueur des exécutions témoins dans le graphe $\mathcal{G}_{\varphi,\psi}$. Cette hypothèse n'a pas été utilisée jusqu'alors, mais elle est nécessaire dans la preuve de la proposition suivante.

Pour tout $(m, c) \in M^2$ tel que $m > 0$, nous notons $\left[\frac{c}{m}\right]$ le plus petit entier k tel que $k \cdot m > c$.

Proposition 4.3.5 *Supposons* $(q_0, y_0) \models \mathbf{E}\, \varphi\, \mathbf{U}_{\sim c}\, \psi$, *et posons* $N = 2 + (|Q| \cdot |\mathcal{P}| + 1) \cdot \left(\left[\frac{c}{m}\right] + 2\right)$ *où* m *est la plus petite constante positive apparaissant dans un intervalle de poids étiquetant les transitions de* $\mathcal{G}_{\varphi,\psi}$*. Alors il existe une exécution* $\xi = (q_0, P_0, \textit{init}) \xrightarrow{\lambda(y)} (q_1, P_1) \xrightarrow{\kappa_1} \cdots \xrightarrow{\kappa_n} (q_n, P_n, \textit{final})$ *de* $\mathcal{G}_{\varphi,\psi}$ *telle que* $y_0 \in P_0$*,* $n \leq N$*, et* $\zeta \in M$ *tel que* $\zeta \in K_\xi(y_0)$ *et* $\zeta \sim c$*.*

Preuve. Nous introduisons tout d'abord la notion de *réalisation* d'une exécution. Soit $\xi = (q_0, P_0, \mathsf{init}) \xrightarrow{\lambda(y)} (q_1, P_1) \xrightarrow{\kappa_1} \cdots \xrightarrow{\kappa_n} (q_n, P_n, \mathsf{final})$ une exécution de $\mathcal{G}_{\varphi,\psi}$. Par o-minimalité, l'ensemble des coûts étiquetant une transition est une union finie de points ou d'intervalles ouverts ; une *réalisation* de ξ est un choix, pour chaque transition de ξ, d'un intervalle (ou point) des coûts étiquetant cette transition (nous notons $\mathcal{I}$ la fonction de choix et $\xi[\mathcal{I}]$ la réalisation). Une *réalisation raisonnable* de ξ est une réalisation telle qu'aucun cycle n'est étiqueté par le seul point $\{0\}$. Une réalisation non raisonnable n'est pas intéressante car les $\{0\}$-cycles peuvent être supprimés.

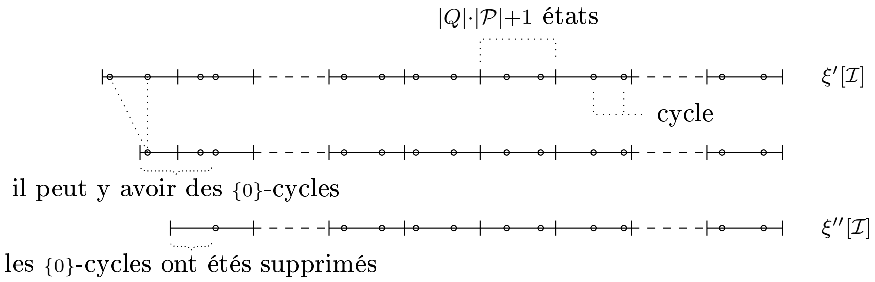


FIG. 4.6 – La construction du témoin

Nous prouvons maintenant la proposition. Soit $q_0 \in Q$, $P_0 \in \mathcal{P}$, $y_0 \in P_0$, et supposons que $(q_0, y_0) \models \mathbf{E}\, \varphi\, \mathbf{U}_{\sim c}\, \psi$. Par la proposition 4.3.4, il existe une exécution $\xi' = (q'_0, P'_0, \mathsf{init}) \xrightarrow{\lambda'(y)} (q'_1, P'_1) \xrightarrow{\kappa'_1} \cdots \xrightarrow{\kappa'_k} (q'_k, P'_k, \mathsf{final})$ de $\mathcal{G}_{\varphi,\psi}$, il existe $\zeta' \in M$ tel que $\zeta' \in K_{\xi'}(y_0)$ et $\zeta' \sim c$. À partir de ξ' nous construisons une exécution $\xi = (q_0, P_0, \mathsf{init}) \xrightarrow{\lambda(y)} (q_1, P_1) \xrightarrow{\kappa_1} \cdots \xrightarrow{\kappa_n} (q_n, P_n, \mathsf{final})$ avec $n \leq N$, $(q_0, P_0) = (q'_0, P'_0)$ et $(q_n, P_n) = (q'_k, P'_k)$ telle qu'il existe $\zeta \in M$ avec $\zeta \in K_\xi(y_0)$ et $\zeta \sim c$.

Nous supposons qu'il existe une réalisation raisonnable de ξ' : si ce n'est pas le cas il y a des cycles de ξ' dans lesquels tous les κ'_i sont réduits à $\{0\}$; nous supprimons alors ces cycles, et obtenons une exécution ξ'_1 pour laquelle le coût accumulé est ξ', et nous faisons le reste de la preuve avec ξ'_1 au lieu de ξ.

Si la longueur de $\xi'[\mathcal{I}]$ est inférieure à N nous avons fini. Sinon ξ' contient au moins $\left(\left[\frac{c}{m}\right] + 2\right)$ cycles simples (voir la figure 4.6), chacun étant étiqueté par un intervalle de coûts (accumulé) dont la borne droite est supérieure ou égale à m. Donc $K_{\xi'[\mathcal{I}]}(y_0) = \langle a, b \rangle$ avec $b \geq c + 2m$. Par hypothèse, il existe $\zeta' \in \langle a, b \rangle$ tel que $\zeta' \sim c$.

Nous supprimons alors le premier cycle simple de l'exécution et supprimons les $\{0\}$-cycles qui ont pû être créés par cette opération (voir la figure 4.6). Nous obtenons une exécution $\xi''[\mathcal{I}]$ strictement plus courte. Notons que $K_{\xi''[\mathcal{I}]}(y_0)$ peut être different de $K_{\xi[\mathcal{I}]}(y_0)$; en effet comme nous avons supprimé certains cycles, $K_{\xi''[\mathcal{I}]}(y_0) = \langle a', b' \rangle$ avec $a' \leq a$ et $b' \leq b$. Néanmoins $\xi''[\mathcal{I}]$ contient au moins $\left[\frac{c}{m}\right]$ cycles simples donc $b' > c$ (comme précédemment). Cela prouve qu'il existe $\zeta'' \in K_{\xi''[\mathcal{I}]}(y_0)$ avec $\zeta'' \sim c$. Nous itérons ce processus jusqu'à obtenir une exécution ξ de longueur inférieure ou égale à N ayant cette propriété.

$\square$

En appliquant la proposition précédente, nous pouvons construire une formule du premier ordre qui vérifie si un état (q, y) satisfait la formule de WCTL $\mathbf{E}\,\varphi\,\mathbf{U}_{\sim c}\,\psi$, et donc obtenir le corollaire suivant.

Corollaire 4.3.6 *Soit $\mathcal{M} = \langle M, +, 0, 1, <, \cdots \rangle$ une structure o-minimale telle que $\mathcal{M}$ est archimédienne et $\mathsf{Th}(\mathcal{M})$ est décidable. Soit $(\mathcal{H}, \mathsf{Coût})$ un $\mathcal{M}$-automate o-minimal pondéré, soit φ et ψ deux formules de $\mathsf{WCTL}_{\mathcal{M}}$. Supposons que nous ayons construit deux partitions finies et définissables $\mathcal{P}_\varphi$ et $\mathcal{P}_\psi$ pour φ et ψ respectivement, alors nous pouvons construire une partition finie et définissable pour la formule $\mathbf{E}\,\varphi\,\mathbf{U}_{\sim c}\,\psi$.*

Preuve. Par les propositions 4.3.4 et 4.3.5, nous déduisons facilement qu'un état (q, y) de $\mathcal{H}$ satisfait la formule $\mathbf{E}\,\varphi\,\mathbf{U}_{\sim c}\,\psi$ si, et seulement s'il existe une exécution réelle de longueur au plus N à partir de (q, y), qui satisfait la formule $\varphi\,\mathbf{U}_{\sim c}\,\psi$.

Pour $n \in \mathbb{N}$, il est possible d'écrire une formule du premier ordre $\theta_n(\cdot)$ telle que $\mathcal{M} \models \theta_n(q, y)$ si, et seulement s'il existe une exécution réelle de longueur exactement n à partir de (q, y), qui satisfait la formule $\varphi\,\mathbf{U}_{\sim c}\,\psi$. Nous obtenons donc l'équivalence suivante :

$$(q, y) \models \mathbf{E}\,\varphi\,\mathbf{U}_{\sim c}\,\psi \text{ si, et seulement si } \mathcal{M} \models \bigvee_{n=0}^{N} \theta_n(q, y).$$

La partition désirée est celle obtenue en raffinant la partition finie et définissable $\mathrm{Suf}(\mathcal{P}_\varphi \sqcap \mathcal{P}_\psi)$ avec la formule $\bigvee_{n=0}^{N} \theta_n(q, y)$. Cette nouvelle partition est clairement définissable et contient au plus $2 \cdot |\mathrm{Suf}(\mathcal{P}_\varphi \sqcap \mathcal{P}_\psi)|$ pièces. $\square$

4.3.3 Partition pour $\mathbf{A}\,\varphi\,\mathbf{U}_{\sim c}\,\psi$

Nous expliquons maintenant comment construire une partition pour la formule $\mathbf{A}\,\varphi\,\mathbf{U}_{\sim c}\,\psi$ en supposant que nous ayons construit des partitions finies et définissables $\mathcal{P}_\varphi$ et $\mathcal{P}_\psi$ pour φ et ψ respectivement.

Il sera utile dans la suite de pouvoir décider s'il existe une exécution infinie de coût toujours inférieur à une constante c, c'est le but du lemme suivant.

Lemme 4.3.7 *Soit (q, y) un état de $\mathcal{H}$ et c une constante définissable. Il est décidable de déterminer s'il existe une exécution infinie à partir de (q, y) de coût strictement inférieur (resp. inférieur ou égal) à c.*

De plus, s'il n'existe pas de telle exécution, alors il existe un indice calculable n_0, ne dépendant pas de (q, y), tel que toute exécution de longueur supérieure ou égale à n_0 a un coût supérieur ou égal (resp. strictement supérieur à c).

Preuve. Nous considérons le cas $< c$, le cas $\leq c$ est analogue. Nous établissons tout d'abord une caractérisation des états à partir desquels il existe une exécution infinie de coût $< c$ et utilisons alors cette caractérisation pour montrer le lemme.

Soit $\mathcal{G}$ le graphe de la sous-section 4.3.2 où $\varphi = \psi = \top$. Nous appellons $\langle 0, \_ \rangle$-cycle un cycle de $\mathcal{G}$ où toutes les étiquettes sont de la forme $]0, a[$, $]0, a]$, $[0, a[$ ou $[0, a]$ pour $a \in M$.

Il existe une exécution infinie de coût $< c$ à partir de (q, y) si, et seulement s'il existe

$$\begin{aligned} \xi = (q_0, P_0, \mathsf{init}) \xrightarrow{\lambda(y)} (q_1, P_1) \xrightarrow{\kappa_1} \cdots \xrightarrow{\kappa_{n-1}} (q_{n-1}, P_{n-1}) \xrightarrow{\kappa_n} (q_n, P_n) \\ \text{une exécution de } \mathcal{G} \text{ avec } q_0 = q, y \in P_0 \text{ et } K_\xi(y) \cap [0, c[\neq \emptyset \\ \text{et } \exists \xi' = (q'_0, P'_0) \xrightarrow{\kappa'_1} (q'_1, P'_1) \xrightarrow{\kappa'_2} \cdots \xrightarrow{\kappa'_k} (q'_k, P'_k) \\ \text{un } \langle 0, \_ \rangle\text{-cycle avec } (q'_0, P'_0) = (q'_k, P'_k) = (q_n, P_n). \end{aligned} \tag{4.1}$$

Nous prouvons maintenant la caractérisation (4.1). Soit $m > 0$ la plus petite constante strictement positive apparaissant dans un intervalle de poids étiquetant les transitions de $\mathcal{G}$. Posons $n_0 = |\mathcal{P}| \cdot \left[\frac{c}{m}\right] + 2$.

Implication directe. Soit $\rho = (q_0, y_0) \xrightarrow{\tau_1, a_1} \cdots$ une exécution infinie de coût $< c$ à partir de (q, y). Par la proposition 4.3.3, il existe une exécution $\xi = (q_0, P_0, \mathsf{init}) \xrightarrow{\lambda(y)} (q_1, P_1) \xrightarrow{\kappa_1} \cdots \xrightarrow{\kappa_{n-1}} (q_{n-1}, P_{n-1}) \xrightarrow{\kappa_n} (q_n, P_n, \mathsf{final})$ de $\mathcal{G}$ telle que $\mathsf{Coût}(\rho) \in K_\xi(y)$. Comme $\mathsf{Coût}(\rho) < c$, par définition de n_0, il existe un $\langle 0, \_ \rangle$-cycle dans ξ : en effet divisons ξ en $\left[\frac{c}{m}\right]$ segments de longueur $|\mathcal{P}|$, chaque segment produit un cycle dans $\mathcal{G}$, et un de ces cycles doit être un $\langle 0, \_ \rangle$-cycle sinon $\mathsf{Coût}(\rho)$ serait $\geq c$. De plus le coût pour atteindre ce $\langle 0, \_ \rangle$-cycle doit être nécessairement $< c$ car c'est le cas pour ξ en entier.

Implication réciproque. Soient ξ et ξ' comme dans (4.1). Par définition de ξ et la proposition 4.3.3, il existe une exécution $\rho = (q_0, y_0) \xrightarrow{\tau_1, a_1} \cdots \xrightarrow{\tau_n, a_n} (q_n, y_n)$ avec $\forall i \; y_i \in P_i$ et $\mathsf{Coût}(\rho) < c$.

Par définition de ξ' et la proposition 4.3.3, $\forall \varepsilon > 0 \; \exists \rho'_\varepsilon = (q'_0, y'_0) \xrightarrow{\tau'_1, a'_1} \cdots \xrightarrow{\tau'_k, a'_k} (q'_k, y'_k)$ avec $\forall i \; y'_i \in P'_i$ et $\mathsf{Coût}(\rho_\varepsilon) < \varepsilon$.

Soit $\varepsilon' = c - \mathsf{Coût}(\rho)$. En concaténant ρ et $\rho_{\frac{\varepsilon'}{2^{n+1}}}$ pour $n \geq 1$, nous obtenons une exécution infinie à partir de (q, y) de coût $< c$. Cela achève la preuve de la caractérisation (4.1).

En utilisant (4.1) il est facile de vérifier s'il existe une exécution à partir de (q, y) de coût $< c$ en considérant le graphe fini $\mathcal{G}$. De plus, s'il n'existe pas de telle exécution, soit ρ une exécution de longueur supérieure ou égale à n_0. Soit ξ l'exécution correspondante de $\mathcal{G}$ obtenue par la proposition 4.3.3. Soit ξ contient un $\langle 0, \_ \rangle$-cycle alors nécessairement $\mathsf{Coût}(\xi) \geq c$ (si ce n'était pas le cas, par (4.1) il existerait une exécution infinie de coût $< c$). Soit ξ ne contient pas de $\langle 0, \_ \rangle$-cycle, alors $\mathsf{Coût}(\rho) \geq \frac{n_0}{|\mathcal{P}|} m \geq c$ (divisons ξ en segments de longueur $|\mathcal{P}|$, chaque segment a un coût supérieur ou égal à m).

Remarquons qu'en utilisant (4.1) nous pouvons donner une formule du premier ordre $\theta(q, y)$ exprimant l'existence d'une exécution infinie à partir de (q, y) de coût $< c$. $\square$

Notons que $\mathbf{E}\,\mathbf{F}_{\sim c}\,\varphi$ est un cas particulier de $\mathbf{E}\,\varphi_1\,\mathbf{U}_{\sim c}\,\varphi_2$, nous pouvons donc appliquer les résultats de la sous-section 4.3.2 pour construire une partition pour $\mathbf{E}\,\mathbf{F}_{\sim c}\,\varphi$ à partir d'une partition pour φ. Nous expliquons maintenant comment construire une partition pour $\mathbf{E}\,\mathbf{G}_{\sim c}\,\varphi$ car cela sera utile par la suite.

Lemme 4.3.8 *Soit φ une formule de $\mathsf{WCTL}_{\mathcal{M}}$ et c une constante définissable. S'il existe une partition finie, définissable et calculable pour φ alors il existe une partition finie, définissable et calculable pour $\mathbf{E}\,\mathbf{G}_{\sim c}\,\varphi$.*

Preuve.

Nous prouvons tout d'abord le cas $> c$; les cas $= c$ et $\geq c$ sont analogues.

Soit n_0 comme dans le lemme 4.3.7. Nous utiliserons la caractérisation suivante :

$(q, y) \models \mathbf{E}\,\mathbf{G}_{>c}\,\varphi$ si, et seulement si

$$\begin{array}{l}\text{il existe une exécution à partir de } (q,y) \text{ de coût } \leq c \\ \text{ou } \exists\rho \text{ à partir de } (q,y) \text{ de longueur } n_0 + |\mathcal{P}| + 1 \text{ telle que}^{8} \rho \models \mathbf{G}_{>c}\,\varphi\end{array} \tag{4.2}$$

L'implication directe est évidente.

Il est aussi évident que s'il existe une exécution infinie à partir de (q, y) de coût $\leq c$ alors $(q, y) \models \mathbf{E}\,\mathbf{G}_{>c}\,\varphi$. Supposons maintenant qu'il n'existe pas d'exécution infinie à partir de (q, y) de coût $\leq c$ mais qu'il existe une exécution ρ à partir de (q, y) de longueur $n_0 + |\mathcal{P}| + 1$ telle que $\rho \models \mathbf{G}_{>c}\,\varphi$, nous devons montrer que $(q, y) \models \mathbf{E}\,\mathbf{G}_{>c}\,\varphi$.

Par la proposition 4.3.3, il existe $\xi = (q_0, P_0, \mathsf{init}) \xrightarrow{\lambda(y)} (q_1, P_1) \xrightarrow{\kappa_1} \cdots \xrightarrow{\kappa_{n_0+|\mathcal{P}|}} (q_{n_0+|\mathcal{P}|}, P_{n_0+|\mathcal{P}|}) \xrightarrow{\kappa_{n_0+|\mathcal{P}|+1}} (q_{n_0+|\mathcal{P}|+1}, P_{n_0+|\mathcal{P}|+1}, \mathsf{final})$ une exécution correspondante avec $q_0 = q$ et $y \in P_0$. Considérons l'exécution $(q_{n_0+1}, P_{n_0+1}) \xrightarrow{\kappa_{n_0+2}} \cdots \xrightarrow{\kappa_{n_0+|\mathcal{P}|}} (q_{n_0+|\mathcal{P}|}, P_{n_0+|\mathcal{P}|})$. Il y a nécessairement une boucle dans cette exécution, et comme dans la preuve du lemme 4.3.7 nous pouvons trouver une exécution infinie ρ' qui itère cette boucle. Par le lemme 4.3.7, après l'indice n_0 le coût est toujours strictement supérieur à c donc φ est toujours satisfaite dans ρ ; comme ρ' est obtenue à partir de ρ en itérant une boucle située après l'indice n_0, φ est également toujours satisfaite dans ρ' ; donc $\rho' \models \mathbf{G}_{>c}\,\varphi$, et $(q, y) \models \mathbf{E}\,\mathbf{G}_{>c}\,\varphi$. Cela achève la preuve de la caractérisation (4.2).

Le lemme 4.3.8 et la caractérisation (4.2) montrent qu'il existe une formule du premier ordre pour $\mathbf{E}\,\mathbf{G}_{>c}\,\varphi$.

Nous expliquons maintenant comment traiter le cas $< c$; le cas $\leq c$ est similaire. Le lemme 4.3.2 donne une partition $\mathcal{P}_1$ pour $\mathbf{E}\top\mathbf{U}\,\varphi = \mathbf{E}\,\mathbf{F}\,\varphi$ et $\mathbf{E}\,\mathbf{G}\,\varphi = \neg\mathbf{A}\,\mathbf{F}\,(\neg\varphi)$. Nous donnons une caractérisation qui permet de construire une formule du premier ordre pour $\mathbf{E}\,\mathbf{G}_{<c}\,\varphi$. Soit n_0 comme dans le lemme 4.3.7, nous utilisons la caractérisation suivante, qui peut être prouvée comme la caractérisation (4.2) :

$(q, y) \models \mathbf{E}\,\mathbf{G}_{<c}\,\varphi$ si, et seulement si

$(q, y) \models \mathbf{E}\,\mathbf{G}\,\varphi$ ou $\exists\rho$ à partir de (q, y) de longueur $n_0 + |\mathcal{P}| + 1$ telle que $\rho \models \mathbf{E}\,\mathbf{G}_{<c}\,\varphi$

Ceci achève la preuve du lemme 4.3.8.

□

[8] La sémantique de WCTL sur les chemins finis est analogue à celle sur les chemins infinis définie page 58.

Remarque 4.3.9 *Dans la preuve du lemme 4.3.8 nous utilisons fortement l'hypothèse que la fonction de coût est croissante. Nous ne savons pas si le lemme 4.3.8 est vrai pour des coûts non-croissants.*

Nous sommes maintenant prêts pour construire une partition pour la formule $\mathbf{A}\,\varphi\,\mathbf{U}_{\sim c}\,\psi$. En fait nous considérerons la formule $\neg\mathbf{A}\,\varphi\,\mathbf{U}_{\sim c}\,\psi$, qui a la même partition. Nous décomposons $\neg\mathbf{A}\,\varphi\,\mathbf{U}_{\sim c}\,\psi$ en deux prédicats de chemins, et montrons que ces prédicats admettent des témoins de longueur finie (et calculable), ce qui permet de prouver l'existence d'une partition pour $\neg\mathbf{A}\,\varphi\,\mathbf{U}_{\sim c}\,\psi$.

Définition 4.3.10 *Un* prédicat de chemin Q *est une fonction* $Q : \mathsf{Exec}(\mathcal{H}) \to \{\top, \bot\}$*. Nous notons* $\rho \models Q$ *pour* $Q(\rho)$*.*

Nous définissons les deux prédicats suivants, dont la disjonction correspondra à $\neg\varphi\,\mathbf{U}_{\sim c}\,\psi$:
- $\rho \models Q_1^{\sim c}$ si, et seulement si $\rho \models \mathbf{G}_{\sim c}\,\neg\psi$
- $\rho \models Q_2^{\sim c}$ si, et seulement s'il existe une position $p \geq 0$ sur ρ telle que $\rho[p] \models (\neg\varphi) \wedge \neg\psi$, et pour toute position $0 \leq p' \leq p$ sur ρ, $\mathsf{Coût}(\rho_{\leq p'}) \sim c \Rightarrow \rho[p'] \models \neg\psi$

Pour $i \in \{1,2\}$ nous disons que $(q,y) \models \varphi_{Q_i^{\sim c}}$ s'il existe une exécution infinie ρ à partir de (q,y) telle que $\rho \models Q_i^{\sim c}$. Notons que $\varphi_{Q_i^{\sim c}}$ n'est pas nécessairement une formule de WCTL, c'est une proposition booléenne dépendant de (q,y).

Définition 4.3.11 *Un prédicat de chemin Q admet* des témoins de longueur $n \in \mathbb{N}$ *à partir de (q,y) si :*
$\exists$ une exécution infinie ρ à partir de (q,y) telle que $\rho \models Q \Leftrightarrow \exists$ une exécution ρ à partir de (q,y) de longueur $\leq n$ telle que $\rho \models Q$.

Remarque 4.3.12 *Si Q est définissable et admet des témoins d'une longueur finie et calculable alors il existe une partition finie, définissable et calculable pour la propriété $(q,y) \models Q$ car nous pouvons construire une formule du premier ordre énumérant tous les témoins comme dans la preuve du corollaire 4.3.6.*

Proposition 4.3.13 $(q,y) \models \neg\mathbf{A}\,\varphi\,\mathbf{U}_{\sim c}\,\psi$ *si, et seulement si* $(q,y) \models \varphi_{Q_1^{\sim c}}$ *ou* $(q,y) \models \varphi_{Q_2^{\sim c}}$*.*

Preuve. $(q,y) \models \neg\mathbf{A}\,\varphi\,\mathbf{U}_{\sim c}\,\psi$ signifie qu'il existe une exécution infinie ρ à partir de (q,y) telle que $\rho \not\models \varphi\,\mathbf{U}_{\sim c}\,\psi$. La proposition sera donc une conséquence de la propriété suivante :

$$\rho \not\models \varphi\,\mathbf{U}_{\sim c}\,\psi \text{ ssi } \rho \models Q_1^{\sim c} \text{ ou } \rho \models Q_2^{\sim c}. \tag{4.3}$$

Remarquons tout d'abord que $\rho \not\models \varphi\,\mathbf{U}_{\sim c}\,\psi$ est équivalent à

$$\forall p_1 \geq 0, (\rho[p_1] \models \psi \text{ et } \mathsf{Coût}(\rho_{\leq p_1}) \sim c) \Rightarrow \exists p_1^{\text{tém}} < p_1 \text{ sur } \rho \text{ avec } \rho[p_1^{\text{tém}}] \models (\neg\varphi) \wedge \neg\psi \tag{4.4}$$

C'est-à-dire que pour toute position p_1 où ψ et $\mathsf{Coût}(\rho_{\leq p_1}) \sim c$ sont vérifiées, il existe un témoin pour $(\neg\varphi) \wedge \neg\psi$ à une position $p_1^{\text{tém}} < p_1$.

Nous montrons maintenant (4.3) par double implication, en utilisant la caractérisation (4.4).
Implication réciproque. Pour $i \in \{1,2\}$, nous supposons que $\rho \models Q_i^{\sim c}$ et montrons que $\rho \not\models \varphi\,\mathbf{U}_{\sim c}\,\psi$.

– $\rho \models Q_1^{\sim c} \Rightarrow \rho \not\models \varphi \, \mathbf{U}_{\sim c} \, \psi$:
  $(\rho[p_1] \models \psi)$ et $(\mathsf{Coût}(\rho_{\leq p_1}) \sim c)$ ne sont jamais vraies simultanément donc il n'y a rien à vérifier.
– $\rho \models Q_2^{\sim c} \Rightarrow \rho \not\models \varphi \, \mathbf{U}_{\sim c} \, \psi$:
  Soit p comme dans la définition de $Q_2^{\sim c}$. Pour toute position $p_1 \leq p$ sur ρ, $(\rho[p_1] \models \psi)$ et $\mathsf{Coût}(\rho_{\leq p_1} \sim c)$ ne sont jamais vraies simultanément donc il n'y a rien à vérifier. Pour toute position $p_1 > p$, p est un témoin pour $(\neg\varphi) \wedge \neg\psi$.

Implication directe. Supposons $\rho \not\models \varphi \, \mathbf{U}_{\sim c} \, \psi$. Si $(\rho[p_1] \models \psi)$ et $\mathsf{Coût}(\rho_{\leq p_1} \sim c)$ ne sont jamais vraies simultanément alors $\rho \models Q_1^{\sim c}$.

Sinon nous pouvons définir $p_0 = \inf\{p_1 \mid \rho[p_1] \models \psi$ et $\mathsf{Coût}(\rho_{\leq p_1}) \sim c\}$. Si cet infimum est en fait un minimum, c'est-à-dire $\rho[p_0] \models \psi$ et $\mathsf{Coût}(\rho_{\leq p_0}) \sim c$, alors par hypothèse il existe un témoin p pour $(\neg\varphi) \wedge \neg\psi$ avec $p < p_0$. Alors pour toute position $p_1 \leq p$, $\mathsf{Coût}(\rho_{\leq p_1} \sim c)$ implique $\rho[p_1] \models \neg\psi$, donc $\rho \models Q_2^{\sim c}$.

Soit $p_0 = (n_0, t_0)$. Si l'infimum n'est pas un minimum alors il existe des positions (n_0, t) avec $t > t_0$ arbitrairement proche de t_0 vérifiant $\rho[(n_0, t)] \models \psi$ et $\mathsf{Coût}(\rho_{\leq (n_0,t)}) \sim c$. Il existe une formule du premier ordre exprimant $E = \{t > t_0 \mid \rho[(n_0, t)] \models \psi$ et $\mathsf{Coût}(\rho_{\leq (n_0,t)}) \sim c\}$. Donc par o-minimalité, E est une union finie de points et d'intervalles ouverts. Comme t_0 n'appartient pas à E, E doit contenir un intervalle de la forme $]t_0, t'[$ avec $t' > t_0$. Alors il doit exister une position $p \leq p_0$ telle que $\rho[p] \models (\neg\varphi) \wedge \neg\psi$ (sinon ρ vérifierait[9] $\varphi \, \mathbf{U}_{\sim c} \, \psi$). Pour toute position $p_1 \leq p$, $\mathsf{Coût}(\rho_{\leq p_1} \sim c)$ implique $\rho[p_1] \models \neg\psi$, donc $\rho \models Q_2^{\sim c}$. □

Remarque 4.3.14 *La proposition 4.3.13 est à notre connaissance la première fois qu'une caractérisation de* $\neg\mathbf{A} \, \varphi \, \mathbf{U}_{\sim c} \, \psi$ *est prouvée de manière rigoureuse.*

Notons que nous avons utilisé l'hypothèse d'o-minimalité. En effet sans cette hypothèse la caractérisation n'est pas correcte : si la véracité de ψ *peut varier infiniment souvent au voisinage d'un point,* $\neg\mathbf{A} \, \varphi \, \mathbf{U}_{\sim c} \, \psi$ *peut être vraie mais pas la caractérisation.*

Considérons l'exécution ρ *décrite sur la figure 4.7 :* φ *est vraie pour* $x \leq 1$ *et* ψ *est vraie aux dates* $x = 1 + \frac{1}{2^n}$ *pour* $n \in \mathbb{N}$. $\rho \models \neg(\varphi \, \mathbf{U} \, \psi)$, *mais elle ne satisfait pas la caractérisation. Notons que même la caractérisation classique dans* LTL, $\neg(\varphi \, \mathbf{U} \, \psi) \equiv \mathbf{G} \, (\neg\psi) \bigvee ((\neg\psi) \, \mathbf{U} \, ((\neg\varphi) \wedge \neg\psi))$ *n'est pas vérifiée par ce modèle.*

La proposition 4.3.13 est cependant robuste, car dans beaucoup de logiques temporisées les modèles sont supposés à variation finie.

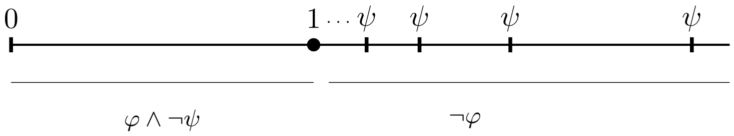


FIG. 4.7 – Un modèle variant infiniment souvent

Proposition 4.3.15 *Soit* $\mathcal{M} = \langle M, +, 0, 1, <, \cdots \rangle$ *une structure o-minimale telle que* $\mathcal{M}$ *est archimédienne et* $\mathsf{Th}(\mathcal{M})$ *est décidable. Soit* $(\mathcal{H}, \mathsf{Coût})$ *un* $\mathcal{M}$*-automate o-minimal pondéré, soit* φ *et* ψ *deux formules de* $\mathsf{WCTL}_{\mathcal{M}}$. *Supposons que nous ayons construit deux partitions finies et définissables* $\mathcal{P}_\varphi$ *et* $\mathcal{P}_\psi$ *pour* φ *et* ψ *respectivement, alors nous pouvons construire une partition finie et définissable pour la formule* $\mathbf{A} \, \varphi \, \mathbf{U}_{\sim c} \, \psi$.

[9] Rappelons que dans WCTL, $\varphi \, \mathbf{U}_{\sim c} \, \psi$ est équivalente à $(\varphi \vee \psi) \, \mathbf{U}_{\sim c} \, \psi$.

Preuve. Nous distinguons différents cas pour $\sim c$ et montrons que pour tout i, soit $\varphi_{Q_i^{\sim c}}$ peut être exprimée par une formule de WCTL pour laquelle nous savons qu'il existe une partition finie, soit $Q_i^{\sim c}$ admet des témoins d'une longueur bornée définissable.

En effet, si $Q_i^{\sim c}$ admet des témoins d'une longueur bornée définissable, nous pouvons écrire une formule énumérant tous ces chemins finis et tester si $\varphi_{Q_i^{\sim c}}$ est satisfaisable sur ceux-ci, définissant ainsi une partition finie, définissable et calculable pour $Q_i^{\sim c}$. Cela montrera donc qu'il existe une partition finie, définissable et calculable pour $\neg\mathbf{A}\,\varphi\,\mathbf{U}_{\sim c}\,\psi$.

Nous traitons le cas $\neg\mathbf{A}\,\varphi\,\mathbf{U}_{=c}\,\psi$, les autres cas sont similaires.

- Le lemme 4.3.8 montre qu'il existe une partition finie, définissable et calculable pour $Q_1^{=c}$.
- Nous montrons le lemme suivant :

  Lemme 4.3.16 *Si* $(q,y) \not\models \varphi_{Q_1^{=c}}$ *alors* $Q_2^{=c}$ *admet des témoins d'une longueur finie définissable à partir de* (q,y).

  Notons que ce lemme est suffisant pour trouver une partition pour $\neg\mathbf{A}\,\varphi\,\mathbf{U}_{=c}\,\psi$ car nous pouvons considérer les prédicats Q_1, $(\neg Q_1^{=c}) \wedge Q_2^{=c}$ (au lieu de $Q_1^{=c}$ et $Q_2^{=c}$).

  Preuve.[Preuve du lemme] Soit n_0 comme dans le lemme 4.3.7. Soit $n = n_0 + |\mathcal{P}|$.

  Nous devons montrer qu'il existe une exécution infinie ρ à partir de (q,y) telle que $\rho \models Q_2^{=c}$ si, et seulement s'il existe une exécution finie ρ à partir de (q,y) de longueur $\leq n$ telle que $\rho \models Q_2^{=c}$.

  L'implication réciproque est évidente. Supposons maintenant qu'il existe une exécution infinie ρ à partir de (q,y) telle que $\rho \models Q_2^{=c}$, et construisons une exécution finie de longueur $\leq n$. Nous avons qu'il existe une position p sur ρ telle que $\rho[p] \models (\neg\varphi) \wedge \neg\psi$, et pour toute position $0 \leq p' \leq p$ sur ρ, $\mathsf{Coût}(\rho_{\leq p'}) = c \Rightarrow \rho[p'] \models \neg\psi$.

  Si la position p est située avant l'indice n alors $\rho_{\leq p}$ est notre témoin. Si ce n'est pas le cas, par le lemme 4.3.7 nous avons que $\mathsf{Coût}(\rho_{\leq n_0}) = c$ donc $\forall (n_0, 0) \leq p' \leq p$, $\mathsf{Coût}(\rho_{\leq p'}) = c$. Cela implique que $\rho_{\geq (n_0,0)} \models (\neg\psi)\,\mathbf{U}\,((\neg\psi)\wedge\neg\varphi)$. Comme nous l'avons remarqué dans la preuve du lemme 4.3.2, il existe donc une exécution ρ' de longueur $\leq |\mathcal{P}|$ à partir de $\rho[(n_0,0)]$ telle que $\rho' \models (\neg\psi)\,\mathbf{U}\,((\neg\psi)\wedge\neg\varphi)$. En concaténant $\rho_{\leq (n_0,0)}$ et ρ' nous obtenons une exécution de longueur $\leq n_0 + |\mathcal{P}|$ satisfaisant Q_2. □

Ceci achève la preuve de la proposition 4.3.15.

□

Deuxième partie

Décidabilité et contrôle de MTL

Chapitre 5

Logiques temporelles : définitions et décidabilité

Les logiques temporisées ont été introduites au début des années 90 [AH89, Koy90], elles étendent naturellement les logiques temporelles [Pnu77] avec des contraintes temps-réel sur les modalités. Il existe deux sémantiques pour les logiques temporisées, la sémantique d'actions où les formules sont évaluées lors d'observations discrètes (les actions d'un mot temporisé), et la sémantique continue où les formules sont évaluées de manière continue. Nous présentons différentes logiques temporisées pour ces deux sémantiques : TPTL qui utilise des horloges de formule pouvant être remises à zéro et comparées à des constantes, MTL qui est une extension de LTL avec des contraintes sur l'opérateur *until*, MITL qui est un fragment de MTL où les intervalles ponctuels sont interdits, et Safety-MTL qui est un fragment de MTL exprimant des propriétés de sûreté. Nous définissons les extensions avec passé de ces logiques et présentons la notion d'expressivité d'une logique.

Les problèmes de satisfaisabilité et de *model-checking* de TPTL ont été prouvés indécidables [AH89] pour les deux sémantiques. Cette preuve d'indécidabilité s'applique également à MTL pour la sémantique continue. Cependant les problèmes de satisfaisabilité et de *model-checking* de MTL sont décidables pour la sémantique d'actions sur les mots finis [OW05], même s'ils sont indécidables sur les mots infinis [OW06a].

La preuve originale d'indécidabilité de TPTL [AH89] utilise une réduction de l'arrêt de machines à deux compteurs. Son principe est naturel mais sa formalisation et preuve de correction sont assez techniques [DP07]. Nous proposons une preuve différente basée sur la réduction de l'accessibilité des automates communicants. Cela nous permet d'obtenir une preuve unifiée de l'indécidabilité du problème de satisfaisabilité de plusieurs logiques temporisées : TPTL, TPTL+Past et MTL+Past pour les deux sémantiques, et MTL pour la sémantique continue. Cette technique a été utilisée dans [OW05] pour établir la complexité du problème de satisfaisabilité de MTL pour la sémantique d'actions.

5.1 Définitions

Nous définissons dans cette section les logiques que nous considérerons et discutons des problèmes d'expressivité.

5.1.1 La logique TPTL

La logique propositionnelle temporisée [AH94, Ras99], notée TPTL (pour *Timed Propositional Temporal Logic*), est une extension de LTL [Pnu77] utilisant des variables supplémentaires (appelées *horloges*) dans les formules. Dans ce chapitre, AP désigne un ensemble infini dénombrable de variables propositionnelles. La syntaxe de TPTL est la suivante :

$$\varphi ::= a \mid \varphi \wedge \varphi \mid \neg\varphi \mid \varphi \,\mathbf{U}\, \varphi \mid x \sim c \mid x.\varphi$$

où $a \in \mathsf{AP}$, x est une horloge, $c \in \mathbb{Q}$ et $\sim\, \in \{\leq, <, =, >, \geq\}$.

Il y a deux sémantiques pour TPTL : la sémantique *continue* qui interprète les formules de TPTL sur les séquences temporisées, et la sémantique *d'actions* qui interprète les formules sur les mots temporisés. Cette dernière sémantique est différente car les formules ne peuvent être interprétées qu'aux points où une action a lieu.

Les deux sémantiques ont été considérées dans la littérature, et les résultats de décidabilité dépendent fortement de la sémantique choisie. Par exemple, Ouaknine et Worrell [OW05] ont montré que la satisfaisabilité de MTL (un fragment de TPTL que nous définirons plus tard) est décidable pour la sémantique d'actions, alors que MTL est indécidable pour la sémantique continue.

Sémantique continue. Pour la sémantique continue, les modèles sont des séquences temporisées finies ou infinies κ, et les formules sont évaluées à une date $t \in \mathbb{T}$ avec une interprétation $\mathbb{I}\colon X \to \mathbb{T}$ (où X est l'ensemble des horloges de la formule). Nous notons $\mathbf{0}$ l'interprétation qui associe à toute horloge la valeur 0. Pour une interprétation $\mathbb{I}$, nous utilisons la notation $\mathbb{I}[x \mapsto t]$ pour l'interprétation qui envoie x sur t et est identique à $\mathbb{I}$ pour toutes les autres variables. La relation de satisfaction, notée $\kappa, t, \mathbb{I} \models_c \varphi$ (nous omettons parfois l'indice c, et écrivons $\kappa, t, \mathbb{I} \models \varphi$, quand c'est clair dans le contexte), est définie inductivement comme suit :

$$\begin{array}{rcl}
\kappa, t, \mathbb{I} \models_c p & \text{si} & p \in \kappa(t) \\
\kappa, t, \mathbb{I} \models_c \varphi_1 \wedge \varphi_2 & \text{si} & \kappa, t, \mathbb{I} \models_c \varphi_1 \text{ et } \kappa, t, \mathbb{I} \models_c \varphi_2 \\
\kappa, t, \mathbb{I} \models_c \neg\varphi & \text{si} & \kappa, t, \mathbb{I} \not\models_c \varphi \\
\kappa, t, \mathbb{I} \models_c \varphi_1 \,\mathbf{U}\, \varphi_2 & \text{si} & \exists t' > t \text{ tel que } \kappa, t', \mathbb{I} \models_c \varphi_2 \\
 & & \quad \text{et } \forall t < t'' < t',\ \kappa, t'', \mathbb{I} \models_c \varphi_1 \vee \varphi_2 \ ^{1} \\
\kappa, t, \mathbb{I} \models_c x \sim c & \text{si} & t - \mathbb{I}(x) \sim c \\
\kappa, t, \mathbb{I} \models_c x.\varphi & \text{si} & \kappa, t, \mathbb{I}[x \mapsto t] \models_c \varphi
\end{array}$$

Nous écrivons $\kappa \models_c \varphi$ pour $\kappa, 0, \mathbf{0} \models_c \varphi$. L'ensemble des mots temporisés finis vérifiant une formule φ est donné par $L^*_c(\varphi) = \{\sigma \in T\Sigma^* \mid \sigma \models_c \varphi\}$. L'ensemble des mots temporisés infinis vérifiant φ est donné par $L^\omega_c(\varphi) = \{\sigma \in T\Sigma^\omega \mid \sigma \models_c \varphi\}$. Quand la sémantique est claire dans le contexte, nous notons simplement $L^*(\varphi)$ et $L^\omega(\varphi)$.

Nous interprétons $x.\varphi$ comme une remise à zéro. Remarquons aussi que la sémantique de $\mathbf{U}$ est stricte : pour satisfaire $\varphi_1 \,\mathbf{U}\, \varphi_2$, une séquence temporisée n'a pas besoin de satisfaire φ_1 ; cette sémantique est plus expressive que la sémantique non stricte (voir la section 5.1.5).

[1] Comme par exemple dans [Ras99], nous demandons aux positions intermédiaires de satisfaire $\varphi_1 \vee \varphi_2$ pour avoir une interprétation naturelle de la satisfaction quand φ_2 est vraie sur un intervalle ouvert à gauche.

Nous utilisons le *until temporisé* $\varphi_1 \mathbf{U}_I \varphi_2$ défini par $x.(\varphi_1 \mathbf{U} (x \in I \wedge \varphi_2))$. Nous ne précisons pas l'intervalle contraignant le *until* quand celui-ci est $[0, +\infty[$. Nous écrivons $\mathbf{U}_{\sim c}$ pour $\mathbf{U}_I$ avec $I = \{t \mid t \sim c\}$. Nous utiliserons les raccourcis classiques suivants : $\top$ pour $p \vee \neg p$, $\bot$ pour $\neg\top$, $\varphi_1 \Rightarrow \varphi_2$ pour $\neg\varphi_1 \vee \varphi_2$, $\mathbf{F}_I \varphi$ pour $\top \mathbf{U}_I \varphi$, et $\mathbf{G}_I \varphi$ pour $\neg(\mathbf{F}_I \neg\varphi)$. Soit $\Sigma \subseteq \mathsf{AP}$ un ensemble fini de variables propositionnelles, nous utilisons également l'opérateur *suivant* noté $\mathbf{X}_\Sigma$ (ou simplement $\mathbf{X}$ quand Σ est clair dans le contexte) défini par $\mathbf{X}_\Sigma \varphi = (\neg \bigvee_{a \in \Sigma} a) \mathbf{U} \varphi$.

Sémantique d'actions. Dans cette sémantique, les modèles sont des mots temporisés finis ou infinis σ, et la satisfaction est interprétée non plus à une date $t \in \mathbb{T}$ mais à une position $i \in \mathbb{N}$ du mot temporisé. Pour un mot temporisé $\sigma = (w, \tau)$ avec $w = (w_i)_{i \geq 0}$ et $\tau = (\tau_i)_{i \geq 0}$, et une interprétation $\mathbb{I}\colon X \to \mathbb{T}$ (où X est l'ensemble des horloges de la formule), la relation de satisfaction $\sigma, i, \mathbb{I} \models_{pw} \varphi$ est définie inductivement comme suit :

$$
\begin{array}{rcl}
\sigma, i, \mathbb{I} \models_{pw} p & \text{si} & w_i = p \\
\sigma, i, \mathbb{I} \models_{pw} \varphi_1 \wedge \varphi_2 & \text{si} & \sigma, i, \mathbb{I} \models_{pw} \varphi_1 \text{ et } \sigma, i, \mathbb{I} \models_{pw} \varphi_2 \\
\sigma, i, \mathbb{I} \models_{pw} \neg\varphi & \text{si} & \sigma, i, \mathbb{I} \not\models_{pw} \varphi \\
\sigma, i, \mathbb{I} \models_{pw} \varphi_1 \mathbf{U} \varphi_2 & \text{si} & \exists j > i \text{ tel que } \sigma, j, \mathbb{I} \models_{pw} \varphi_2 \\
 & & \quad \text{et } \forall i < k < j,\ \sigma, k, \mathbb{I} \models_{pw} \varphi_1 \\
\sigma, i, \mathbb{I} \models_{pw} x \sim c & \text{si} & \tau_i - \mathbb{I}(x) \sim c \\
\sigma, i, \mathbb{I} \models_{pw} x.\varphi & \text{si} & \sigma, i, \mathbb{I}[x \mapsto \tau_i] \models_{pw} \varphi
\end{array}
$$

Nous écrivons $\sigma \models_{pw} \varphi$ pour $\rho, 0, \mathbf{0} \models_{pw} \varphi$. Nous omettons parfois l'indice *pw* quand c'est clair dans le contexte.

L'ensemble des mots temporisés finis vérifiant une formule φ est donné par $L^*_{pw}(\varphi) = \{\sigma \in T\Sigma^* \mid \sigma \models_{pw} \varphi\}$. L'ensemble des mots temporisés infinis vérifiant φ est donné par $L^\omega_{pw}(\varphi) = \{\sigma \in T\Sigma^\omega \mid \sigma \models_{pw} \varphi\}$. Quand la sémantique est claire dans le contexte, nous notons simplement $L^*(\varphi)$ et $L^\omega(\varphi)$.

Exemple 5.1.1 *Considérons le mot temporisé* $\sigma = (a; 0)(a; 1,1)(b; 2)\cdots$, *et la formule de* TPTL $\varphi = x.\mathbf{F} (x = 1 \wedge y.\mathbf{F} (y = 1 \wedge b))$. *Cette formule exprime intuitivement qu'une unité de temps après le point* 1, *la proposition atomique* b *est vérifiée.* $\sigma \not\models_{pw} \varphi$ *car il n'y a pas d'action à la date* 1 *dans* σ.

Un mot temporisé peut toujours être vu comme un cas particulier de séquence temporisée. Par exemple, σ *correspond à la séquence temporisée*

$$\kappa = (\{a\}, [0; 0])(\emptyset,]0; 1,1[)(\{a\}, [1,1; 1,1])(\emptyset,]1,1; 2[)(\{b\}, [2; 2])\cdots$$

Alors, $\kappa \models_c \varphi$ *car au point* $t = 1$, κ *vérifie la formule* $y.\mathbf{F} (y = 1 \wedge b)$.

5.1.2 La logique MTL

La logique MTL [Koy90, AH93] étend LTL avec des contraintes quantitatives sur l'opérateur *until*. La syntaxe de MTL est la suivante :

$$\varphi ::= a \mid \varphi \wedge \varphi \mid \neg\varphi \mid \varphi \mathbf{U}_I \varphi$$

où $a \in \mathsf{AP}$ et $I \in \mathcal{I}_\mathbb{Q}$ [2].

[2] Nous rappelons que $\mathcal{I}_\mathbb{Q}$ est l'ensemble des intervalles de $\mathbb{R}_{\geq 0}$ dont les bornes sont rationnelles

Nous définissons la sémantique d'une formule de MTL en la considérant comme une formule de TPTL : $\varphi_1 \mathbf{U}_I \varphi_2$ est interprétée comme $x.(\varphi_1 \mathbf{U} (x \in I \wedge \varphi_2))$.

Nous utilisons les mêmes raccourcis que précédement, comme $\mathbf{F}_I$ ou $\mathbf{G}_I$.

Exemple 5.1.2 *La formule φ de l'exemple 5.1.1 peut être exprimée dans* MTL *par $\mathbf{F}_{=1} \mathbf{F}_{=1} b$. Pour la sémantique continue, cette formule est équivalente à $\mathbf{F}_{=2} b$,* mais ce n'est pas le cas *pour la sémantique d'actions.*

Exemple 5.1.3 *Nous inspirant de [AM04], nous donnons un exemple de formule φ de* MTL *dont la « détemporisation » (c'est-à-dire la projection de $L^*(\varphi)$ sur Σ), notée détemp($L^*(\varphi)$) n'est pas un langage régulier. Soit $\Sigma = \{a, b\}$, la formule $\varphi_1 = \mathbf{G} (a \Rightarrow \mathbf{F}_{=1} b)$ exprime que toute action a est suivie une unité de temps plus tard par une action b. Soit $L \subseteq T\Sigma^*$ composé des mots temporisés dont la détemporisation est dans a^*b^* et tels que deux actions différentes ne se produisent pas à la même date. Il existe une formule φ_2 de* MTL *telle que $L = L^*(\varphi_2)$. Remarquons alors que détemp($L^*(\varphi_1 \wedge \varphi_2)$) $= \{a^n b^m \mid m \geq n\}$, qui n'est pas un langage régulier.*

5.1.3 Les logiques MITL et Safety-MTL

La logique métrique temporelle des intervalles (MITL pour *Metric Interval Temporal Logic*) [AFH96] est une version restreinte de MTL où les intervalles contraignant le *until* ne sont pas des singletons. Empêcher l'utilisation de singletons comme intervalles rend le *model-checking* décidable : pour la sémantique continue le model checking de MITL est EXPSPACE-complet alors qu'il est indécidable pour MTL [AFH96].

Si φ_1 et φ_2 sont des formules de MTL, l'opérateur *until dual* est noté $\varphi_1 \widetilde{\mathbf{U}}_I \varphi_2$. Sa sémantique est celle de la formule $\neg((\neg\varphi_1) \mathbf{U}_I (\neg\varphi_2))$. En utilisant l'opérateur *until dual* et la disjonction, il est possible de réécrire toute formule de MTL en « forme normale positive ». Safety-MTL est le fragment de MTL composé des formules de MTL en forme normale positive qui n'utilisent l'opérateur $\mathbf{U}_I$ que pour des intervalles I bornés. Notons qu'il n'y a pas de restriction sur le *until dual*.

Formellement la syntaxe de Safety-MTL est la suivante :

$$\varphi ::= a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \widetilde{\mathbf{U}}_I \varphi \mid \varphi \mathbf{U}_J \varphi$$

où $a \in \mathsf{AP}$, $I \in \mathcal{I}_\mathbb{Q}$ et $J \in \mathcal{I}_\mathbb{Q}$ est un intervalle borné.

La sémantique de Safety-MTL est obtenue en interprétant une formule de Safety-MTL comme une formule de MTL.

Exemple 5.1.4 *Les formules de* Safety-MTL *peuvent en particulier exprimer l'invariance ; en effet l'opérateur $\mathbf{G}$ peut être utilisé dans* Safety-MTL *car il est équivalent à $\perp \widetilde{\mathbf{U}} \varphi$. Ainsi la formule $\mathbf{G} (a \Rightarrow \mathbf{F}_{\leq 5} b)$ appartient à* Safety-MTL*. Par contre la formule $\mathbf{G} (a \Rightarrow \mathbf{F} b)$ n'appartient pas à* Safety-MTL *car l'opérateur $\mathbf{F}$ n'est pas contraint par un intervalle borné.*

5.1.4 Logiques temporelles avec passé

Les logiques définies plus haut n'utilisent que des modalités relatives au futur. Tout comme pour LTL, nous définissons une version symétrique du *until*, appelée *since* qui est relative aux événements du passé [Kam68, LPZ85]. La sémantique du *since* est la suivante :

– Pour la sémantique continue :

$$\kappa, t, \mathbb{I} \models_c \varphi_1 \, \mathbf{S} \, \varphi_2 \quad \text{si} \quad \exists t' < t \text{ tel que } \kappa, t', \mathbb{I} \models_c \varphi_2$$
$$\text{et } \forall t' < t'' < t,\ \kappa, t'', \mathbb{I} \models_c \varphi_1 \vee \varphi_2$$

– Pour la sémantique d'actions :

$$\rho, i, \mathbb{I} \models_{pw} \varphi_1 \, \mathbf{S} \, \varphi_2 \quad \text{si} \quad \exists j < i \text{ tel que } \rho, j, \mathbb{I} \models_{pw} \varphi_2$$
$$\text{et } \forall j < k < i \ \rho, k, \mathbb{I} \models_{pw} \varphi_1$$

La formule de MTL correspondante est définie de manière naturelle. Par exemple, la formule de TPTL $x.(p \, \mathbf{S} \, (q \wedge x \leq -2))$ exprime que q était vraie il y a 2 unités de temps ou plus tôt, et que p est vraie depuis. Nous la notons $p \, \mathbf{S}_{]-\infty,-2]} \, q$, ou $p \, \mathbf{S}_{\leq -2} \, q$, dans MTL. Nous utilisons les raccourcis classiques $\mathbf{F}_I^{-1} \, \varphi$ pour $\top \, \mathbf{S}_I \, \varphi$, et $\mathbf{G}_I^{-1} \, \varphi$ pour $\neg \mathbf{F}_I^{-1} \, \neg\varphi$.

Nous notons MTL+Past (resp. MITL+Past, TPTL+Past) la logique MTL (resp. MITL, TPTL) étendue avec la modalité *since*. Ces extensions ont été définies et étudiées dans [AH92a, AH93].

Exemple 5.1.5 *La formule $\mathbf{G} \, (a \Rightarrow \mathbf{F}_{=-1}^{-1} \, b)$ de* MTL+Past *exprime pour la sémantique continue que toute action a est précédée une unité de temps plus tôt par une action b. Pour la sémantique d'actions, cette même formule exprime que toute action a,* si elle est précédée une unité de temps plus tôt par une action*, est alors précédée une unité de temps plus tôt par une action b.*

5.1.5 Expressivité relative

Soit $\mathcal{M}$ un ensemble de modèles, et soient $\mathcal{L}$ et $\mathcal{L}'$ des langages d'une logique interprétée sur $\mathcal{M}$. Nous disons qu'une formule $\varphi \in \mathcal{L}$ est *équivalente* à $\varphi' \in \mathcal{L}'$ sur $\mathcal{M}$ si pour tout $\pi \in \mathcal{M}$, π satisfait φ si, et seulement si π satisfait φ'. Le langage $\mathcal{L}'$ est dit *au moins plus expressif* que $\mathcal{L}$ sur $\mathcal{M}$ si, et seulement si toute formule de $\mathcal{L}$ a une formule équivalente de $\mathcal{L}'$ sur $\mathcal{M}$. Il est dit *strictement plus expressif* sur $\mathcal{M}$ si de plus il existe une formule de $\mathcal{L}'$ qui n'a pas d'équivalent dans $\mathcal{L}$ sur $\mathcal{M}$. Deux langages $\mathcal{L}$ et $\mathcal{L}'$ sont *expressivement équivalents* sur $\mathcal{M}$ (nous disons qu'ils *ont la même expressivité*) si chacun est au moins aussi expressif que l'autre sur $\mathcal{M}$. Quand c'est clair dans le contexte, nous omettons de mentionner l'ensemble des modèles $\mathcal{M}$.

Rappelons quelques résultats d'expressivité classiques des logiques temporelles (non temporisées) linéaires :
- tout d'abord il est évident que la logique faite des opérateurs booléens et du *until* strict est au moins aussi expressive que celle avec le *until* non-strict. Elles n'ont pas la même expressivité en général : par exemple le *until* strict peut tester deux occurrences consécutives d'une même lettre, ce qui est impossible au *until* non-strict.
- ajouter des modalités du passé à LTL n'augmente pas son expressivité : toute formule de LTL+Past peut être exprimée dans LTL [Kam68, GPSS80], même s'il existe des cas où la formule de LTL résultante est nécessairement exponentiellement plus grande [LMS02, Mar03b]. Ces résultats ne se généralisent pas au cas temporisé : les modalités du passé augmentent strictement l'expressivité de MITL pour la sémantique continue [AH92a].

Prouver des résultats d'expressivité est parfois complexe. Pour montrer qu'une formule φ ne peut être exprimée dans une logique $\mathcal{L}$, la technique naïve consiste à construire deux modèles M et N tels que φ peut les *distinguer* (ou *séparer*) (*i.e.*, est vraie sur l'un des deux modèles et fausse sur l'autre), et prouver qu'il n'existe pas de formule de $\mathcal{L}$ qui puisse distinguer ces deux modèles.

Cette technique n'est pas suffisante pour montrer par exemple que TPTL est strictement plus expressive que MTL : en effet, soient deux modèles que TPTL peut séparer. Ces modèles sont donc différents ; il existe une proposition atomique a et une date d tels que $\mathbf{F}_{=d}\, a$ est vraie sur un des modèles et fausse sur l'autre. Cette approche naïve ne peut que comparer le *pouvoir de séparation* des logiques, qui est plus grossier que l'expressivité. La remarque ci-dessus montre que TPTL et MTL ont le même pouvoir de séparation. Nous pouvons aussi facilement remarquer que LTL a moins de pouvoir de séparation que TPTL.

Une technique plus fine, que nous utiliserons dans la suite, consiste à construire deux familles de modèles (M_i) et (N_i) telles que pour tout i, φ sépare M_i et N_i, et telle qu'il n'existe pas de formule de $\mathcal{L}$ de taille inférieure à i séparant M_i et N_i. Cette technique a déjà été employée par exemple dans [EH86, Eme91, Lar95, BCL05].

Parmi d'autres techniques, nous pouvons évoquer la traduction de logiques temporelles à d'autres formalismes comme les automates, la théorie des langages ou les structures algébriques [Kam68, GPSS80, AH92a, TW96, Mar03b].

5.2 Indécidabilité des logiques temporisées

Dans cette section, nous montrons que le problème de satisfaisabilité est indécidable pour la plupart des logiques définies précédemment sur les mots finis et infinis. Plus précisément, pour la sémantique continue la satisfaisabilité de MTL et TPTL (et donc MTL+Past et TPTL+Past) est indécidable ; et pour la sémantique d'actions la satisfaisabilité de TPTL et MTL+Past est indécidable. Nous prouvons en fait tous les cas d'indécidabilité de MTL et TPTL sur les mots finis, car la satisfaisabilité de MTL pour la sémantique d'action est décidable [OW05]. Sur les mots infinis, la satisfaisabilité de MTL est indécidable [OW06a], mais notre approche ne permet pas de retrouver ce résultat.

Historiquement TPTL a été montrée indécidable dans [AH89] en utilisant une réduction de l'arrêt des machines à deux compteurs. Nous présentons ici une preuve différente que nous espérons plus simple à partir d'une réduction de l'accessibilité des automates communicants. Cette technique a été utilisée dans [OW05] pour établir la complexité de MTL pour la sémantique d'actions.

Pour une logique $\mathcal{L}$, le problème de satisfaisabilité est défini de la manière suivante :

Problème 6 (Problème de satisfaisabilité) *Soit $\mathcal{L}$ une logique interprétée sur un ensemble de modèles $\mathcal{M}$. Le problème de satisfaisabilité sur $\mathcal{L}$ est, étant donné une formule φ de $\mathcal{L}$, de déterminer s'il existe un modèle $M \in \mathcal{M}$ tel que $M \models \varphi$.*

5.2.1 Automates communicants

Un *automate communicant* est un n-uplet $\mathcal{S} = (M, S, s_0, \Delta, s_{stop})$ où M est un ensemble fini de messages, S un ensemble fini d'états, $s_0 \in S$ l'état initial, $\Delta \subseteq S \times \{m!, m? \mid m \in M\} \times S$ la relation de transition, $s_{stop} \in S$ l'état d'arrêt.

Un automate communicant $\mathcal{S} = (M, S, s_0, \Delta, s_{stop})$ est dit *déterministe* s'il satisfait les deux conditions suivantes : (1) si $(s, a, s_1) \in \Delta$ et $(s, a, s_2) \in \Delta$ alors $s_1 = s_2$; (2) si $(s, m!, s_1) \in \Delta$ et $(s, a, s_2) \in \Delta$ alors $a = m!$ et $s_1 = s_2$.

La sémantique d'un automate communicant déterministe est donnée par un système de transitions étiqueté $G(\mathcal{S})$, dont les sommets sont des couples (s, x) avec $s \in S$ et $x \in M^*$ (représentant le contenu du canal), et dont la relation de transition est définie par : $(s, x) \xrightarrow{a} (s', y)$ si, et seulement si $(s, a, s') \in \Delta$ et, soit $a = m!$ et $y = m \cdot x$, soit $a = m?$ et $x = y \cdot m$. L'état s_{stop} est dit *accessible* dans $\mathcal{S}$ s'il existe $x \in M^*$ et un chemin de $G(\mathcal{S})$ de (s_0, ε) à (s_{stop}, x).

Le problème d'accessibilité pour les automates communicants déterministes est, étant donné un automate communicant déterministe $\mathcal{S} = (M, S, s_0, \Delta, s_{stop})$, de déterminer si s_{stop} est accessible dans $\mathcal{S}$. Le résultat principal sur les automates communicants que nous utiliserons est le suivant :

Proposition 5.2.1 ([BZ83]) *Le problème d'accessibilité pour les automates communicants déterministes est indécidable.*

5.2.2 Codage d'une exécution par un mot temporisé

Soit $\mathcal{S} = (M, S, s_0, \Delta, s_{stop})$ un automate communicant déterministe et soit $M_c = \{m?, m! \mid m \in M\}$ l'ensemble des messages du canal. Nous encodons les exécutions de $\mathcal{S}$ par des mots temporisés sur $\Sigma = M_c \cup S$. Un mot temporisé est correct s'il alterne entre états de $\mathcal{S}$ et messages du canal selon la fonction de transition de $\mathcal{S}$ et si tout message envoyé est reçu une unité de temps plus tard (et réciproquement).

Formellement un mot temporisé $\sigma = (a_0, t_0)(a_1, t_1) \cdots (a_n, t_n)$ sur Σ appartient à $L_{correct}$ s'il vérifie les conditions suivantes :

(R1) $a_0 = s_0$ et pour tout i, $(a_{2i}, a_{2i+1}, a_{2i+2}) \in \Delta$

(R2) il n'y a pas deux actions au même moment : $\forall i, j,\ i \neq j \Rightarrow t_i \neq t_j$,

(R3) à toute action $m!$ correspond une action $m?$ une unité de temps plus tard :
$$\forall i,\ (a_i = m! \text{ et } \exists j\ t_j \geq t_i + 1) \Rightarrow \exists k\ (a_k = m? \text{ et } t_k = t_i + 1),$$

(R4) à toute action $m?$ correspond une action $m!$ une unité de temps plus tôt :
$$\forall i,\ (a_i = m?) \Rightarrow \exists k\ (a_k = m! \text{ et } t_k = t_i - 1).$$

Remarquons que la condition **(R1)** entraîne que a_i appartient à S pour i pair et à M_c pour i impair. Les propriétés **(R3)** et **(R4)** peuvent être schématisées comme suit :

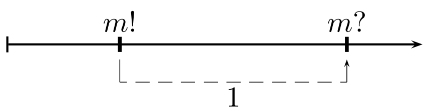


FIG. 5.1 – La propriété **(R3)**

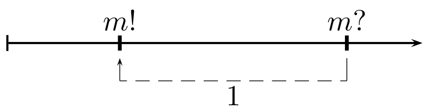


FIG. 5.2 – La propriété **(R4)**

$L_{correct}$ est l'ensemble des codages d'exécutions de $\mathcal{S}$. Nous définissons $L_{stop} = L_{correct} \cap \{w \in T\Sigma^* \mid \exists i\ a_i = s_{stop}\}$ l'ensemble des codages d'exécutions de $\mathcal{S}$ atteignant s_{stop}. Nous avons que s_{stop} est accessible dans $\mathcal{S}$ si, et seulement si L_{stop} est non vide. Nous montrerons l'indécidabilité de certaines logiques temporelles en montrant qu'elles peuvent définir L_{stop}.

5.2.3 Résultats d'indécidabilité

Nous montrons que certaines logiques temporelles définies dans ce chapitre ont un problème de satisfaisabilité indécidable. Une remarque importante est que pour les logiques considérées dans ce chapitre la sémantique continue est *plus expressive* que la sémantique d'actions. Précisons cette affirmation : la syntaxe d'une logique donne en fait lieu à deux logiques distinctes selon qu'elle est interprétée sur les séquences temporisées (sémantique continue) ou sur les mots temporisés (sémantique d'actions). Pour pouvoir comparer l'expressivité de ces deux logiques il faut les considérer sur un même ensemble de modèles (voir la section 5.1.5). Comme tout mot temporisé peut être vu comme une séquence temporisée (voir l'exemple 5.1.1, page 79), nous pouvons comparer l'expressivité d'une même logique selon les deux sémantiques en l'interprétant sur les mots temporisés. Soit $\mathcal{L}$ une logique appartenant à MTL, MTL+Past, TPTL, TPTL+Past, MITL, MITL+Past, nous notons $\mathcal{L}_c$ pour la logique $\mathcal{L}$ interprétée sur les mots temporisés pour la sémantique continue et $\mathcal{L}_{pw}$ pour la logique $\mathcal{L}$ interprétée sur les mots temporisés pour la sémantique d'actions. Nous avons alors le résultat suivant :

Proposition 5.2.2 *Soit $\mathcal{L}$ une logique sur les mots finis ou infinis appartenant à l'ensemble* $\{\mathsf{TPTL}, \mathsf{TPTL{+}Past}, \mathsf{MTL}, \mathsf{MTL{+}Past}, \mathsf{MITL}, \mathsf{MITL{+}Past}\}$*. Alors $\mathcal{L}_c$ est au moins aussi expressive que $\mathcal{L}_{pw}$.*

Preuve. Soit φ une formule de $\mathcal{L}$ *interprétée pour la sémantique d'actions*. Nous construisons une formule φ' de $\mathcal{L}$ qui, *interprétée pour la sémantique continue*, reconnaît les mêmes modèles que φ.

Soit Σ l'ensemble des variables propositionnelles apparaissant dans φ et φ_{action} la formule $\bigvee_{a\in\Sigma} a$, la traduction *trad* est définie inductivement de la façon suivante :

$$
\begin{array}{lcl}
trad(a) & = & a \text{ si } a \in \Sigma \\
trad(\neg\varphi) & = & \neg trad(\varphi) \\
trad(\varphi_1 \vee \varphi_2) & = & trad(\varphi_1) \vee trad(\varphi_2) \\
trad(\varphi_1 \mathbf{U} \varphi_2) & = & (\varphi_{\text{action}} \Rightarrow trad(\varphi_1)) \mathbf{U} (\varphi_{\text{action}} \wedge trad(\varphi_2))
\end{array}
$$

Pour les mots temporisés qui ont une action à l'instant initial, la formule φ' est $trad(\varphi)$. Pour un mot temporisé quelconque, il faut considérer la formule $\varphi' = (\varphi_{\text{action}} \Rightarrow trad(\varphi)) \wedge ((\neg\varphi_{\text{action}}) \Rightarrow (\neg\varphi_{\text{action}}) \mathbf{U} (\varphi_{\text{action}} \wedge trad(\varphi)))$ car pour la sémantique d'actions le modèle est interprété sur la première action alors que pour la sémantique continue il est toujours interprété en 0. $\square$

Nous montrons l'indécidabilité des logiques de l'ensemble suivant $E = \{\mathsf{TPTL}_c, \mathsf{TPTL}_{pw}, \mathsf{TPTL{+}Past}_c, \mathsf{TPTL{+}Past}_{pw}, \mathsf{MTL}_c, \mathsf{MTL{+}Past}_c, \mathsf{MTL{+}Past}_{pw}\}$. Nous considérons tout d'abord ces logiques sur les mots finis, nous expliquerons plus tard comment adapter la preuve aux mots infinis. Remarquons que si une logique $\mathcal{L}_1$ est au moins aussi expressive qu'une logique

$\mathcal{L}_2$ (et ce de façon effective, c'est-à-dire qu'il existe un algorithme qui donne pour toute formule de $\mathcal{L}_2$, une formule équivalente de $\mathcal{L}_1$), et si le problème de satisfaisabilité de $\mathcal{L}_2$ est indécidable, alors le problème de satisfaisabilité de $\mathcal{L}_1$ est aussi indécidable.

Par la proposition 5.2.2 il nous suffit donc de montrer que le problème de satisfaisabilité des trois logiques de l'ensemble E' suivant est indécidable : $E' = \{\mathsf{TPTL}_{pw}, \mathsf{MTL}_c, \mathsf{MTL{+}Past}_{pw}\}$. Notons que ces trois logiques sont toutes au moins aussi expressives que MTL_{pw}.

Soit $\mathcal{S} = (M, S, s_0, \Delta, s_{stop})$ un automate communicant déterministe et soit $M_c = \{m?, m! \mid m \in M\}$ l'ensemble des messages du canal. Soit $\Sigma = M_c \cup S$.

Nous utilisons dans la suite les versions non-strictes de $\mathbf{F}$ et $\mathbf{G}$, elles sont définies par $\overline{\mathbf{F}}_I = \varphi \vee \mathbf{F}_I\, \varphi$ et $\overline{\mathbf{G}}_I = \neg \overline{\mathbf{F}}_I \neg\varphi$.

Soit φ_{action} la formule $\bigvee_{a\in\Sigma} a$, les propriétés **(R1)**, **(R2)** et **(R3)** peuvent respectivement être exprimées dans MTL_{pw} par les formules suivantes :

$$\begin{array}{lcl}
\varphi_{\text{struct}} & = & s_0 \wedge \bigwedge\limits_{s\in Q\setminus\{s_{stop}\}} \mathbf{G}\left(s \Rightarrow \bigvee\limits_{(s,a,s')\in\Delta} ((\mathbf{X}\, a) \wedge (\mathbf{X}\ \mathbf{X}\, s'))\right) \\
\varphi_{\text{sim}} & = & \neg\overline{\mathbf{F}}\,(\varphi_{\text{action}} \wedge \mathbf{F}_{=0}\, \varphi_{\text{action}}) \\
\varphi_{\text{avant}} & = & \bigwedge\limits_{m\in M} \overline{\mathbf{G}}\,((m! \wedge \mathbf{F}_{\geq 1}\, \varphi_{\text{action}}) \Rightarrow \mathbf{F}_{=1}\, m?)
\end{array}$$

Ces trois formules sont donc exprimables dans toutes les logiques de E'. Il est également facile d'exprimer que s_{stop} est atteint par la formule $\mathbf{F}\, s_{stop}$.

Il nous reste donc à montrer que la condition **(R4)** peut être exprimée dans les logiques TPTL_{pw}, MTL_c et $\mathsf{MTL{+}Past}_{pw}$. Nous devons ici distinguer les cas.

- Dans $\mathsf{MTL{+}Past}_{pw}$ comme les modalités du passé sont autorisées, **(R4)** s'exprime naturellement par la formule : $\varphi_{\text{arrière}} = \overline{\mathbf{G}}\,(m? \Rightarrow \mathbf{F}^{-1}_{=-1}\, m!)$.
- Dans MTL_c, nous tirons parti du fait que la sémantique est continue pour exprimer **(R4)** par le fait que quand $\mathbf{F}_{=1}\, m?$ est vraie $m!$ doit être vrai. Ce procédé permet de « regarder dans le passé » indirectement en utilisant la sémantique continue. La propriété **(R4)** peut alors s'exprimer par la formule suivante :

$$\varphi'_{\text{arrière}} = \bigwedge_{m\in M} \overline{\mathbf{G}}_{<1}\,(\neg m?) \wedge \bigwedge_{m\in M} \overline{\mathbf{G}}\,((\mathbf{F}_{=1}\, m?) \Rightarrow m!)$$

- Pour TPTL_{pw} nous ne pouvons utiliser le procédé précédent directement. En effet dans ce cas $\varphi'_{\text{arrière}}$ n'est interprétée qu'aux points où une action se produit. Pour que cette formule soit correcte il faut pouvoir s'assurer que dès qu'il y a une action $m?$ au temps $t+1$, il y a une action au temps t. Grâce à deux horloges il est possible d'imposer cette condition. Nous exprimons la condition **(R4)** dans TPTL_{pw} par la formule suivante :

$$\varphi''_{\text{arrière}} = \varphi'_{\text{arrière}} \wedge \overline{\mathbf{G}}\,(x.\, \mathbf{X}\, y.(\neg\overline{\mathbf{F}}\,(x > 1 \wedge y < 1 \wedge m?)))$$

  Remarquons que **(R4)** n'est pas exprimable dans MTL_{pw} (sinon la satisfaisabilité de cette logique serait indécidable).

Finalement nous avons que s_{stop} est accessible dans $\mathcal{S}$ si, et seulement si $\varphi_{\mathcal{S}} = \varphi_{\text{struct}} \wedge \varphi_{\text{sim}} \wedge \varphi_{\text{avant}} \wedge \varphi_{\text{arrière}} \wedge \mathbf{F}\, s_{stop}$ est satisfaisable (et de même en remplaçant $\varphi_{\text{arrière}}$ par $\varphi'_{\text{arrière}}$ ou $\varphi''_{\text{arrière}}$).

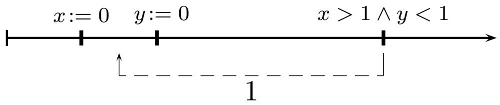


FIG. 5.3 – Exprimer **(R4)** avec deux horloges

La preuve s'adapte facilement aux mots infinis : il suffit d'ajouter une nouvelle action # qui permet de compléter un mot fini en un mot infini[3]. Les formules φ_{struct}, φ_{sim}, φ_{avant} et $\varphi_{\text{arrière}}$ sont inchangées.

Nous avons donc proposé une nouvelle preuve du résultat suivant :

Théorème 5.2.3 *Le problème de satisfaisabilité des logiques suivantes sur les mots finis ou infinis est indécidable :* $\mathsf{TPTL}_c, \mathsf{TPTL}_{pw}$, $\mathsf{TPTL{+}Past}_c$, $\mathsf{TPTL{+}Past}_{pw}$, MTL_c, $\mathsf{MTL{+}Past}_c$, $\mathsf{MTL{+}Past}_{pw}$.

[3]En fait l'ajout de l'action # n'est même pas nécessaire puisqu'il est possible de compléter un mot fini satisfaisant la formule par des actions s_{stop}.

Chapitre 6

Contrôle temporisé pour des spécifications MTL

Le modèle de contrôle pour les systèmes à événements discrets remonte à plusieurs années [RW89, PR89], ces résultats ont été prolongés dans le cadre temporisé au début des années 90 [WTH91]. La forme de contrôle temporisé utilisée aujourd'hui a été introduite dans [MPS95] : il s'agit, étant donné un système modélisé par un automate temporisé $\mathcal{P}$ plongé dans un environnement, de déterminer s'il est possible de contrôler $\mathcal{P}$ pour satisfaire une propriété φ, quoi que fasse l'environnement. Plusieurs types de propriétés ont été étudiées : les propriétés d'accessibilité ou de sûreté [MPS95, AMPS98, HK99], exprimées dans LTL [FLM02a], ou dans TCTL [FLM02b], ou même par des automates temporisés [DM02, BDMP03].

Dans ce chapitre, nous étudions le problème de contrôle temporisé pour des propriétés exprimées dans des logiques temporisées de temps linéaire. Comme le problème de contrôle est plus difficile que celui de la satisfaisabilité, nous nous restreignons à des logiques dont le problème de satisfaisabilité est décidable. Nous considérons la sémantique d'actions et des propriétés exprimées dans MTL sur les mots finis et Safety-MTL sur les mots infinis ; ces logiques ayant été montrées décidables [OW05, OW06b].

Nous utilisons une technique similaire à celle que nous avons utilisée pour montrer l'indécidabilité de certaines logiques temporisées dans le chapitre 5 : nous réduisons l'accessibilité des automates communicants à un problème de contrôle pour montrer que le contrôle temporisé pour des spécifications MTL sur les mots finis et Safety-MTL sur les mots infinis est indécidable. Nous utilisons l'interaction entre l'environnement et le contrôleur pour vérifier que toute action p est précédée une unité de temps plus tôt par une action q. Cette propriété ne peut être exprimée dans MTL pour la sémantique d'actions, mais suffit à mener à l'indécidabilité.

Comme le contrôle pour des spécifications MTL ou Safety-MTL est indécidable, nous suivons [DM02] et restreignons la puissance du contrôleur en fixant son nombre d'horloges et les constantes de ses gardes. Nous obtenons ainsi le problème de contrôle à ressources fixées. Nous ne pouvons pas transformer notre problème en un jeu de parité via l'automate des régions comme dans [DM02], nous utilisons des techniques d'ordre partiel pour les configurations temporisées [AN01, OW04, OW05] et les adaptons au cadre du contrôle pour obtenir la décidabilité du contrôle à ressources fixées pour des spécifications MTL sur les mots finis et Safety-MTL sur les mots infinis.

6.1 Problème de contrôle

Nous définissons tout d'abord quelques notions qui nous seront utiles et présentons ensuite les problèmes de contrôle que nous considérerons dans ce chapitre.

6.1.1 Granularité

Nous considèrerons des sous-classes d'automates temporisés dont les constantes et le nombre d'horloges sont fixés. Nous utilisons pour cela le concept de *granularité*.

Définition 6.1.1 *Une* granularité *est un triplet* $\mu = (X, m, M)$ *où* X *est un ensemble fini d'horloges,* m *un entier strictement positif, et* M *un entier positif.*

Intuitivement M représente la constante maximale utilisable et m représente la finesse des gardes de μ : toutes les constantes utilisées dans une garde de granularité μ sont multiples de $\frac{1}{m}$.

Une garde g est *de granularité* μ= (X, m, M) si elle utilise des horloges de X et toute constante apparaissant dans g est de la forme $\frac{\alpha}{m}$ avec $\alpha \leq M$. Soient μ et μ' deux granularités, μ est dite *plus fine* que μ' si toute garde de granularité μ' est aussi de granularité μ.

Soit G un ensemble fini de gardes, $\mu = (X, m, M)$ est la granularité de G si X est l'ensemble des horloges utilisées dans G, m est le plus petit dénominateur commun des constantes de G, et $\frac{M}{m}$ est la plus grande constante apparaissant dans G.

Une garde g de granularité μ est *atomique pour* μ si pour toute garde g' de granularité μ, $[\![g]\!] \subseteq [\![g']\!]$ ou $[\![g]\!] \cap [\![g']\!] = \emptyset$. Un automate temporisé est *atomique* si toutes ses gardes sont atomiques pour sa granularité.

6.1.2 Systèmes de transitions symboliques

Nous fixons un alphabet symbolique, qui est un ensemble de contraintes d'horloges (voir page 22) et définissons un système de transitions symbolique comme un système de transitions sur cet alphabet. Les systèmes de transitions symboliques sont une généralisation des automates temporisés, car leurs étiquettes de transitions sont fixées, mais ils peuvent avoir un nombre infini d'états.

Définition 6.1.2 *Soit* Γ *un alphabet symbolique, un* système de transitions symbolique *sur* Γ *est un n-uplet* $\mathcal{T} = (\Gamma, S, s_0, \rightarrow)$ *où* S *est un ensemble (éventuellement infini) d'états,* $s_0 \in S$ *un état initial, et* $\rightarrow \subseteq S \times \Gamma \times S$ *la relation de transition.*

Un automate temporisé $\mathcal{A}$ peut être vu comme un système de transitions symbolique à un nombre fini d'états avec des conditions d'acceptation. Nous notons $\mathcal{T}(\mathcal{A})$ le système de transitions symbolique associé à $\mathcal{A}$.

Une exécution de $\mathcal{T}$ sur un mot $w = w_0 w_1 \cdots \in \Gamma^\infty$ est une suite $\rho = s_0 \xrightarrow{w_0} s_1 \xrightarrow{w_1} \cdots$ Nous notons $L^*_{symb}(\mathcal{T})$ (resp. $L^\omega_{symb}(\mathcal{T})$) l'ensemble des mots finis (resp. infinis) de Γ sur lesquels $\mathcal{T}$ a une exécution.

Un système de transitions symbolique $\mathcal{T} = (\Gamma, S, s_0, \rightarrow)$ est *symboliquement déterministe* si $s \xrightarrow{b} s_1$ et $s \xrightarrow{b} s_2$ impliquent $s_1 = s_2$. Pour tout état $s \in S$, nous notons $\mathsf{Enb}_{\mathcal{T}}(s)$ l'ensemble des actions symboliques possibles en s, c'est-à-dire l'ensemble des $b \in \Gamma$ tels qu'il existe $s' \in S$ avec $s \xrightarrow{b} s'$.

Si $\mathcal{T}$ est symboliquement déterministe, pour tout mot $\gamma \in L^*_{symb}(\mathcal{T})$, il y a exactement une exécution de $\mathcal{T}$ sur γ. Nous notons $\textit{état}_\mathcal{T}(\gamma)$ le dernier état de cette exécution. Si $\gamma \notin L^*_{symb}(\mathcal{T})$, $\textit{état}_\mathcal{T}(\gamma)$ n'est pas défini.

Nous donnons maintenant une construction pour déterminiser un système de transitions symbolique. Il s'agit d'une simple généralisation de la méthode classique des sous-ensembles pour les automates finis. Soit $\mathcal{T} = (\Gamma, S, s_0, \rightarrow)$ un système de transitions symbolique. La *déterminisation* de $\mathcal{T}$, notée $Det(\mathcal{T})$, est le système de transitions symbolique déterministe $(\Gamma, 2^S, \{s_0\}, \rightarrow_D)$, où

$$S_1 \xrightarrow{b}_D S_2 \text{ si, et seulement si } S_2 = \{s_2 \in S \mid \exists s_1 \in S_1 \ s_1 \xrightarrow{b} s_2\} \text{ et } S_2 \neq \emptyset.$$

Notons que $L^*_{symb}(Det(\mathcal{T})) = L^*_{symb}(\mathcal{T})$.

Tout comme pour les automates temporisés nous définissons le langage temporisé d'un système de transitions symbolique : $L^*(\mathcal{T})$ est défini par $tw(L^*_{symb}(\mathcal{T}))$ et $L^\omega(\mathcal{T})$ par $tw(L^\omega_{symb}(\mathcal{T}))$.

Un système de transitions symbolique $\mathcal{T}$ est *déterministe* s'il n'y a pas deux transitions distinctes $q \xrightarrow{g_1,a,Y_1} q_1$ et $q \xrightarrow{g_2,a,Y_2} q_2$ avec $[\![g_1]\!] \cap [\![g_2]\!] \neq \emptyset$. Cette notion est plus forte que le déterminisme symbolique. Remarquons qu'un système de transitions symbolique déterministe avec un nombre fini d'états est un automate temporisé déterministe.

Nous définissons maintenant le produit synchronisé de systèmes de transitions symboliques. Il s'agit du produit synchronisé classique où les transitions se synchronisent sur toute lettre de l'alphabet. Soient $\mathcal{T}_1 = (\Gamma, S_1, s_0^1, \rightarrow_1)$ et $\mathcal{T}_2 = (\Gamma, S_2, s_0^2, \rightarrow_2)$ deux systèmes de transitions symboliques sur un alphabet symbolique Γ. Le *produit synchronisé* de $\mathcal{T}_1$ et $\mathcal{T}_2$, noté $\mathcal{T}_1 \parallel \mathcal{T}_2$, est le système de transitions symbolique $(\Gamma, S, s_0, \rightarrow)$ où :

- $S = S_1 \times S_2$
- $s_0 = (s_0^1, s_0^2)$
- $(p_1, p_2) \xrightarrow{g,a,Y} (s_1, s_2)$ si, et seulement s'il existe $p_1 \xrightarrow{g_1,a,Y_1}_1 s_1$ et $p_2 \xrightarrow{g_2,a_2,Y_2}_2 s_2$ avec $g = g_1 \wedge g_2$ et $Y = Y_1 \cup Y_2$

6.1.3 Contrôle pour des spécifications MTL

Soit $\Sigma = \Sigma_C \cup \Sigma_E$ un alphabet partitionné en un ensemble d'actions *contrôlables* Σ_C et un ensemble d'actions *incontrôlables* Σ_E (ce sont les actions appartenant à l'environnement). Un *système* $\mathcal{P}$ sur Σ est un automate temporisé déterministe sur Σ tel que pour tout $\gamma \in L_{symb}(\mathcal{P})$, $tw(\gamma) \neq \emptyset$ [1].

Définition 6.1.3 *Soit $\Sigma = \Sigma_C \cup \Sigma_E$ un alphabet partitionné et $\mathcal{P}$ un système sur Σ. Soit $X_\mathcal{P}$ l'ensemble des horloges de $\mathcal{P}$, $X_\mathcal{C}$ un ensemble fini d'horloges distinct de $X_\mathcal{P}$ et $\mu = (X_\mathcal{P} \cup X_\mathcal{C}, m, M)$ une granularité plus fine que celle de $\mathcal{P}$. Un μ-*contrôleur *pour $\mathcal{P}$ est un système de transitions symbolique déterministe $\mathcal{C}$ sur un alphabet symbolique basé sur $(\Sigma, X_\mathcal{P} \cup X_\mathcal{C})$ de granularité μ satisfaisant les conditions suivantes :*

(C1) *$\mathcal{C}$ ne remet pas à zéro les horloges du système : $q_\mathcal{C} \xrightarrow{g,a,Y} q'_\mathcal{C}$ dans $\mathcal{C}$ implique $Y \subseteq X_\mathcal{C}$.*

(C2) *$\mathcal{C}$ ne restreint pas les actions de l'environnement (condition dite de *non-restriction*) : si $e \in \Sigma_E$, $\sigma \in L^*(\mathcal{T}(\mathcal{P} \parallel \mathcal{C}))$ et $\sigma \cdot (e,t) \in L^*(\mathcal{T}(\mathcal{P}))$, alors $\sigma \cdot (e,t) \in L^*(\mathcal{T}(\mathcal{P} \parallel \mathcal{C}))$.*

[1] Nous expliquons cette condition dans la remarque 6.1.4.

(C3) *$\mathcal{C}$ est* non bloquant *: si $\sigma \in L^*(\mathcal{T}(\mathcal{P}\|\mathcal{C}))$ et $\sigma \cdot (a,t) \in L^*(\mathcal{T}(\mathcal{P}))$, alors il existe $b \in \Sigma$ et $t' \in \mathbb{T}$ tels que $\sigma \cdot (b,t') \in L^*(\mathcal{T}(\mathcal{P}\|\mathcal{C}))$.*

Ces restrictions naturelles du pouvoir du contrôleur ont été initialement énoncées dans [DM02].

Pour un langage temporisé de mots finis $L \subseteq T\Sigma^*$, nous disons qu'un μ-contrôleur $\mathcal{C}$ *contrôle* $\mathcal{P}$ pour la spécification souhaitée (resp. interdite) $\mathcal{L}$ si, et seulement si $L^*(\mathcal{P}\|\mathcal{C}) \subseteq \mathcal{L}$ (resp. $L^*(\mathcal{P}\|\mathcal{C}) \cap \mathcal{L} = \emptyset$). La définition est similaire pour les mots temporisés infinis.

Remarque 6.1.4 *Étant donné un système $\mathcal{P}$ interprété sur les mots finis, il est facile de construire un système équivalent ayant la propriété que pour tout $\gamma \in L_{symb}(\mathcal{P})$, $tw(\gamma) \neq \emptyset$, en utilisant la construction classique de l'automate des régions [AD94].*

Sur les mots infinis, cette construction demande à changer le langage symbolique reconnu et ne s'applique donc pas directement aux problème de contrôle. Nous demandons que tout chemin accepteur de $\mathcal{P}$ reconnaisse des mots temporisés pour empêcher d'avoir des contrôleurs pouvant contrôler correctement le système en étant zénon, c'est-à-dire en faisant converger le temps. En effet, si $tw(\gamma) = \emptyset$ (ce qui est le cas si $tw(\gamma)$ ne contient que des mots zénons), tout contrôleur jouant γ est gagnant car il n'y aura pas de mots temporisés dans $tw(\gamma)$.

Nous présentons maintenant les problèmes de contrôle qui nous intéresseront. Le problème de contrôle à ressources non-fixées consiste à déterminer s'il existe un contrôleur pour le système donné en entrée. Le problème de contrôle à ressources fixées consiste à déterminer s'il existe un contrôleur de ressources fixées pour le système. Ce deuxième problème semble donc à première vue plus simple puisqu'une partie des paramètres du contrôleur sont fixés.

Problème 7 *Le* ***problème de contrôle à ressources non-fixées pour des spécifications souhaitées (resp. interdites)*** *est, étant donné un système $\mathcal{P}$, une spécification $\mathcal{L}$, de déterminer s'il existe une granularité μ plus fine que celle de $\mathcal{P}$ et un μ-contrôleur $\mathcal{C}$ qui contrôle $\mathcal{P}$ pour la spécification désirée (resp. interdite) $\mathcal{L}$.*

Problème 8 *Le* ***problème de contrôle à ressources fixées pour des spécifications souhaitées (resp. interdites)*** *est, étant donné un système $\mathcal{P}$, une spécification $\mathcal{L}$, et une granularité μ plus fine que celle de $\mathcal{P}$, de déterminer s'il existe un μ-contrôleur $\mathcal{C}$ qui contrôle $\mathcal{P}$ pour la spécification désirée (resp. interdite) $\mathcal{L}$.*

Nous nous intéresserons aux problèmes de contrôle où les spécifications sont données par des formules de MTL, c'est-à-dire où $\mathcal{L} = L^*(\varphi)$ ou $\mathcal{L} = L^\omega(\varphi)$ pour $\varphi \in$ MTL. Dans [DM02], les problèmes de contrôle à ressources fixées ou non-fixées sont résolus pour des spécifications données par des automates de Büchi temporisés.

Le problème de satisfaisabilité d'une logique peut être réduit trivialement au problème de contrôle sur cette logique. Pour la sémantique continue, les problèmes de contrôle MTL et Safety-MTL, sur les mots finis ou infinis, pour des spécifications souhaitées ou interdites, sont donc clairement indécidables. Dans tout ce chapitre, nous interprétons donc les logiques MTL et Safety-MTL pour la sémantique d'actions.

Le tableau suivant récapitule les résultats que nous montrerons dans la suite de ce chapitre (le cas de Safety-MTL à ressources non-fixées et pour des spécifications interdites est à ce jour ouvert) :

	ressources fixées	**ressources non-fixées**
MTL sur les mots finis *spécifications souhaitées ou interdites*	décidable	indécidable
Safety-MTL sur les mots infinis *spécifications souhaitées*	décidable	indécidable
Safety-MTL sur les mots infinis *spécifications interdites*	décidable	?

6.2 Contrôle à ressources non-fixées

Dans cette section, nous montrons qu'à ressources non-fixées, les problèmes de contrôle MTL sur les mots finis et Safety-MTL sur les mots infinis pour des spécifications souhaitées sont indécidables.

6.2.1 Retour sur les automates communicants

Nous montrerons l'indécidabilité des problèmes de contrôle précédents par réduction de l'accessibilité des automates communicants. Dans la section 5.2 nous avons réduit ce problème à la satisfaisabilité de certaines logiques temporisées pour montrer leur indécidabilité. Ici, nous avons besoin de raffiner un peu le modèle des automates communicants, nous allons imposer des conditions supplémentaires au modèle, qui n'altèrent cependant pas l'indécidabilité de l'accessibilité.

Soit $\mathcal{S} = (M, S, s_0, \Delta, s_{stop})$ un automate communicant déterministe. Un *cycle d'écriture* est une suite $s_1, \cdots, s_k$ avec $s_k = s_1$ et $(s_i, m_i!, s_{i+1}) \in \Delta$ pour tout $1 \leq i < k$. Rappelons que dans un automate communicant déterministe tel que défini dans le chapitre 5, il y a au plus une exécution possible à partir d'une configuration donnée.

$\mathcal{S}$ a *un canal divergent* si la propriété suivante est vérifiée : si l'unique chemin maximal de $G(\mathcal{S})$ partant de (s_0, ε) est infini, alors la taille du canal est non-bornée sur cette exécution.

Lemme 6.2.1 *Étant donné un automate communicant déterministe $\mathcal{S}' = (M', S', s'_0, \Delta', s'_{stop})$, nous pouvons construire un automate communicant déterministe $\mathcal{S} = (M, S, s_0, \Delta, s_{stop})$ tel que s_{stop} est le seul état sans transition sortante, ne contenant pas de cycle d'écriture, ayant un canal divergent et tel que s'_{stop} est accessible dans $\mathcal{S}'$ si, et seulement si s_{stop} est accessible dans $\mathcal{S}$.*

Preuve. Nous construisons $\mathcal{S}$ en plusieurs étapes :
- nous supprimons tout d'abord toutes les transitions sortantes de s'_{stop}, puis nous supprimons toutes les transitions faisant partie d'un cycle d'écriture. Comme l'automate communicant obtenu est déterministe et qu'il n'y a pas de transition sortante de s'_{stop}, un cycle d'écriture amène à boucler sans atteindre s'_{stop}. Supprimer un tel cycle ne change donc pas l'accessibilité de s'_{stop}.
- nous supprimons alors tous les états différents de s'_{stop} qui n'ont pas de transition sortante et répétons cette opération jusqu'à ce qu'il ne reste plus de tel état. Soit $\mathcal{S}_1 = (M', S', s'_0, \Delta_1, s'_{stop})$ l'automate communicant déterministe résultant.
- nous transformons alors $\mathcal{S}_1$ en un automate communicant déterministe $\mathcal{S}$ qui a un canal divergent. Pour cela, nous ajoutons un nouveau message # à M. L'idée de la construction

est que $\mathcal{S}$ va simuler $\mathcal{S}_1$, et après toute étape de $\mathcal{S}_1$, un message # sera ajouté dans le canal de $\mathcal{S}$. Dans tout état de $\mathcal{S}$, un message # peut être lu et réécrit immédiatement (ce qui permet de « cycler » les messages #).

$\mathcal{S}$ est formellement défini comme suit :
- $M = M' \cup \{\#\}$, $s_0 = s'_0$, $s_{stop} = s'_{stop}$
- $S = S' \cup \overline{S'} \cup S'_{\Delta_1}$ où $\overline{S'} = \{\overline{s'} \mid s' \in S'\}$ et $S'_{\Delta_1} = \{s'_\delta \mid \delta \in \Delta_1\}$
- Δ est défini comme suit :
  - pour tout $s' \in S' \backslash \{s'_{stop}\}$ tel qu'il n'existe pas de transition étiquetée par une action d'écriture sortant de s', $(s', \#?, \overline{s'}) \in \Delta$ et $(\overline{s'}, \#!, s') \in \Delta$
  - si $\delta = (s'_1, a, s'_2) \in \Delta_1$, alors $(s'_1, a, s'_\delta) \in \Delta'$ et $(s'_\delta, \#!, s'_1) \in \Delta'$

$\mathcal{S}$ a un canal divergent parce qu'après avoir simulé n étapes de $\mathcal{S}_1$, le canal contient au moins n messages # (en plus des messages se trouvant normalement dans le canal de $\mathcal{S}_1$). Remarquons que s_{stop} est toujours le seul état sans transition sortante, et que nous n'avons pas ajouté de cycle d'écriture en construisant $\mathcal{S}$.

□

6.2.2 Contrôle MTL sur les mots finis

Nous traitons tout d'abord le cas du contrôle MTL sur les mots finis. Nous utilisons le codage de la section 5.2 pour coder les exécutions d'un automate communicant et montrons que nous pouvons exprimer l'accessibilité de l'état final par un problème de contrôle.

Nous fixons un automate communicant déterministe $\mathcal{S} = (M, S, s_0, \Delta, s_{stop})$ tel que s_{stop} est le seul état sans transition sortante, ne contenant pas de cycle d'écriture, et ayant un canal divergent.

Nous considérons un codage des configurations d'un automate communicant déterministe similaire à celui exposé dans la section 5.2. Soit $\Sigma = \{m?, m! \mid m \in M\}$ l'ensemble des messages du canal. Un mot temporisé $\sigma = (a_0, t_0)(a_1, t_1) \cdots (a_n, t_n)$ sur Σ appartient à $L_{correct}$ s'il vérifie les conditions suivantes :

(R1) il existe $s_1, s_2, \cdots \in S$ tels que $s_1 = s_0$ et pour tout $i \geq 1$ $(s_i, a_i, s_{i+1}) \in \Delta$,

(R2) il n'y a pas deux actions au même moment : $\forall i, j,\ i \neq j \Rightarrow t_i \neq t_j$,

(R3) à toute action $m!$ correspond une action $m?$ une unité de temps plus tard :
$$\forall i,\ (a_i = m! \wedge \exists j\ t_j \geq t_i + 1) \Rightarrow \exists k\ (a_k = m? \text{ et } t_k = t_i + 1),$$

(R4) à toute action $m?$ correspond une action $m!$ une unité de temps plus tôt :
$$\forall i,\ (a_i = m?) \Rightarrow \exists k\ (a_k = m! \text{ et } t_k = t_i - 1),$$

Remarque 6.2.2 *Le codage des exécutions que nous proposons est un peu différent que celui du chapitre 5 puisqu'il est composé de messages uniquement et non plus de messages et d'états discrets. En effet, nous ferons jouer le contrôleur sur un automate ayant la même structure que $\mathcal{S}$, les états discrets seront donc obtenus par le chemin suivi dans l'automate. Nous utilisons ce nouveau codage dans le but de traiter le contrôle* Safety-MTL *; en effet la formule φ_{struct} ne peut être écrite dans* Safety-MTL. *Il serait néanmoins possible de conserver le codage du chapitre 5 en imposant au contrôleur une certaine réactivité et en réécrivant certaines formules.*

Nous rappelons que les propriétés **(R2)** et **(R3)** peuvent respectivement être exprimées dans MTL par les formules suivantes :

$$\begin{aligned}\varphi_{\text{sim}} &= \neg\overline{\mathbf{F}}\,(\varphi_{\text{action}} \wedge \mathbf{F}_{=0}\,\varphi_{\text{action}})\\ \varphi_{\text{avant}} &= \bigwedge_{m\in M} \overline{\mathbf{G}}\,((m! \wedge \mathbf{F}_{\geq 1}\,\varphi_{\text{action}}) \Rightarrow \mathbf{F}_{=1}\,m?)\end{aligned}$$

La propriété **(R4)** ne peut pas être exprimée dans MTL sur les mots finis (pour la sémantique d'actions que nous considérons ici), sinon cette logique serait indécidable. Pour traduire cette condition nous aurons à utiliser l'interaction avec l'environnement.

Nous construisons maintenant un problème de contrôle $(\mathcal{P}_\mathcal{S}, L^*(\varphi_\mathcal{S}))$ (où $\mathcal{P}_\mathcal{S}$ est un système et $\varphi_\mathcal{S}$ une formule de MTL) tel que s_{stop} est accessible dans $\mathcal{S}$ si, et seulement s'il existe une granularité μ plus fine que celle de $\mathcal{P}$ et un μ-contrôleur $\mathcal{C}$ qui contrôle $\mathcal{P}_\mathcal{S}$ pour la spécification désirée $L^*(\varphi_\mathcal{S})$.

L'idée de la réduction est la suivante : le système est intuitivement l'automate communicant $\mathcal{S}$ où toutes les actions $m!$ et $m?$ sont contrôlables. Il permet au contrôleur de jouer une exécution de $\mathcal{S}$ selon le codage décrit précédemment. Nous ajoutons également deux actions incontrôlables *Nil* et *Test*. Après chaque action du contrôleur, l'environnement répond par une action incontrôlable (en temps nul). Quand c'est son tour, l'environnement peut soit jouer l'action *Nil* pour continuer la simulation, soit jouer l'action *Test*, ce qui met fin au jeu (la signification de l'action *Test* est expliquée plus bas). Le but du contrôleur est de jouer une exécution correcte de $\mathcal{S}$ atteignant s_{stop} (bien sûr cela n'est possible que si s_{stop} est réellement accessible dans $\mathcal{S}$). Si s_{stop} est atteint, le contrôleur est déclaré gagnant (si l'exécution jouée jusque là est correcte).

Le système $\mathcal{P}_\mathcal{S}$ est formellement défini par $\mathcal{P}_\mathcal{S} = (\Sigma_C \cup \Sigma_E, X, Q, q_0, F, \rightarrow)$ où :
- $\Sigma_C = \{m!, m? \mid m \in M\}$, $\Sigma_E = \{\mathit{Nil}, \mathit{Test}\}$, et $X = \{x\}$,
- $Q = S \cup \{q_\delta \mid \delta \in \Delta\} \cup \{q_{Fin}\}$, $q_0 = s_0$, et $F = Q$,
- $q \xrightarrow{\top,\ a,\ \{x\}} q_\delta$ pour tout $\delta = (q, a, q') \in \Delta$,
- $q_\delta \xrightarrow{x=0,\ \mathit{Nil},\ \{x\}} q'$ pour tout $\delta = (q, a, q') \in \Delta$,
- $q_\delta \xrightarrow{x=0,\ \mathit{Test},\ \{x\}} q_{Fin}$ pour tout $\delta \in \Delta$.

Remarque 6.2.3 *Nous avons utilisé un système avec une horloge, qui sert uniquement à forcer l'environnement à répondre au contrôleur immédiatement. Cette horloge n'est pas nécessaire car nous pourrions assurer cette condition dans la spécification.*

Nous expliquons maintenant comment utiliser la spécification et l'action *Test* pour assurer que le contrôleur joue un mot de $L_{correct}$ (ou perde dans le cas contraire) :
- la propriété **(R1)** est satisfaite car $\mathcal{P}_\mathcal{S}$ a la même structure que $\mathcal{S}$.
- les propriétés **(R2)** et **(R3)** sont encodées dans la spécification par les formules φ_{sim} et φ_{avant} décrites dans le chapitre 5 et rappelées ci-avant.
- la propriété **(R4)** est vérifiée par l'environnement. Après chaque action du contrôleur, l'environnement peut décider de jouer l'action *Test* et d'arrêter le jeu. Dans ce cas, si l'action *Test* est jouée juste après (mais en temps 0) une action $m?$ et qu'il n'y a pas d'action $m!$ correspondant une unité de temps plus tôt, le contrôleur est déclaré perdant (dans la spécification). Si l'environnement a joué l'action *Test* à tort (c'est-à-dire s'il n'y

a pas d'action $m?$ ou s'il y a une action $m!$ une unité de temps plus tôt), le contrôleur est déclaré gagnant.
Le contrôleur est donc forcé de simuler une exécution correcte de $\mathcal{S}$ parce que s'il essaie d'insérer une action $m?$ à laquelle ne correspond aucune action $m!$ une unité de temps plus tôt, il risque de perdre si l'environnement joue l'action *Test* à ce moment.

Nous donnons maintenant la définition formelle de la spécification. Nous devons prendre garde au détail technique suivant : les formules φ_{sim} et φ_{avant} décrivent les propriétés **(R2)** et **(R3)** pour un mot de Σ. Or une exécution du système produit un mot sur $\Sigma \cup \{\mathit{Nil}, \mathit{Test}\}$, nous devons donc « ignorer » les actions *Nil* et *Test* en utilisant une technique classique.

Soit $\varphi_{\text{action}} = \bigvee_{c \in \Sigma} c$; nous considérons les formules suivantes :

$$\begin{aligned}
\varphi_{\text{sim}} &= \neg \overline{\mathbf{F}}\,(\varphi_{\text{action}} \wedge \mathbf{F}_{=0}\,\varphi_{\text{action}}) \\
\varphi_{\text{avant}} &= \bigwedge_{m \in M} \overline{\mathbf{G}}\,((m! \wedge \mathbf{F}_{\geq 1}\,\varphi_{\text{action}}) \Rightarrow \mathbf{F}_{=1}\,m?) \\
\varphi_{\text{test}} &= \textstyle\bigwedge_{m \in M} \Big(\big(\overline{\mathbf{F}}\,(m? \wedge \mathbf{F}_{=0}\,\mathit{Test})\big) \Rightarrow \overline{\mathbf{F}}\,(m! \wedge \mathbf{F}_{=1}\,\mathit{Test}) \Big)
\end{aligned}$$

La spécification $\varphi_{\mathcal{S}}$ est alors donnée par $\varphi = \varphi_{\text{sim}} \wedge \varphi_{\text{avant}} \wedge \varphi_{\text{test}}$. La formule φ_{test} assure que si l'action *Test* est jouée au même moment qu'une action $m?$ alors il doit exister une action $m!$ une unité de temps plus tôt. Nous montrons maintenant la correction de cette réduction :

Proposition 6.2.4 *s_{stop} est accessible dans $\mathcal{S}$ si, et seulement s'il existe un contrôleur de $\mathcal{P}_{\mathcal{S}}$ pour la spécification souhaitée $\varphi_{\mathcal{S}}$.*

Preuve.
Implication directe. Supposons s_{stop} accessible dans $\mathcal{S}$. Alors il existe une exécution correcte finie atteignant s_{stop} dont toutes les dates appartiennent à $\mathbb{Q}_{\geq 0}$. Le contrôleur $\mathcal{C}$ va simplement jouer ce mot temporisé ; un tel contrôleur peut être réalisé avec une unique horloge qui est remise à zéro après chaque transition. Si l'environnement répond *Nil* jusqu'à la fin de l'exécution la spécification est satisfaite. Si l'environnement joue l'action *Test* à un moment, φ_{test} sera vérifiée car à toute action $m?$ correspond une action $m!$ une unité de temps plus tôt.

Implication réciproque. Nous expliquons tout d'abord intuitivement pourquoi si s_{stop} n'est pas accessible dans $\mathcal{S}$, il ne peut exister de contrôleur de $\mathcal{P}_{\mathcal{S}}$ pour la spécification souhaitée $\varphi_{\mathcal{S}}$.

La spécification assure que $\mathcal{C}$ doit jouer une exécution correcte de $\mathcal{S}$. Le fait que s_{stop} ne soit pas accessible dans $\mathcal{S}$ peut provenir de deux situations :

- $\mathcal{S}$ peut être bloqué dans un état différent de s_{stop}. Dans ce cas, comme $\mathcal{C}$ est non-bloquant il va tout de même devoir jouer une action, violant ainsi la spécification.
- il y a une exécution infinie de $\mathcal{S}$ qui n'atteint jamais s_{stop}. Dans ce cas, comme $\mathcal{S}$ a un canal divergent, le canal sera non-borné sur cette exécution et un contrôleur ne pourra pas simuler cette exécution car il aurait besoin pour cela d'une infinité d'horloges (une par symbole sur le canal).

Nous montrons maintenant formellement qu'il ne peut exister de contrôleur de $\mathcal{P}_{\mathcal{S}}$ pour la spécification souhaitée $\varphi_{\mathcal{S}}$. Supposons par l'absurde qu'il existe un tel contrôleur $\mathcal{C}$. Considérons un mot maximal w de $L^{\infty}(\mathcal{P}_{\mathcal{S}} \parallel \mathcal{C})$ ne contenant pas l'action *Test* (ce mot correspond à ce que fait le contrôleur si l'environnement ne joue jamais l'action *Test*).

Soit π la projection sur $\{m!, m? \mid m \in M\}$, nous montrons le lemme suivant :

Lemme 6.2.5 *$\pi(w)$ est infini et appartient à $L_{correct}$.*

Preuve.[Preuve du lemme] $\pi(w)$ satisfait la propriété **(R1)** car $\mathcal{P}_\mathcal{S}$ a la même structure que $\mathcal{S}$. $\pi(w)$ satisfait les propriétés **(R2)** et **(R3)** car $\mathcal{C}$ doit satisfaire φ_{sim} et φ_{avant}.

$\pi(w)$ satisfait la propriété **(R4)**. Si ce n'était pas le cas, il existerait un préfixe $w' = (b_0, t_0) \cdots (b_{n-1}, t_{n-1})(m?, t_n)$ de $\pi(w)$ tel que pour tout i, $(b_i, t_i) \neq (m!, t_n - 1)$. Alors $w'' = (b_0, t_0)(\mathit{Nil}, t_0)(b_1, t_1)(\mathit{Nil}, t_1) \cdots (b_{n-1}, t_{n-1})(\mathit{Nil}, t_{n-1})(m?, t_n)(\mathit{Test}, t_n)$ serait dans $L^*(\mathcal{P}_\mathcal{S} \parallel \mathcal{C})$. Comme $w'' \not\models \varphi_{\mathit{Test}}$, c'est impossible. $\pi(w)$ appartient donc à $L_{correct}$.

Nous pouvons maintenant montrer que $\pi(w)$ est infini. Supposons par l'absurde que $\pi(w)$ soit fini, alors par ce qui précède, $\pi(w) \in L_{correct}$. Alors comme $\pi(w)$ est maximal, s_{stop} est le seul état sans transition sortante, et $\mathcal{C}$ est non bloquant, $\pi(w)$ mène nécessairement à s_{stop}. L'état s_{stop} est donc accessible dans $\mathcal{S}$ ce qui est absurde. $\pi(w)$ est donc infini. □

Revenons à la preuve de la proposition 6.2.4. Comme s_{stop} n'est pas accessible dans $\mathcal{S}$ et que $\mathcal{S}$ a un canal divergent, le canal de $\mathcal{S}$ est non borné sur l'unique exécution maximale de $\mathcal{S}$. En particulier, à un moment il y aura trop de lettres dans le canal pour que $\mathcal{C}$ puisse se souvenir de toutes. Nous considérerons le préfixe correspondant de w et montrons en utilisant ce préfixe que $\mathcal{C}$ ne peut être un contrôleur pour $\varphi_\mathcal{S}$. En fait il est possible que ce préfixe satisfasse la spécification, mais nous montrerons qu'il existe nécessairement un autre mot temporisé de $L^*(\mathcal{P}_\mathcal{S} \parallel \mathcal{C})$ qui n'est pas dans $L^*(\varphi_\mathcal{S})$.

Soit Γ l'alphabet symbolique de $\mathcal{C}$ et soit m le plus petit multiple commun des dénominateurs des constantes apparaissant dans Γ (toute constante de Γ est alors un multiple entier de $\frac{1}{m}$). Soit $(s_0, \varepsilon) \xrightarrow{a_0} (s_1, c_1) \xrightarrow{a_1} \cdots$ l'exécution infinie de $\mathcal{S}$. Comme $\mathcal{S}$ a un canal divergent, il existe $n \in \mathbb{N}$ tel que $|c_n| > m \cdot |\Gamma| + 1$. De plus, comme $\mathcal{S}$ ne contient pas de cycle d'écriture, il existe $n' > n$ tel que toute lettre de c_n a été lue à la $n'^{\text{ème}}$ étape.

Soit $w' = (a_0, t_0)(a_1, t_1) \cdots (a_{2n'}, t_{2n'})$ le préfixe fini de w de longueur $2 \cdot n'$ (ce qui correspond au préfixe de longueur n' dans $\mathcal{S}$ car à toute action du contrôleur correspond une réponse *Nil* ou *Test* de l'environnement). Toutes les lettres sur le canal c_n doivent avoir été écrites entre les dates $t_{2n} - 1$ et t_{2n}. Il existe alors un intervalle de longueur $\frac{1}{m}$ à l'intérieur duquel il y a au moins $|\Gamma| + 1$ actions d'écriture. Formellement, il existe $i_0, i_1, \cdots, i_{|\Gamma|} < n$ tels que $a_{i_j} = m_{i_j}!$ pour des m_{i_j} donnés, et $t_{i_{|\Gamma|}} - t_{i_0} < \frac{1}{m}$.

Comme $\pi(w)$ est dans $L_{correct}$, pour tout $0 \leq j \leq |\Gamma|$ il existe une action $m_{i_j}?$ à la date $t_{i_j} + 1$. Soit i'_j l'indice de l'action qui se produit à la date $t_{i_j} + 1$ dans w'.

Considérons l'exécution de $\mathcal{P}_\mathcal{S} \parallel \mathcal{C}$ sur w' : soit $(g_0, a_0, Y_0) \cdot (g_1, a_1, Y_1) \cdots (g_{2n'}, a_{2n'}, Y_{2n'})$ le mot symbolique associé à w'. Nous considérons en particulier les gardes $g_{i'_j}$ (qui correspondent à l'action prise au temps $t_{i_j} + 1$). Nous prouvons tout d'abord que les gardes $g_{i'_0}$, $g_{i'_1}$, $\cdots$, $g_{i'_{|\Gamma|}}$ sont des gardes où le temps ne peut pas s'écouler (c'est-à-dire que pour tout $0 \leq j \leq |\Gamma|$, pour tout $v \in [\![g_{i'_j}]\!]$ et pour tout $t \in \mathbb{T}$, $v + t \notin [\![g_{i'_j}]\!]$). Si ce n'était pas le cas pour une $g_{i'_{|\Gamma|}}$ alors l'action $a_{i'_j}$ pourrait être prise un peu plus tard, à la date $t_{i_j} + 1 + \varepsilon$ pour un certain $\varepsilon > 0$, ce qui donnerait lieu à un mot temporisé appartenant à $L^*(\mathcal{P} \parallel \mathcal{C})$ mais pas à $L^*(\varphi_\mathcal{S})$.

Comme il y a $|\Gamma| + 1$ gardes $g_{i'_0}$, $g_{i'_1}$, $\cdots$, $g_{i'_{|\Gamma|}}$, il doit exister deux indices $k < l$ tels que $g_{i'_k} = g_{i'_l}$. Or $(t_{i_l} + 1) - (t_{i_k} + 1) < \frac{1}{m}$. Or le seul moyen de vérifier une garde où le temps ne peut pas s'écouler et ensuite de vérifier la même garde moins d'$\frac{1}{m}$ unité de temps plus tard est de ne pas laisser le temps s'écouler. Il y a donc contradiction car cela implique que deux actions contrôlables se produisent en même temps dans w'. Il ne peut donc exister de contrôleur de

$\mathcal{P}_{\mathcal{S}}$ pour la spécification $\varphi_{\mathcal{S}}$. Ceci achève la preuve de la proposition 6.2.4. $\square$

La proposition 6.2.4 montre que le problème de contrôle MTL sur les mots finis à ressources non-fixées pour des spécifications souhaitées est indécidable. Comme MTL est close par complémentaire, les problèmes de contrôle pour des spécifications souhaitées ou interdites sont équivalents.

Nous avons donc montré le théorème suivant :

Théorème 6.2.6 *Les problèmes de contrôle* MTL *sur les mots finis à ressources non-fixées pour des spécifications souhaitées ou interdites sont indécidables.*

6.2.3 Contrôle Safety-MTL sur les mots infinis

Nous adaptons maintenant la réduction précédente pour montrer que le problème de contrôle Safety-MTL sur les mots infinis pour des spécifications souhaitées est indécidable.

Nous ajoutons simplement une nouvelle action incontrôlable jouable dans q_{Fin} pour étendre les exécutions finies en exécutions infinies. La difficulté est de réécrire la spécification précédente dans Safety-MTL. Les formules φ_{sim} et φ_{avant} peuvent être réécrites dans Safety-MTL en remplaçant simplement les implications par leur définition. Nous expliquons maintenant comment remplacer la formule φ_{test} par une formule de Safety-MTL.

Rappelons que $\varphi_{\text{test}} = \bigwedge_{m \in M} \Big((\overline{\mathbf{F}}\,(m? \wedge \mathbf{F}_{=0}\,\mathit{Test})) \Rightarrow \overline{\mathbf{F}}\,(m! \wedge \mathbf{F}_{=1}\,\mathit{Test}) \Big)$.

Rappelons que $\varphi_{\text{action}} = \bigvee\limits_{c \in \Sigma} c$ et posons $\mathbf{X}_{act}\,\psi = (\neg\varphi_{\text{action}})\,\mathbf{U}\,\psi$; nous considérons alors $\varphi'_{\text{test}} = \bigwedge_{m \in M} \Big((\overline{\mathbf{F}}\,(m? \wedge \mathbf{F}_{=0}\,\mathit{Test})) \Rightarrow \varphi_0 \Big)$ [2] où

$$\varphi_0 = \Big(\overline{\mathbf{G}}_{<1}\,(\neg\,\mathit{Test})\Big) \wedge \Big(\neg\overline{\mathbf{F}}\,((\mathbf{F}_{>1}\;\mathit{Test}) \wedge (\,\mathbf{X}_{act}\,\overline{\mathbf{F}}_{<1}\;\mathit{Test}))\Big) \wedge \Big(\overline{\mathbf{G}}\,((\mathbf{F}_{=1}\;\mathit{Test}) \Rightarrow m!)\Big)$$

L'idée de la construction est que nous savons que l'action *Test* apparaîtra au plus une fois dans une exécution du système, donc nous voulons remplacer dans φ_{test} la formule $\overline{\mathbf{F}}\,(m! \wedge \mathbf{F}_{=1}\,\mathit{Test})$ (qui n'est pas dans Safety-MTL) par $\overline{\mathbf{G}}\,\big((\mathbf{F}_{=1}\;\mathit{Test}) \Rightarrow m!\big)$. Mais ce remplacement n'est pas correct car nous considérons la sémantique d'actions, il n'est correct que si une action se produit vraiment une unité de temps avant l'action *Test*. Pour assurer ce point, nous utilisons une technique similaire à celle que nous avons employée dans la section 5.2 pour exprimer $\varphi_{\text{arrière}}$ dans TPTL_{pw} : nous interdisons l'existence de deux actions consécutives où $\mathbf{F}_{>1}\;\mathit{Test}$ puis $\overline{\mathbf{F}}_{<1}\;\mathit{Test}$ sont vraies.

Il est important de remarquer que φ_0 n'est pas une formule équivalente à $\overline{\mathbf{F}}\,(m! \wedge \mathbf{F}_{=1}\,\mathit{Test})$ dans MTL : son utilisation est correcte car nous savons que l'action *Test* se produira au plus une fois dans une exécution du système. C'est l'objet du lemme facile suivant :

Lemme 6.2.7 *Soit $\sigma \in T\Sigma^\omega$ contenant l'action Test exactement une fois. Alors $\sigma \models \overline{\mathbf{F}}\,(m! \wedge \mathbf{F}_{=1}\,\mathit{Test})$ si, et seulement si $\sigma \models \varphi_0$.*

Corollaire 6.2.8 *Soit $\sigma \in T\Sigma^\omega$ contenant l'action Test au plus une fois. Alors $\sigma \models \varphi_{test}$ si, et seulement si $\sigma \models \varphi'_{test}$.*

[2] Pour des raisons de lisibilité nous n'écrivons pas φ'_{test} dans Safety-MTL, mais cela peut-être fait trivialement en poussant les négations à l'intérieur.

Nous avons donc construit un problème de contrôle avec une spécification dans Safety-MTL équivalent à l'accessibilité de $\mathcal{S}$, ce qui montre le théorème suivant :

Théorème 6.2.9 *Le problème de contrôle* Safety-MTL *sur les mots infinis à ressources non-fixées pour des spécifications souhaitées est indécidable.*

Remarque 6.2.10 *Dans la preuve nous n'avons pas utilisé la spécificité des mots infinis. En fait la spécification* Safety-MTL *que nous avons proposée fonctionne aussi pour le problème de contrôle* Safety-MTL *sur les mots finis, qui est aussi indécidable.*

Remarque 6.2.11 *Comme* Safety-MTL *n'est pas close par complémentaire, le problème de contrôle pour des spécifications interdites ne se réduit pas à celui pour des spécifications souhaitées. Le problème de contrôle* Safety-MTL *(sur les mots finis ou infinis) à ressources non-fixées pour des spécifications interdites est à ce jour ouvert.*

6.3 Contrôle à ressources fixées

Dans cette section, nous montrons qu'à ressources fixées, les problèmes de contrôle pour MTL sur les mots finis et Safety-MTL sur les mots infinis (pour des spécifications souhaitées ou interdites) sont décidables. Nous utilisons des techniques d'ordre partiel pour les systèmes temporisés, introduites initialement dans [AN01], puis utilisées dans le cadre des automates temporisés dans [OW04].

6.3.1 Jeux temporisés

Nous procédons comme dans [DM02] et réduisons le problème de contrôle temporisé (pour des spécifications quelconques) à un jeu temporisé entre deux joueurs. Dans [DM02], le jeu temporisé résultant est résolu en utilisant la construction de l'automate des régions et les jeux de parité. Dans notre cadre nous utiliserons des techniques différentes, nous exprimerons les spécifications MTL et Safety-MTL par des automates temporisés alternants à une horloge et utiliserons des méthodes d'ordre partiel.

Un tel jeu temporisé est joué sur un système de transitions symbolique $\mathcal{T} = (\Gamma, S, s_0, \rightarrow)$ avec une *fonction de validité* $val : 2^\Gamma \rightarrow 2^{(2^\Gamma)}$. Les deux joueurs C (pour contrôleur) et E (pour environnement) jouent alternativement mais leur rôle n'est pas symétrique. En effet, si le jeu se trouve dans l'état s, le joueur C choisit un sous-ensemble U de $val(\mathsf{Enb}_\mathcal{T}(s))$; le joueur E choisit alors un élément $u \in U$ et le jeu continue dans l'état successeur de s par u.

Les deux joueurs forment ainsi un mot γ. Si $tw(\gamma)$ vérifie la spécification, C est déclaré gagnant, sinon C est déclaré perdant.

Formellement, soit Γ un alphabet symbolique, une *fonction de validité* sur Γ est une fonction $val : 2^\Gamma \rightarrow 2^{(2^\Gamma)}$ telle que tout ensemble $V \in 2^\Gamma$ est envoyé sur un sous-ensemble non vide de sous-ensembles de V.

Soit $\mathcal{T}$ un système de transitions sur Γ et val une fonction de validité sur Γ, une *stratégie* de $\mathcal{T}$ respectant val est une fonction $f : D \subseteq L^*_{symb}(\mathcal{T}) \rightarrow 2^\Gamma$ satisfaisant les conditions suivantes :

- $\varepsilon \in D$
- si $\gamma \in D$ et $b \in f(\gamma)$ alors $\gamma \cdot b \in D$

– si $\gamma \in D$ alors $f(\gamma) \in val(\mathsf{Enb}_{\mathcal{T}}(\acute{e}tat_{\mathcal{T}}(\gamma)))$

Un mot $\gamma \in \Gamma^\infty$ est *compatible avec* f si pour tout préfixe $\gamma' \cdot b$ de γ, $b \in f(\gamma')$. Une stratégie f est *à états finis* s'il existe un système de transitions symbolique symboliquement déterministe $\mathcal{T}_{fin}$ avec un nombre fini d'états tel que pour tout $\gamma \in \Gamma^*$ compatible avec f, $f(\gamma)$ est l'ensemble des actions symboliques possibles dans l'état $\acute{e}tat_{\mathcal{T}_{fin}}(\gamma)$.

Définition 6.3.1 *Un* jeu temporisé sur les mots finis (resp. les mots infinis) *est un triplet* $\mathbb{G} = (\mathcal{A}, val, L)$ *où* $\mathcal{A}$ *est un automate temporisé déterministe atomique sur un alphabet symbolique* Γ *tel que* $\forall \gamma \in L_{symb}(\mathcal{A})$, $tw(\gamma) \neq \emptyset$*,* val *est une fonction de validité sur* Γ*, et* $L \subseteq T\Sigma^*$ *(resp.* $L \subseteq T\Sigma^\omega$*) est un langage temporisé de mots finis (resp. de mots infinis).*

Soit $\mathbb{G} = (\mathcal{A}, val, L)$ un jeu temporisé, une *stratégie de* $\mathbb{G}$ est une stratégie de $\mathcal{T}(\mathcal{A})$ respectant val. Si L est un langage temporisé de mots finis, une stratégie f de $\mathbb{G}$ est *gagnante pour des spécifications souhaitées* (resp. *interdites*) si, et seulement si tout mot $\gamma \in L^*_{symb}(\mathcal{A})$ compatible avec f vérifie $tw(\gamma) \subseteq L$ (resp. $tw(\gamma) \cap L = \emptyset$). La définition est similaire si L est un langage temporisé de mots infinis.

Un *jeu temporisé* sur MTL (resp. sur Safety-MTL) sur les mots finis est un jeu temporisé $\mathbb{G} = (\mathcal{A}, val, L)$ où L est l'ensemble des modèles d'une formule de MTL (resp. Safety-MTL) sur les mots finis. Nous définissons de manière similaire un *jeu temporisé* sur MTL (resp. sur Safety-MTL) sur les mots infinis.

En adaptant la construction de [DM02] à notre formalisme, nous obtenons que :

Proposition 6.3.2 *Soit* $\mathcal{P}$ *un système,* μ *une granularité plus fine que celle de* $\mathcal{P}$ *et* L *un langage temporisé de mots finis ou infinis.*

Alors nous pouvons construire un jeu temporisé $\mathbb{G} = (\mathcal{A}, val, L)$ *tel qu' il existe un* μ*-contrôleur (à états finis)* $\mathcal{C}$ *qui contrôle* $\mathcal{P}$ *pour des spécifications souhaitées (resp. interdites) si, et seulement s'il existe une stratégie gagnante (à états finis) de* $\mathbb{G}$ *pour des spécifications souhaitées (resp. interdites).*

La preuve de cette proposition est identique à celle de [DM02] : il s'agit tout d'abord de rendre les gardes de $\mathcal{P}$ μ-atomiques puis de construire l'automate des régions associé, et ensuite d'exprimer les conditions **(C1)**, **(C2)** et **(C3)** par une fonction de validité.

La proposition 6.3.2 assure qu'à ressources fixées, le problème de contrôle MTL sur les mots finis (resp. Safety-MTL sur les mots infinis) peut-être réduit à l'existence d'une stratégie gagnante dans un jeu temporisé sur MTL sur les mots finis (resp. Safety-MTL sur les mots infinis).

6.3.2 Automates temporisés alternants

Dans cette sous-section, nous introduisons les *automates temporisés alternants à une horloge* [OW05, LW05]. Nous notons x l'unique horloge de tels automates. Soit Q un ensemble fini, $\Phi(Q)$ désigne l'ensemble des formules définies par la grammaire suivante :

$$\psi \ ::= \psi \wedge \psi \ \mid \ \psi \vee \psi \ \mid \ q \ \mid \ x \sim k \ \mid \ x.\psi$$

où $q \in Q$, $k \in \mathbb{N}$, et $\sim \in \{<, \leq, =, \geq, >\}$. L'expression $x.\psi$ correspond à remettre l'horloge x à zéro.

Définition 6.3.3 *Un* automate temporisé alternant à une horloge (ATA_1) *est un n-uplet $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$ où Σ est un alphabet, Q un ensemble d'états, $q_0 \in Q$ un état initial, $F \subseteq Q$ un ensemble d'états accepteurs, et $\delta : Q \times \Sigma \to \Phi(Q)$ une fonction de transition.*

Soit c_{max} une constante plus grande que la constante maximale apparaissant dans la fonction de transition d'un ATA_1 $\mathcal{A}$. Suivant [OW07], nous définissons l'ensemble des valeurs d'holorge de $\mathcal{A}$ comme $\mathrm{Val} = [0, c_{max}] \cup \{\top\}$. Nous étendons les propriétés arithmétiques à $\top$ par $v + t = \top$ si $v, t \in \mathbb{T}$ et $v + t > c_{max}$; $\top + t = \top$ pour $t \in \mathbb{T}$ et $\top > v$ pour tout $v \in \mathrm{Val}$. Dans ce chapitre nous utilisons un formalisme similaire pour les automates temporisés.

Remarque 6.3.4 *Le symbole $\top$ représente toute valeur d'horloge plus grande que c_{max}. Cette identification n'a pas d'incidence puisque ces valeurs d'horloge sont indistingables par les gardes. Cette représentation nous permettra de présenter certains résultats comme la proposition 6.3.7 plus clairement.*

Une *configuration* de $\mathcal{A}$ est un ensemble fini de couples (q, u) où $q \in Q$ est un état et $u \in \mathrm{Val}$ une valeur d'horloge. La *configuration initiale* est $\{(q_0, 0)\}$. Une configuration C est acceptante si pour tout $(q, u) \in C$, $q \in F$ (remarquons que la configuration vide est acceptante).

Étant donnée une valeur d'horloge u, nous définissons la relation de satisfaction $\models_u$ entre configurations et formules de $\Phi(Q)$ inductivement comme suit :

$C \models_u q$	si	$(q, u) \in C$
$C \models_u x \sim k$	si	$u \sim k$
$C \models_u x.\psi$	si	$C \models_0 \psi$
$C \models_u \psi_1 \vee \psi_2$	si	$C \models_u \psi_1$ ou $C \models_u \psi_2$
$C \models_u \psi_1 \wedge \psi_2$	si	$C \models_u \psi_1$ et $C \models_u \psi_2$

Un ATA_1 $\mathcal{A}$ est *complet* si pour tout $q \in Q$, $a \in \Sigma$, et $u \in \mathrm{Val}$, il existe une configuration C telle que $C \models_u \delta(q, a)$. Nous disons qu'une configuration M est un *modèle minimal* de $\psi \in \Phi(Q)$ par rapport à $u \in \mathrm{Val}$ si $M \models_u \psi$ et s'il n'existe pas de sous-ensemble strict C de M avec $C \models_u \psi$.

Soit $C = \{(q_i, u_i)\}_{i \in I}$ et C' deux configurations. Pour $a \in \Sigma$, $t \in \mathbb{T}$, nous définissons la relation de transition entre configurations par $C \xrightarrow{a,t} C'$ si $C' = \bigcup_{i \in I} \{M_i \mid M_i$ est un modèle minimal de $\delta(q_i, a)$ par rapport à $u_i + t\}$.

Une exécution sur un mot temporisé $\sigma = (a_0, \tau_0)(a_1, \tau_1) \cdots (a_n, t_n)$ est une suite $C_0 \xrightarrow{a_0,d_0} C_1 \xrightarrow{a_1,d_1} \cdots \xrightarrow{a_n,d_n} C_n$ telle que C_0 est la configuration initiale et pour tout $0 \leq i \leq n$, $d_i = \tau_i - \tau_{i-1}$ (en posant $\tau_{-1} = 0$). Une telle exécution est *acceptante* si C_n est acceptante.

Le langage d'un ATA_1 $\mathcal{A}$ est l'ensemble des mots temporisés sur lesquels $\mathcal{A}$ a une exécution acceptante, il est noté $L^*(\mathcal{A})$.

Exemple 6.3.5 *Il existe un* ATA_1 *reconnaissant le même langage que la formule* $\mathbf{G}\,(a \Rightarrow \mathbf{F}_{=1} b)$ *de l'exemple 5.1.3. Soit l'automate temporisé alternant à une horloge ayant pour ensemble d'états $\{q_0, q_1\}$ avec q_0 initial et final, et la fonction de transition δ donnée par :*

- $\delta(q_0, a) = q_0 \wedge x.q_1$
- $\delta(q_0, b) = q_0$
- $\delta(q_1, a) = q_1$
- $\delta(q_1, b) = (x = 1) \vee q_1$

L'automate se trouve constamment dans l'état q_0, et quand un a se produit une copie de q_1 est lancée ; cette copie ne peut être acceptante que si un b se produit une unité de temps plus tard.

6.3.3 Configurations abstraites

Dans cette partie, nous rappelons certains résultats de [OW05] et montrons certaines propriétés qui seront la base de la résolution du contrôle MTL et Safety-MTL à ressources fixées. Notre approche utilise des beaux-ordres et des meilleurs-ordres, nous commençons donc par rappeler des résultats classiques dans ce domaine.

Un *ordre partiel* est un couple $(S, \preceq)$ où S est un ensemble et $\preceq$ est une relation binaire réflexive et transitive sur S. Un *bel-ordre partiel* est un ordre partiel $(S, \preceq)$ tel que pour toute suite infinie $x_0, x_1, x_2, \cdots$ d'éléments de S, il existe des indices $i < j$ tels que $x_i \preceq x_j$.

Étant donné un ordre partiel $(S, \preceq)$, nous serons intéressés par les ordres partiels déduits de $(S, \preceq)$ suivants :

- *l'ordre de domination monotone* est l'ordre partiel $(S^*, \preceq^*)$, où $x_1 \cdots x_m \preceq^* y_1 \cdots y_n$ si, et seulement s'il existe une injection monotone $h : \{1, \cdots, m\} \to \{1, \cdots, n\}$ telle que pour tout $1 \leq i \leq m,\ x_i \preceq y_{h(i)}$;
- *l'ordre des parties* est l'ordre partiel $(2^S, \sqsubseteq)$, où pour tout $S_1, S_2 \subseteq S$, $S_1 \sqsubseteq S_2$ si, et seulement si $\forall x_2 \in S_2\ \exists x_1 \in S_1\ x_1 \preceq x_2$.

Un *meilleur-ordre partiel* est un bel-ordre partiel ayant de plus fortes propriétés. Nous ne rappelons par la définition de meilleur-ordre partiel (voir par exemple [AN00]) car celle-ci est assez technique et nous ne ferons qu'utiliser des résultats connus sur les meilleurs-ordres partiels.

Proposition 6.3.6 ([AN00, AN01])

1. *Tout meilleur-ordre partiel est un bel-ordre partiel.*
2. *Si S est fini, alors $(2^S, \subseteq)$ est un meilleur-ordre partiel.*
3. *Si $(S, \preceq)$ est un meilleur-ordre partiel, alors $(S^*, \preceq^*)$ est un meilleur ordre partiel.*
4. *Si $(S, \preceq)$ est un meilleur ordre partiel, alors $(2^S, \sqsubseteq)$ est un meilleur ordre partiel.*

Nous rappelons maintenant le codage des configurations d'un automate temporisé synchronisé avec un automate temporisé alternant à une horloge [LW05, OW05].

Nous fixons un automate temporisé déterministe atomique $\mathcal{A} = (\Sigma, X, Q, q_0, F^{\mathcal{A}}, R^{\mathcal{A}}, \delta^{\mathcal{A}})$ sur un alphabet symbolique Γ de granularité $\mu = (X, m, M)$, et un ATA_1 complet $\mathcal{B} = (P, p_0, F^{\mathcal{B}}, \delta^{\mathcal{B}})$ dont l'unique horloge est x.

Une $\mathcal{A}/\mathcal{B}$-configuration est un couple $((q, \nu), G)$, où (q, ν) est une configuration de $\mathcal{A}$ et G est une configuration de $\mathcal{B}$. Pour une $\mathcal{A}/\mathcal{B}$-configuration $((q, \nu), G)$, $t \in \mathbb{T}$, et une action symbolique $(g, a, Y) \in \Gamma$, nous définissons

$$\begin{aligned} Succ^{\mathcal{A}}((q,\nu), t, (g,a,Y)) &:= \{(q',\nu') \mid (q,\nu) \xrightarrow[t]{g,a,Y} (q',\nu')\}\ [3] \\ Succ^{\mathcal{B}}(G, t, a) &:= \{G' \mid G \xrightarrow{a,t} G'\} \end{aligned}$$

Le *produit synchronisé* de $\mathcal{A}$ et $\mathcal{B}$ est un système de transitions symbolique infini non-dénombrable sur Γ, noté $\mathcal{T}_{\mathcal{A}/\mathcal{B}}$, représentant intuitivement $\mathcal{A}$ et $\mathcal{B}$ s'exécutant en parallèle. Formellement, $\mathcal{T}_{\mathcal{A}/\mathcal{B}} = (\Gamma, S, s_0, \twoheadrightarrow)$, où

- S est l'ensemble des $\mathcal{A}/\mathcal{B}$-configurations
- $s_0 = ((q_0, \mathbf{0}), \{p_0, 0\})$ correspond à la $\mathcal{A}/\mathcal{B}$-configuration initiale

[3] C'est-à-dire que $q \xrightarrow{g,a,Y} q'$ est une transition de $\mathcal{A}$, $\nu + t \in [\![g]\!]$, et $\nu' = (\nu + t)[Y \leftarrow 0]$.

– $((q_1,\nu_1),G_1) \xrightarrow{g,a,Y} ((q_2,\nu_2),G_2)$ si

$$\exists t \in \mathbb{T} \text{ tel que } \begin{cases} (q_2,\nu_2) \in Succ^{\mathcal{A}}((q_1,\nu_1),t,(g,a,Y)) \\ G_2 \in Succ^{\mathcal{B}}(G_1,t,a) \text{ et} \end{cases}$$

Nous rappelons maintenant la construction d'un automate des région généralisé pour abstraire les valeurs précises des horloges des $\mathcal{A}/\mathcal{B}$-configurations, en ne conservant que des valeurs entières et l'ordre relatif des parties fractionnaires des horloges. Nous rappelons la construction d'une relation d'équivalence, qui est une bisimulation sur $\mathcal{T}_{\mathcal{A}/\mathcal{B}}$, en codant des ensembles d'$\mathcal{A}/\mathcal{B}$-configurations par des mots.

Soit K un entier plus grand que M et plus grand que la plus grande constante apparaissant dans les transitions de $\mathcal{B}$. Nous considérons $c_{max} = K$ et nous définissons REG_K l'ensemble des régions unidimensionnelles $\{r_0, r_1, \cdots, r_{2K+1}\}$: pour $0 \leq i \leq K$, $r_{2i} = \{i\}$, $r_{2i+1} =]i, i+1[$, et $r_{2K+1} = \{\top\}$. Pour $u \in \mathrm{Val}$, $reg(u)$ est l'unique région de REG_K contenant u.

Nous définissons l'alphabet $\Lambda = 2^{(Q\times X\times REG_K)\cup(P\times REG_K)}$: une lettre de Λ est un ensemble fini de paires (p,r) et de triplets (q,y,r), où q et p sont des états de $\mathcal{A}$ et $\mathcal{B}$ respectivement, $y \in X$ est une horloge de $\mathcal{A}$, et r est une région unidimensionnelle de REG_K. De plus nous notons $(\Lambda^*, \preceq)$ l'ordre de domination monotone induit par le meilleur-ordre partiel $(\Lambda, \subseteq)$, et par $(2^{\Lambda^*}, \sqsubseteq)$ l'ordre des parties induit par $(\Lambda^*, \preceq)$. Par la proposition 6.3.6, $(\Lambda^*, \preceq)$ et $(2^{\Lambda^*}, \sqsubseteq)$ sont des beaux-ordres partiels[4].

Nous montrons maintenant comment associer à une $\mathcal{A}/\mathcal{B}$-configuration $s = ((q,\nu),G)$ un mot $H(s) \in \Lambda^*$. Remarquons tout d'abord que s peut être représentée comme l'ensemble G' donné par $G \cup \{(q,y,\nu(y)) \mid y \in X\}$.

Nous partitionnons G' en une suite de sous-ensembles non-vides $G_1, \cdots, G_n$ tels que pour tout $1 \leq i \leq j \leq n$, pour toute paire (p,u) ou triplet (q,y,u) de G_i, et pour toute paire (p',v) ou triplet (q',y',v) de G_j, la condition suivante est vérifiée : $i \leq j$ si, et seulement si $\langle u \rangle \leq \langle v \rangle$ [5].

Nous définissons $H(s)$ comme le mot de Λ^* donné par $Abs(G_1)\cdots Abs(G_n)$, où pour tout $1 \leq i \leq n$,

$$Abs(G_i) = \{(p, reg(u)) \mid (p,u) \in G_i\} \cup \{(q,y,reg(u)) \mid (q,y,u) \in G_i\}$$

Nous disons que deux $\mathcal{A}/\mathcal{B}$-configurations s et s' sont équivalentes, et écrivons $s \sim s'$, si $H(s) = H(s')$.

Proposition 6.3.7 ([OW05, OW07]) *La relation $\sim$ est une bisimulation sur $\mathcal{T}_{\mathcal{A}/\mathcal{B}}$, c'est-à-dire que $s_1 \sim s_1'$ et $s_1 \xrightarrow{g,a,Y} s_2$ impliquent qu'il existe une $\mathcal{A}/\mathcal{B}$-configuration s_2' telle que $s_1' \xrightarrow{g,a,Y} s_2'$ et $s_2 \sim s_2'$.*

Le quotient induit par la bisimulation $\sim$ sur $\mathcal{T}_{\mathcal{A}/\mathcal{B}}$ est le système de transitions symbolique $\mathcal{T}_\sim = (\Gamma, W, w_0, \hookrightarrow)$ défini par :
- $W = \{H(s) \mid s \text{ est une } \mathcal{A}/\mathcal{B}\text{-configuration}\}$
- $w_0 = H(s_0)$

[4] Ce sont même des meilleurs-ordres partiels, mais nous n'aurons pas besoin de cette propriété.
[5] $\langle u \rangle$ est la partie fractionnaire de u.

– $w_1 \xhookrightarrow{g,a,Y} w_2$ si, et seulement s'il existe $s_1 \in H^{-1}(\{w_1\})$ et $s_2 \in H^{-1}(\{w_2\})$ tels que $s_1 \xrightarrow{g,a,Y} s_2$.

Proposition 6.3.8 ([OW05]) *1. Pour tout $w \in W$, l'ensemble des successeurs de w dans $\mathcal{T}_\sim$ est fini et calculable.*

2. La relation de transition $\hookrightarrow$ de $\mathcal{T}_\sim$ est compatible vers le bas *par rapport à $\preceq$, c'est-à-dire que $w'_1 \preceq w_1$ et $w_1 \xhookrightarrow{g,a,Y} w_2$ impliquent qu'il existe $w'_2 \preceq w_2$ avec $w'_1 \xhookrightarrow{g,a,Y} w'_2$.*

Nous étendons maintenant la proposition précédente à la déterminisation $Det(\mathcal{T}_\sim)$ de $\mathcal{T}_\sim$.

Remarque 6.3.9 *Le système de transitions $Det(\mathcal{T}_\sim)$ jouera un rôle important dans la suite de ce chapitre, car contrairement aux problèmes d'accessibilité ou satisfaisabilité, le problème de contrôle dépend de l'ensemble des états dans lesquels peut se trouver le système ; c'est pour cela que nous devons déterminiser $\mathcal{T}_\sim$.*

Nous commençons par remarquer que nous pouvons associer à un mot $w \in W$ une unique région de $\mathcal{A}$ au sens classique du terme [AD94]. Considérons en effet le sous-mot maximal de w ne contenant pas d'occurrence d'états de $\mathcal{B}$, nous notons $reg_{\mathcal{A}}(w)$ ce sous-mot.

Comme $\mathcal{B}$ est complet et $\mathcal{A}$ est atomique et déterministe, nous remarquons que pour tout $w_1, w_2 \in W$ avec $reg_{\mathcal{A}}(w_1) = reg_{\mathcal{A}}(w_2)$, si $w_1 \xhookrightarrow{g,a,Y} w'_1$ et $w_2 \xhookrightarrow{g,a,Y} w'_2$ alors $reg_{\mathcal{A}}(w'_1) = reg_{\mathcal{A}}(w'_2)$. Il s'ensuit que tout état $\mathcal{C} \subseteq W$ de $Det(\mathcal{T}_\sim)$ accessible à partir de la configuration initiale a la propriété que pour tout $w, w' \in \mathcal{C}$, $reg_{\mathcal{A}}(w) = reg_{\mathcal{A}}(w')$. Nous notons SW l'ensemble des états non vides $\mathcal{C} \subseteq W$ ayant cette propriété. Nous définissons le système de transitions symbolique $\mathcal{DT}_\sim = (\Gamma, SW, \{w_0\}, \hookrightarrow_{\mathcal{D}})$ la restriction de $Det(\mathcal{T}_\sim)$ à l'ensemble des états de SW. Par les remarques précédentes, nous avons que $L^*_{symb}(\mathcal{DT}_\sim) = L^*_{symb}(Det(\mathcal{T}_\sim))$.

Proposition 6.3.10 *1. Pour tout $\mathcal{C}_1, \mathcal{C}_2 \in SW$, si $\mathcal{C}_1 \sqsubseteq \mathcal{C}_2$, alors $\mathsf{Enb}_{\mathcal{DT}_\sim}(\mathcal{C}_1) = \mathsf{Enb}_{\mathcal{DT}_\sim}(\mathcal{C}_2)$.*

2. La relation de transition $\hookrightarrow_{\mathcal{D}}$ de $\mathcal{DT}_\sim$ est compatible vers le bas *par rapport à $\sqsubseteq$, c'est-à-dire que $\mathcal{C}'_1 \sqsubseteq \mathcal{C}_1$ et $\mathcal{C}_1 \xhookrightarrow{g,a,Y}_{\mathcal{D}} \mathcal{C}_2$ impliquent qu'il existe $\mathcal{C}'_2 \sqsubseteq \mathcal{C}_2$ tel que $\mathcal{C}'_1 \xhookrightarrow{g,a,Y}_{\mathcal{D}} \mathcal{C}'_2$.*

Comme $\mathcal{B}$ est complet, les actions possibles à partir de $\mathcal{C} \in SW$ ne dépendent que de la $\mathcal{A}$-composante des mots de $\mathcal{C}$. Si $\mathcal{C}_1 \sqsubseteq \mathcal{C}_2$, pour tout $w_1 \in \mathcal{C}_1$ et $w_2 \in \mathcal{C}_2$, $reg_{\mathcal{A}}(w_1) = reg_{\mathcal{A}}(w_2)$ donc $\mathsf{Enb}_{\mathcal{DT}_\sim}(\mathcal{C}_1) = \mathsf{Enb}_{\mathcal{DT}_\sim}(\mathcal{C}_2)$.

Nous prouvons maintenant la seconde propriété. Par la première propriété et le fait que $\mathcal{DT}_\sim$ est symboliquement déterministe, il y a un unique $\mathcal{C}'_2 \in SW$ tel que $\mathcal{C}'_1 \xhookrightarrow{g,a,Y}_{\mathcal{D}} \mathcal{C}'_2$. Nous devons montrer que $\mathcal{C}'_2 \sqsubseteq \mathcal{C}_2$. Soit $w_2 \in \mathcal{C}_2$, nous montrons qu'il existe $w'_2 \in \mathcal{C}'_2$ tel que $w'_2 \preceq w_2$. Comme $\mathcal{C}_1 \xhookrightarrow{g,a,Y}_{\mathcal{D}} \mathcal{C}_2$, il existe $w_1 \in \mathcal{C}_1$ tel que $w_1 \xhookrightarrow{g,a,Y} w_2$. $\mathcal{C}'_1 \sqsubseteq \mathcal{C}_1$ implique qu'il existe $w'_1 \in \mathcal{C}'_1$ tel que $w'_1 \preceq w_1$. Par la seconde propriété de la proposition 6.3.8, il existe un mot w'_2 tel que $w'_2 \preceq w_2$ et $w'_1 \xhookrightarrow{g,a,Y} w'_2$. Alors comme $w'_1 \in \mathcal{C}'_1$ et $\mathcal{C}'_1 \xhookrightarrow{g,a,Y}_{\mathcal{D}} \mathcal{C}'_2$, nous avons que $w'_2 \in \mathcal{C}'_2$.

6.3.4 Contrôle MTL sur les mots finis

Dans cette sous-section, nous montrons que le contrôle MTL sur les mots finis à ressources fixées est décidable. Comme MTL est close par négation, il nous suffit de montrer que le problème de contrôle pour des spécifications interdites est décidable.

Nous fixons un jeu temporisé sur MTL sur les mots finis $\mathbb{G} = (\mathcal{A}, val, L^*(\varphi))$. Soit Γ l'alphabet symbolique de $\mathcal{A}$. En appliquant les résultats de [OW05], nous pouvons construire un ATA_1 complet $\mathcal{B}_\varphi = (\Gamma, P, p_0, F^\varphi, \delta)$ tel que $L^*(\mathcal{B}_\varphi) = L^*(\varphi)$.

Soit $\mathcal{T}_{\mathcal{A}/\varphi}$ le produit synchronisé de $\mathcal{A}$ et $\mathcal{B}_\varphi$ décrit dans la sous-section 6.3.3, $\mathcal{T}_\sim = (\Gamma, W, w_0, \hookrightarrow)$ et $\mathcal{DT}_\sim = (\Gamma, SW, \{w_0\}, \hookrightarrow_\mathcal{D})$ le système de transitions symbolique induit par $\mathcal{T}_{\mathcal{A}/\varphi}$ défini précédemment.

Nous disons qu'une $\mathcal{A}/\mathcal{B}_\varphi$ configuration $((q, \nu), G)$ est *mauvaise* si q et G sont tous deux accepteurs. Comme nous sommes dans le cadre du contrôle pour des spécifications interdites, une telle configuration n'est pas souhaitable car elle correspond à un état accepteur de $\mathcal{A}$ où φ est vérifiée. Un mot $w \in W$ est *mauvais* s'il existe $s \in H^{-1}(\{w\})$ tel que s est une mauvaise configuration (remarquons que dans ce cas toute configuration $s \in H^{-1}(\{w\})$ est mauvaise). Un ensemble de mots $\mathcal{C} \in SW$ est *mauvais* si $\mathcal{C}$ contient un mauvais mot.

Une stratégie f de $\mathcal{DT}_\sim$ respectant *val* [6] est *sûre* si pour tout mot γ compatible avec f, $\textit{état}_{\mathcal{DT}_\sim}(\gamma)$ n'est pas mauvais. Le lemme suivant montre que $\mathcal{DT}_\sim$ est une bonne abstraction de $\mathbb{G}$.

Lemme 6.3.11 *Il existe une stratégie gagnante (à états finis) du jeu temporisé $\mathbb{G}$ pour des spécifications interdites si, et seulement s'il existe une stratégie sûre (à états finis) de $\mathcal{DT}_\sim$.*

Preuve. Comme $\mathcal{B}_\varphi$ est complet, nous avons que $L^*_{symb}(\mathcal{T}(\mathcal{A})) = L^*_{symb}(\mathcal{T}_{\mathcal{A}/\varphi})$. Donc pour toute $f : D \subseteq \Gamma^* \to 2^\Gamma$, f est une stratégie de $\mathbb{G}$ si, et seulement si f est une stratégie de $\mathcal{DT}_\sim$.

Soit f une stratégie gagnante de $\mathbb{G}$ pour des spécifications interdites, montrons que f est une stratégie sûre de $\mathcal{DT}_\sim$. En effet supposons par l'absurde qu'il existe $\gamma \in \Gamma^*$ compatible avec f tel que $\textit{état}_{\mathcal{DT}_\sim}(\gamma)$ soit mauvais. Par définition il existe un mot $w \in \textit{état}_{\mathcal{DT}_\sim}(\gamma)$ mauvais ; alors comme $\mathcal{DT}_\sim$ est construit par la méthode des sous-ensembles, il existe un chemin de w_0 à w dans $\mathcal{T}_\sim$. Par la proposition 6.3.7 il existe un chemin dans $\mathcal{T}_{\mathcal{A}/\varphi}$ de la configuration initiale à une mauvaise configuration étiqueté γ. Nous avons donc que $\gamma \in L^*_{symb}(\mathcal{A})$ et $tw(\gamma) \cap L^*(\varphi) \neq \emptyset$, ce qui est absurde.

De manière similaire, si f est une stratégie sûre de $\mathbb{G}$ pour des spécifications interdites, alors f est une stratégie gagnante de $\mathcal{DT}_\sim$. $\square$

Le lemme 6.3.11 montre que pour décider l'existence d'une stratégie gagnante dans le jeu temporisé $\mathbb{G}$ pour des spécifications interdites, il est équivalent de décider l'existence d'une stratégie sûre de $\mathcal{DT}_\sim$. Nous montrons maintenant que ce problème est décidable, en étendant l'approche utilisée dans [ABd03] pour les jeux clos par le bas. La correction et la terminaison de notre algorithme repose sur le bel-ordre partiel $(SW, \sqsubseteq)$.

Nous construisons une partie T de l'arbre (infini dénombrable) correspondant au dépliage de $\mathcal{DT}_\sim$ depuis l'état initial $\{w_0\}$. L'arbre T est construit de la manière suivante : nous commençons avec la racine étiquetée $\{w_0\}$, et à chaque étape nous prenons une feuille x étiquetée $\mathcal{C} \in SW$ et nous faisons l'une des actions suivantes :

- si l'ensemble $\mathcal{C}$ est mauvais, nous déclarons le nœud *mauvais* et nous le fermons (c'est-à-dire que nous ne déplions plus l'arbre à partir de ce nœud),
- si l'ensemble $\mathcal{C}$ n'est pas mauvais et qu'il existe un ancêtre de x étiqueté $\mathcal{C}'$ avec $\mathcal{C}' \sqsubseteq \mathcal{C}$, nous déclarons le nœud *bon* et nous le fermons,

[6] Dans la suite nous omettons la référence à *val* quand c'est clair dans le contexte.

– sinon, pour toute transition de $\mathcal{DT}_\sim$ de la forme $\mathcal{C} \xhookrightarrow{g,a,Y}_{\mathcal{D}} \mathcal{C}'$ nous ajoutons un nouveau nœud y étiqueté $\mathcal{C}'$ et une arête de x à y étiquetée (g, a, Y). Si $\mathcal{C}$ n'a pas de successeur, nous déclarons x *mort* et le fermons.

Remarque 6.3.12 *La construction est un dépliage standard, sauf pour le deuxième point : si nous trouvons une configuration $\mathcal{C}$ telle que nous avons déjà exploré une configuration $\mathcal{C}' \sqsubseteq \mathcal{C}$, nous considérons la configuration $\mathcal{C}$ comme gagnante. En effet pour tout mot $w \in \mathcal{C}$, il existe alors un mot $w' \in \mathcal{C}'$ avec $w' \preceq w$. Le mot w' a moins d'états de $\mathcal{B}$ que w et donc est plus susceptible de mener à une configuration mauvaise (qui est une configuration où $\mathcal{A}$ et $\mathcal{B}$ sont tous deux accepteurs). Intuitivement $\mathcal{C}$ est « moins dangereux » que $\mathcal{C}'$ et n'a donc pas besoin d'être exploré.*

La terminaison de cette algorithme est assurée par le lemme de König et le fait que $(SW, \sqsubseteq)$ est un bel-ordre partiel. En effet, le dépliage de $\mathcal{DT}_\sim$ est à branchement fini donc si T était infini il y aurait un chemin infini de T, $\mathcal{C}_0 \xhookrightarrow{g_0,a_0,Y_0}_{\mathcal{D}} \mathcal{C}_1 \xhookrightarrow{g_1,a_1,Y_1}_{\mathcal{D}} \cdots$. Par construction ce chemin vérifierait que pour tout $i < j$, $\mathcal{C}_i \not\sqsubseteq \mathcal{C}_j$ ce qui contredit le fait que $(SW, \sqsubseteq)$ est un bel-ordre partiel.

Nous attribuons ensuite à tout nœud de T une valeur dans $\{\top, \bot\}$ de manière classique à partir des feuilles :

– les nœuds bons et morts ont pour valeur $\top$; les nœuds mauvais ont pour valeur $\bot$;
– pour tout nœud interne x étiqueté $\mathcal{C}$ dont la valeur de tous les successeurs est déterminée, la valeur de x est attribuée comme suit :
  – s'il existe un ensemble d'actions symboliques $U \in \mathit{val}(\mathsf{Enb}_{\mathcal{DT}_\sim}(\mathcal{C}))$ tel que pour tout $(g, a, Y) \in U$, l'arête de T de x étiquetée (g, a, Y) mène à un nœud de valeur $\top$, alors la valeur de x est $\top$,
  – sinon la valeur de x est $\bot$.

Notre algorithme répond « oui » si la racine a pour valeur $\top$, et répond « non » sinon.

La correction de l'algorithme est assurée par la proposition suivante :

Proposition 6.3.13 *Si l'algorithme répond « non », alors il n'existe pas de stratégie sûre de $\mathcal{DT}_\sim$.*
Si l'algorithme répond « oui », alors il existe une stratégie sûre de $\mathcal{DT}_\sim$ à états finis, et celle-ci peut être construite effectivement.

Preuve. Si l'algorithme répond « non », la racine a pour valeur $\bot$; il est alors facile de voir que quoi que fasse le contrôleur, l'environnement peut toujours choisir un sucesseur étiqueté $\bot$. Comme T est fini, un mot maximal compatible avec de tels choix de l'environnement atteint un mauvais état, donc il ne peut y avoir de stratégie sûre de $\mathcal{DT}_\sim$.

Réciproquement supposons que l'algorithme réponde « oui ». Soit T' l'arbre obtenu à partir de T en supprimant tous les états de valeur $\bot$. Pour tout nœud x de T', nous notons $\mathcal{C}(x)$ l'état de $\mathcal{DT}_\sim$ associé à x. Comme la racine a pour valeur $\top$, il existe un arbre fini T_{strat} (pas nécessairement unique) satisfaisant la condition suivante : si x est un nœud de T_{strat} (et n'est pas bon), alors l'ensemble des arêtes de T_{strat} partant de x est un sous-ensemble des arêtes de T' partant de x tel que l'ensemble des étiquettes de ces arêtes appartient à $\mathit{val}(\mathsf{Enb}_{\mathcal{DT}_\sim}(\mathcal{C}(x)))$.

Remarque 6.3.14 *Intuitivement l'arbre T' représente à chaque étape l'ensemble des actions qui permettent au contrôleur de gagner ; nous extrayons de T' un sous-arbre T_{strat} proposant à chaque étape une unique action contrôlable (ainsi que toutes les actions incontrôlables possibles à ce moment). Ainsi l'arbre T_{strat} correspond « presque » à une stratégie (il nous reste à déterminer ce que fait cette stratégie lorsque qu'elle atteint un état $\mathcal{C}' \sqsubseteq \mathcal{C}$, ce que nous faisons ci-dessous).*

Nous construisons un système de transitions symbolique à partir de T_{strat} qui nous permettra de définir aisément la stratégie (sûre) associée à cet arbre. Ce système de transitions correspond à identifier les nœuds de T_{strat} étiquetés $\mathcal{C}$ où $\mathcal{C}' \sqsubseteq \mathcal{C}$ avec un nœud ancêtre étiqueté $\mathcal{C}'$. D'un point de vue de contrôle, lorsqu'il se trouve en $\mathcal{C}$, le contrôleur fait *comme s'il était en $\mathcal{C}$'.*
Soit $T_{strat} = (\Gamma, X, x_0, \to)$ le système de transitions symbolique symboliquement déterministe à états finis défini par :
- X est l'ensemble des nœuds de T_{strat} qui ne sont pas bons,
- x_0 est la racine de T_{strat},
- $x \xrightarrow{g,a,Y} x'$ si l'une des conditions suivantes est vérifiée :
  - il y a une arête de T_{strat} de x à x' étiquetée (g, a, Y),
  - il y a une bonne feuille y de T_{strat} telle que x est le père de y, il existe une arête de x à y étiquetée (g, a, Y), et x' est le plus jeune ancêtre de y tel que $\mathcal{C}(x') \sqsubseteq \mathcal{C}(y)$ (dans ce cas nous identifions y et x').

Nous pouvons maintenant définir notre candidat de stratégie sûre de $\mathcal{DT}_\sim$ respectant val. $f : L^*_{symb}(T_{strat}) \to 2^\Gamma$ est définie par : pour tout $\gamma \in L^*_{symb}(T_{strat})$, $f(\gamma) = \mathsf{Enb}_{T_{strat}}(\acute{e}tat_{T_{strat}}(\gamma))$.
Nous montrons le lemme suivant qui assure que f est sûre et respecte val :

Lemme 6.3.15 *Soit $\rho = \mathcal{C}_0 \xhookrightarrow{g_1,a_1,Y_1}_{\mathcal{D}} \mathcal{C}_1 \cdots \xhookrightarrow{g_n,a_n,Y_n}_{\mathcal{D}} \mathcal{C}_n$ une exécution de $\mathcal{DT}_\sim$ compatible avec f.*

Alors il existe une exécution $x_0 \xrightarrow{g_1,a_1,Y_1} x_1 \xrightarrow{g_2,a_2,Y_2} \cdots \xrightarrow{g_n,a_n,Y_n} x_n$ de T_{strat} avec pour tout $0 \le i \le n$ $\mathcal{C}(x_i) \sqsubseteq \mathcal{C}_i$ et $f(\rho_i) \in val(\mathsf{Enb}_{\mathcal{DT}_\sim}(\mathcal{C}_i))$ [7].

Preuve. Nous raisonnons par récurrence sur n. Pour $n = 0$ le lemme est évident.

Supposons le lemme vrai à l'indice n, et montrons-le à l'indice $n+1$. Soit $g_{n+1}, a_{n+1}, Y_{n+1}$ et $\mathcal{C}_{n+1}$ tels que $\mathcal{C}_n \xhookrightarrow{g_{n+1},a_{n+1},Y_{n+1}}_{\mathcal{D}} \mathcal{C}_{n+1}$ soit compatible avec f. Par définition de f, il existe $x_{n+1} \in X$ tel que $x_n \xrightarrow{g_{n+1},a_{n+1},Y_{n+1}} x_{n+1}$ dans T_{strat}. Nous distinguons deux cas :
- si la transition $x_n \xrightarrow{g_{n+1},a_{n+1},Y_{n+1}} x_{n+1}$ provient d'une transition $\mathcal{C}(x_n) \xhookrightarrow{g_{n+1},a_{n+1},Y_{n+1}}_{\mathcal{D}} \mathcal{C}(x_{n+1})$ de $\mathcal{DT}_\sim$.
  Alors $\mathcal{C}(x_n) \sqsubseteq \mathcal{C}_n$ implique par la proposition 6.3.10 qu'il existe $\mathcal{C}' \in SW$ tel que $\mathcal{C}(x_n) \xhookrightarrow{g_{n+1},a_{n+1},Y_{n+1}}_{\mathcal{D}} \mathcal{C}'$ et $\mathcal{C}' \sqsubseteq \mathcal{C}_{n+1}$. Comme $\mathcal{DT}_\sim$ est symboliquement déterministe, nécessairement $\mathcal{C}' = \mathcal{C}(x_{n+1})$ et donc $\mathcal{C}(x_{n+1}) \sqsubseteq \mathcal{C}_{n+1}$.
- sinon x_{n+1} est un ancêtre de x_n et la transition $x_n \xrightarrow{g_{n+1},a_{n+1},Y_{n+1}} x_{n+1}$ provient d'une transition $\mathcal{C}(x_n) \xhookrightarrow{g_{n+1},a_{n+1},Y_{n+1}}_{\mathcal{D}} \mathcal{C}(y)$ avec $\mathcal{C}(x_{n+1}) \sqsubseteq \mathcal{C}(y)$. Nous raisonnons comme dans le premier cas pour montrer que $\mathcal{C}(y) \sqsubseteq \mathcal{C}_{n+1}$ et donc $\mathcal{C}(x_{n+1}) \sqsubseteq \mathcal{C}_{n+1}$.

[7] ρ_i correspond à $\mathcal{C}_0 \xhookrightarrow{g_1,a_1,Y_1}_{\mathcal{D}} \mathcal{C}_1 \xhookrightarrow{g_2,a_2,Y_2}_{\mathcal{D}} \cdots \xhookrightarrow{g_i,a_i,Y_i}_{\mathcal{D}} \mathcal{C}_i$.

Il nous reste à montrer que $f(\rho_{n+1}) \in val(\mathsf{Enb}_{\mathcal{DT}_\sim}(\mathcal{C}_{n+1}))$. Par définition de f, $f(\rho_{n+1}) = \mathsf{Enb}_{T_{strat}}(x_{n+1}) \in val(\mathsf{Enb}_{\mathcal{DT}_\sim}(\mathcal{C}(x_{n+1})))$. Or nous avons montré que $\mathcal{C}(x_{n+1}) \sqsubseteq \mathcal{C}_{n+1}$, par la proposition 6.3.10 nous avons que $\mathsf{Enb}_{\mathcal{DT}_\sim}(\mathcal{C}(x_{n+1})) = \mathsf{Enb}_{\mathcal{DT}_\sim}(\mathcal{C}_{n+1})$ donc $f(\rho_{n+1}) \in val(\mathsf{Enb}_{\mathcal{DT}_\sim}(\mathcal{C}_{n+1}))$. □

Le lemme précédent assure que f est une stratégie sûre de $\mathcal{DT}_\sim$. Comme f peut être construite effectivement et est par construction une stratégie à états finis, ceci achève la preuve de la proposition 6.3.13. □

Finalement, par le lemme 6.3.11 et la proposition 6.3.13, le fait que MTL soit close par négation et la proposition 6.3.2, nous obtenons le résultat principal de cette sous-section :

Théorème 6.3.16 *Le problème de contrôle* MTL *sur les mots finis à ressources fixées pour des spécifications souhaitées ou interdites est décidable. De plus s'il existe un contrôleur, il est possible de construire un contrôleur à états finis.*

Remarque 6.3.17 *Comme le problème de satisfaisabilité de* MTL *peut aisément être réduit à un problème de contrôle, le problème de contrôle* MTL *sur les mots finis à ressources fixées a une complexité non primitive-récursive [OW05].*

Remarque 6.3.18 *Comme notre algorithme est basé sur une traduction de* MTL *aux* ATA_1*, le résultat précédent s'étend aux spécifications données directement par des* ATA_1*.*

6.3.5 Contrôle Safety-MTL pour des spécifications souhaitées

Les techniques utilisées pour décider le contrôle Safety-MTL sur des mots infinis pour des spécifications souhaitées sont proches de celles de la sous-section précédente, en effet nous ne pouvons identifier ici des *mauvais* états de $\mathcal{DT}_\sim$, mais nous pouvons identifier des états « dangereux » (que nous appelerons *condamnés*) tel qu'il sera facile de déterminer si un tel état est gagnant ou non.

Nous fixons un jeu temporisé Safety-MTL sur les mots infinis $\mathbb{G} = (\mathcal{A}, val, L^\omega(\varphi))$. Soit Γ l'alphabet symbolique de $\mathcal{A}$, d'après [OW05] nous pouvons construire un ATA_1 complet $\mathcal{B}_\varphi = (\Gamma, P, p_0, \emptyset, \delta)$ sans état accepteur tel que pour tout $\sigma \in T\Sigma^\omega$, $\sigma \not\models \varphi$ si, et seulement s'il existe un préfixe $\overline{\sigma}$ de σ tel que $\overline{\sigma} \in L^*(\mathcal{B}_\varphi)$.

Soit $\mathcal{T}_{\mathcal{A}/\varphi}$ le produit synchronisé de $\mathcal{A}$ et $\mathcal{B}_\varphi$; soient $\mathcal{T}_\sim = (\Gamma, W, w_0, \hookrightarrow)$ et $\mathcal{DT}_\sim = (\Gamma, SW, \{w_0\}, \hookrightarrow_\mathcal{D})$ les systèmes de transitions symboliques définis dans la sous-section 6.3.3.

Un mot $w \in W$ est *condamné* s'il existe $((q,\nu), G) \in H^{-1}(\{w\})$ tel que $G = \emptyset$. Un ensemble de mots $\mathcal{C} \in SW$ est *condamné* s'il contient un mot condamné. Nous utilisons le terme *condamné* car le successeur d'un mot (resp. d'un ensemble de mots) condamné est aussi condamné.

Une stratégie de $\mathcal{DT}_\sim$ est *sûre* s'il n'existe pas de mot $\gamma \in L^\omega_{symb}(\mathcal{A})$ et d'exécution $\mathcal{C}_0 \xhookrightarrow{g_1,a_1,Y_1}_\mathcal{D} \mathcal{C}_1 \xhookrightarrow{g_2,a_2,Y_2}_\mathcal{D} \cdots$ avec $\gamma = (g_1, a_1, Y_1)(g_2, a_2, Y_2)\cdots$ et d'indice $i \in \mathbb{N}$ tel que $\mathcal{C}_i$ est condamné.

Intuitivement une exécution $\mathcal{C}_0 \xhookrightarrow{g_1,a_1,Y_1}_\mathcal{D} \mathcal{C}_1 \xhookrightarrow{g_2,a_2,Y_2}_\mathcal{D} \cdots$ où $\mathcal{C}_i$ est condamné donne lieu à un mot symbolique γ tel que $tw(\gamma) \nsubseteq L^\omega(\varphi)$. Le lemme suivant montre que les stratégies sûres correspondent aux stratégies gagnantes de $\mathbb{G}$.

Lemme 6.3.19 *Il existe une stratégie gagnante (à états finis) du jeu temporisé* Safety-MTL *sur les mots infinis* $\mathbb{G}$ *pour des spécifications souhaitées si, et seulement s'il existe une stratégie sûre (à états finis) de* $\mathcal{DT}_\sim$.

La preuve de ce lemme est analogue à celle du lemme 6.3.11.

Remarque 6.3.20 *La principale différence entre l'algorithme de décision du problème de contrôle* MTL *et* Safety-MTL *pour des spécifications souhaitées est que dans le cas de* MTL*, une configuration* mauvaise *était toujours perdante pour le contrôleur. Ce n'est pas le cas ici avec les ensembles de mots condamnés, en effet comme il y a une condition d'acceptation dans* $\mathcal{A}$*, un ensemble de mots condamnés peut encore être gagnée par le contrôleur s'il arrive à jouer un mot n'appartenant pas à* $L^\omega_{symb}(\mathcal{A})$*. La proposition suivante assure que nous pouvons déterminer si une configuration condamnée est gagnante pour le contrôleur.*

Nous notons $\mathcal{DT}^{\mathcal{C}}_\sim$ le système de transitions $\mathcal{DT}_\sim$ avec pour état initial $\mathcal{C}$.

Proposition 6.3.21 *Pour un ensemble de mots condamné* $\mathcal{C} \in SW$*, l'existence d'une stratégie sûre de* $\mathcal{DT}^{\mathcal{C}}_\sim$ *respectant* val *est décidable. De plus s'il existe une stratégie sûre, une stratégie sûre à états finis est calculable.*

Preuve. Comme $\mathcal{C}$ est condamné, la spécification φ ne joue plus aucun rôle, la seule façon de gagner $\mathcal{DT}^{\mathcal{C}}_\sim$ est de jouer un mot $\gamma \notin L^\omega_{symb}(\mathcal{A})$ (voir la remarque 6.3.20). Il s'agit donc d'un jeu sur un graphe fini classique qui peut-être résolu en le transformant par exemple en jeu de parité [DM02]. ☐

Nous expliquons maintenant comment décider l'existence d'une stratégie sûre de $\mathcal{DT}_\sim$. Notre algorithme est très similaire à celui utilisé dans la sous-section 6.3.4 pour MTL sur les mots finis. Nous commençons avec la racine étiquetée $\{w_0\}$, et à chaque étape nous prenons une feuille x étiquetée $\mathcal{C} \in SW$ et nous faisons l'une des actions suivantes :

- si l'ensemble $\mathcal{C}$ est condamné, nous utilisons la proposition 6.3.21 pour déterminer s'il existe une stratégie sûre à partir de $\mathcal{C}$ respectant val. Si c'est le cas nous déclarons le nœud *bon*, sinon nous déclarons le nœud *mauvais* ; ensuite nous fermons le nœud.
- si l'ensemble $\mathcal{C}$ n'est pas condamné et qu'il existe un ancêtre de x étiqueté $\mathcal{C}'$ avec $\mathcal{C}' \sqsubseteq \mathcal{C}$, nous déclarons le nœud *bon* et nous le fermons.
- sinon, pour toute transition de $\mathcal{DT}_\sim$ de la forme $\mathcal{C} \xhookrightarrow{g,a,Y}_{\mathcal{D}} \mathcal{C}'$, nous ajoutons un nouveau nœud y étiqueté $\mathcal{C}'$ et une arête de x à y étiquetée (g, a, Y). Si $\mathcal{C}$ n'a pas de successeur, nous déclarons x *mort* et le fermons.

Tout comme pour MTL sur le mots finis, la terminaison de cette construction est assurée par le lemme de König et le fait que $(SW, \sqsubseteq)$ est un bel-ordre partiel. Nous affectons une valeur dans $\{\top, \bot\}$ à chaque nœud de l'arbre comme dans la sous-section 6.3.4. L'algorithme répond « oui » si, et seulement si la valeur de la racine est $\top$.

La correction de l'algorithme est assurée par la proposition suivante :

Proposition 6.3.22 *Si l'algorithme répond « non », alors il n'existe pas de stratégie sûre de* $\mathcal{DT}_\sim$.
Si l'algorithme répond « oui », alors il existe une stratégie sûre de $\mathcal{DT}_\sim$ *à états finis, et celle-ci peut être construite effectivement.*

La preuve de cette proposition est similaire à celle de la proposition 6.3.13.

Nous avons donc montré le théorème suivant :

Théorème 6.3.23 *Le problème de contrôle* Safety-MTL *sur les mots infinis à ressources fixées pour des spécifications souhaitées est décidable. De plus, s'il existe un contrôleur, il est possible de construire un contrôleur à états finis.*

Remarque 6.3.24 *Le problème de* model-checking *de* Safety-MTL *sur les mots infinis a une complexité non primitive-récursive : c'est une conséquence de [OW05], comme cela nous a été communiqué par les auteurs de cet article. Or ce problème peut facilement être réduit au problème de contrôle* Safety-MTL *sur les mots infinis pour des spécifications souhaitées. La complexité du problème de contrôle* Safety-MTL *sur les mots infinis pour des spécifications souhaitées est donc non primitive-récursive.*

6.3.6 Contrôle Safety-MTL pour des spécifications interdites

Les preuves de décidabilité du contrôle MTL et Safety-MTL pour des spécifications souhaitées sont assez similaires. Pour le contrôle Safety-MTL pour des spécifications interdites nous aurons recours à une technique plus élaborée : nous définirons aussi une notion de stratégie *sûre* mais nous ne pourrons plus déplier partiellement l'arbre d'exécution de $\mathcal{DT}_{\sim}$. Nous caractériserons l'ensemble des configurations gagnantes par un point fixe et montrerons que celui-ci a une base finie calculable. Cette technique a été utilisée dans [OW06b] pour établir la décidabilité de la satisfaisabilité de Safety-MTL.

Nous rappelons tout d'abord quelques résultats de la théorie des beaux-ordres partiels. Étant donné un ordre partiel $(S, \preceq)$, nous disons que $L \subseteq S$ est *fermé vers le bas* si $x \in S$, $y \in L$, et $x \preceq y$ impliquent $x \in L$. Nous disons que $L \subseteq S$ est *fermé vers le haut* si $x \in S$, $y \in L$, et $y \preceq x$ impliquent $x \in L$. La *fermeture vers le haut* de $S_1 \subseteq S$, notée $\uparrow S_1$ est l'ensemble $\{x \in S \mid \exists y \in S_1 : \ y \preceq x\}$. Une *base* d'un ensemble fermé vers le haut U est un sous-ensemble U_b de U tel que $U = \uparrow U_b$. Une *co-base* d'un ensemble fermé vers le bas L est une base de l'ensemble fermé vers le haut $S \setminus L$.

Proposition 6.3.25 ([FS01]) *Soit $(S, \preceq)$ un bel-ordre partiel. Alors*

1. *tout ensemble fermé vers le bas $L \subseteq S$ a une co-base finie.*
2. *toute suite infinie décroissante $L_0 \supseteq L_1 \supseteq L_2 \supseteq \cdots$ d'ensembles fermés vers le bas se stabilise, c'est-à -dire qu'il existe $k \in \mathbb{N}$ tel que $L_n = L_k$ pour tout $n \geq k$.*

Nous fixons un jeu temporisé Safety-MTL sur les mots infinis $\mathbb{G} = (\mathcal{A}, val, L^{\omega}(\varphi))$ avec $\mathcal{A} = (\Sigma, X, Q, q_0, R^{\mathcal{A}}, \delta^{\mathcal{A}})$. Soit Γ l'alphabet symbolique de $\mathcal{A}$, d'après [OW05] nous pouvons construire un ATA_1 complet et local[8] $\mathcal{B}_{\varphi} = (\Gamma, P, p_0, P \backslash \{p\}, \delta)$ où[9] p est un état puit (c'est-à-dire que pour tout $a \in \Sigma$, $\delta(p, a) = p$). En particulier pour tout mot $\sigma \in T\Sigma^{\omega}$, $\sigma \models \varphi$ si, et seulement s'il existe une exécution infinie de $\mathcal{B}_{\varphi}$ ne visitant que des configurations qui ne contiennent pas p.

[8] Un ATA_1 est local si son horloge est remise à zéro lorsqu'il change d'état.

[9] Cet automate correspond intuitivement à un automate acceptant le complémentaire du langage de l'ATA_1 $\mathcal{B}_{\varphi}$ que nous avons considéré dans la sous-section précédente car il accepte les modèles vérifiant φ. Nous le notons également $\mathcal{B}_{\varphi}$ car il joue le même rôle que précédemment.

Soit $\mathcal{T}_{\mathcal{A}/\varphi}$ le produit synchronisé de $\mathcal{A}$ et $\mathcal{B}_\varphi$; soient $\mathcal{T}_\sim = (\Gamma, W, w_0, \hookrightarrow)$ et $\mathcal{DT}_\sim = (\Gamma, SW, \{w_0\}, \hookrightarrow_\mathcal{D})$ les systèmes de transitions symboliques définis dans la sous-section 6.3.3. Un mot $w \in W$ est *sauvé* s'il existe $((q, \nu), G) \in H^{-1}(\{w\})$ tel que $(p, u) \in G$ pour $u \in \mathbb{T}$. Un ensemble de mots $\mathcal{C} \in SW$ est dit *sauvé* si pour tout $w \in \mathcal{C}$, w est sauvé. Nous utilisons le terme *sauvé* car le successeur d'un mot (resp. d'un ensemble de mots) sauvé est aussi sauvé. Intuitivement, si $\mathcal{C}_i$ est sauvé, le mot ne pourra être accepté par $\mathcal{B}_\varphi$ et il est donc déjà assuré que la spécification interdite sera satisfaite.

Un ensemble de mots $\mathcal{C} \in SW$ est dit *répété* s'il existe $w \in \mathcal{C}$ et $(q, y, r) \in Q \times X \times REG_K$ tel que $q \in R^\mathcal{A}$ et $\{(q, y, r)\} \preceq w$. Intuitivement $\mathcal{C}$ si l'état de $\mathcal{A}$ associé est répété.

Une stratégie f de $\mathcal{DT}_\sim$ est dite *sûre* si pour tout $\gamma \in \Gamma^\omega$ compatible avec f, l'exécution de γ sur $\mathcal{DT}_\sim$, $\rho = \mathcal{C}_0 \xhookrightarrow{g_1, a_1, Y_1}_\mathcal{D} \mathcal{C}_1 \xhookrightarrow{g_2, a_2, Y_2}_\mathcal{D} \cdots$, vérifie : $\{i \mid \mathcal{C}_i \text{ est répété}\}$ est fini ou il existe $i \in \mathbb{N}$ tel que $\mathcal{C}_i$ est sauvé. Le lemme suivant montre que les stratégies sûres correspondent aux stratégies gagnantes de $\mathbb{G}$.

Lemme 6.3.26 *Il existe une stratégie gagnante (à états finis) du jeu temporisé* Safety-MTL *sur les mots infinis $\mathbb{G}$ pour des spécifications interdites si, et seulement s'il existe une stratégie sûre (à états finis) de $\mathcal{DT}_\sim$.*

La preuve de ce lemme est analogue à celle du lemme 6.3.11.

Remarque 6.3.27 *Dans les sous-sections précédentes, nous avons utilisé la construction d'un arbre fini correspondant à une partie du dépliage de $\mathcal{DT}_\sim$ pour résoudre les problèmes de contrôle* MTL *sur les mots finis et* Safety-MTL *sur les mots infinis pour des spécifications souhaitées. Cette technique ne peut être appliquée pour le contrôle* Safety-MTL *pour des spécifications interdites ; en effet, dans les deux cas précédents, le but du contrôleur était d'éviter les états mauvais ou condamnés[10] ; si dans la construction de l'arbre nous rencontrions une configuration $\mathcal{C}_1$ puis une configuration $\mathcal{C}_2$, avec $\mathcal{C}_1 \sqsubseteq \mathcal{C}_2$, nous pouvions arrêter le dépliage à partir de $\mathcal{C}_2$, car la configuration $\mathcal{C}_2$ pouvait être considérée moins dangereuse que la configuration $\mathcal{C}_1$. Dans le cadre du contrôle* Safety-MTL *pour des spécifications interdites ce n'est pas le cas, en effet le but du contrôleur est maintenant d'atteindre une configuration sauvée, donc si nous recontrons une configuration $\mathcal{C}_1$ puis une configuration $\mathcal{C}_2$, avec $\mathcal{C}_1 \sqsubseteq \mathcal{C}_2$, $\mathcal{C}_2$ ne peut être considérée moins dangereuse pour le contrôleur, car il peut exister une suite de configurations $\mathcal{C}_1 \sqsubseteq \mathcal{C}_2 \sqsubseteq \mathcal{C}_3 \sqsubseteq \cdots$ où aucune $\mathcal{C}_i$ n'est sauvée, et qui fait perdre le contrôleur.*

La configuration $\mathcal{C}_2$ ne peut pas non plus être considérée comme plus dangereuse que la configuration $\mathcal{C}_1$ pour le contrôleur, car si $\mathcal{C}_1 \sqsubseteq \mathcal{C}_2$, la configuration $\mathcal{C}_2$ « a plus de chances » de mener à une configuration sauvée que $\mathcal{C}_1$.

Les techniques développées précédemment ne peuvent donc s'appliquer, c'est pourquoi nous avons recours à des méthodes de point fixe.

Nous notons Ω l'ensemble des ensembles de mots $\mathcal{C} \in SW$ tel qu'il n'existe *pas* de stratégie sûre de $\mathcal{DT}^\mathcal{C}_\sim$. Il est évident qu'il existe une stratégie sûre de $\mathcal{DT}_\sim$ si, et seulement si $\{w_0\} \notin \Omega$. Nous montrons maintenant que nous pouvons construire une représentation finie de Ω.

Définition 6.3.28 *Pour $L \subseteq SW$, nous définissons $\mathcal{P}_{i+}(L)$ comme l'ensemble des $\mathcal{C} \in SW$ tels que $\mathcal{C}$ n'est pas sauvé et pour toute stratégie f de $\mathcal{DT}^\mathcal{C}_\sim$, il existe un mot $\gamma \in \Gamma^+$ compatible avec f et $\mathcal{C}' \in SW$ tels que $\mathcal{C} \xhookrightarrow{\gamma}{}^*_\mathcal{D} \mathcal{C}'$, $\mathcal{C}'$ est répété et $\mathcal{C}' \in L$.*

[10]Pour plus de clarté, nous négligeons la condition d'acceptation de $\mathcal{A}$ dans cette discussion.

Intuitivement $\mathcal{P}_{i+}(L)$ correspond à l'ensemble des configurations non sauvées à partir desquelles le contrôleur ne peut empêcher l'environnement d'atteindre L en étant passé par un état répété. Ω est donc intuitivement le plus grand point fixe de $\mathcal{P}_{i+}$ car $\mathcal{C} \in \Omega$ si l'environnement peut rester dans des configurations non sauvées et passer infiniment souvent par des états répétés de $\mathcal{A}$, formant ainsi un mot violant la spécification interdite. Nous montrons ce résultat formellement dans le lemme suivant :

Lemme 6.3.29 *Ω est le plus grand point fixe de $\mathcal{P}_{i+}(-) : 2^{SW} \to 2^{SW}$.*

Preuve. Nous montrons tout d'abord que tout point fixe de $\mathcal{P}_{i+}(L)$ est inclus dans Ω. Soit $L \subseteq SW$ tel que $\mathcal{P}_{i+}(L) = L$, et soit $\mathcal{C}_0 \in L$. Nous montrons que $\mathcal{C}_0 \in \Omega$, c'est-à-dire qu'il n'existe pas de stratégie sûre de $\mathcal{DT}^{\mathcal{C}_0}_{\sim}$. Soit donc f_0 une stratégie de $\mathcal{DT}^{\mathcal{C}_0}_{\sim}$. Comme $\mathcal{C}_0 \in \mathcal{P}_{i+}(L)$, il existe un mot $\gamma_1 \in \Gamma^+$ compatible avec f_0 et $\mathcal{C}_1 \in SW$ tels que $\mathcal{C}_0 \xhookrightarrow{\gamma_1}{}^{*}_{\mathcal{D}} \mathcal{C}_1$, $\mathcal{C}_1$ est répété et $\mathcal{C}_1 \in L$. Soit f_1 la stratégie de $\mathcal{DT}^{\mathcal{C}_1}_{\sim}$ correspondant au jeu de f_0 après γ_1. Comme $\mathcal{C}_1 \in L$, nous pouvons appliquer le même raisonnement à f_1. Nous construisons ainsi un mot infini $\gamma = \gamma_1\gamma_2\cdots$ compatible avec f_0 tel que $\mathcal{C}_0 \xhookrightarrow{\gamma_1}_{\mathcal{D}} \mathcal{C}_1 \xhookrightarrow{\gamma_2}_{\mathcal{D}} \cdots$ avec pour tout i, $\mathcal{C}_i \in L$ et est répété. Le chemin de $\mathcal{DT}^{\mathcal{C}_0}_{\sim}$ sur γ ne visite que des configurations non sauvées et passe infiniment souvent par des configurations répétées. La stratégie f_0 n'est donc pas sûre, ce qui montre que $\mathcal{C}_0 \in \Omega$. Nous avons donc montré que $L \subseteq \Omega$.

Nous montrons maintenant que Ω est un point fixe de $\mathcal{P}_{i+}$.

Montrons tout d'abord que $\mathcal{P}_{i+}(\Omega) \subseteq \Omega$.

Soit $\mathcal{C} \in \mathcal{P}_{i+}(\Omega)$ et soit f une stratégie de $\mathcal{DT}^{\mathcal{C}}_{\sim}$; montrons que f n'est pas sûre. Comme $\mathcal{C} \in \mathcal{P}_{i+}(\Omega)$, il existe un mot $\gamma_1 \in \Gamma^+$ compatible avec f et $\mathcal{C}_1 \in SW$ tels que $\mathcal{C} \xhookrightarrow{\gamma_1}{}^{*}_{\mathcal{D}} \mathcal{C}_1$, $\mathcal{C}_1$ est répété et $\mathcal{C}_1 \in \Omega$. Soit f' la stratégie de $\mathcal{DT}^{\mathcal{C}_1}_{\sim}$ correspondant au jeu de f après γ_1. Comme $\mathcal{C}_1 \in \Omega$, f' n'est pas une stratégie sûre de $\mathcal{DT}^{\mathcal{C}_1}_{\sim}$. Il existe donc $\gamma \in \Gamma^\omega$ compatible avec f' tel que le chemin de $\mathcal{DT}^{\mathcal{C}_1}_{\sim}$ sur γ ne visite donc que des configurations non sauvées et passe infiniment souvent par des configurations répétées. Le mot $\gamma_0\gamma$, compatible avec f, montre alors que f n'est pas une stratégie sûre de $\mathcal{DT}^{\mathcal{C}}_{\sim}$ et donc que $\mathcal{C} \in \Omega$.

Montrons maintenant que $\Omega \subseteq \mathcal{P}_{i+}(\Omega)$, nous montrons en réalité que $\mathcal{P}_{i+}(\Omega)^c \subseteq \Omega^c$. Soit $\mathcal{C} \notin \mathcal{P}_{i+}(\Omega)$ et montrons que $\mathcal{C} \notin \Omega$. Si $\mathcal{C}$ est sauvé, il est évident que $\mathcal{C} \notin \Omega$. Sinon il existe une stratégie f de $\mathcal{DT}^{\mathcal{C}}_{\sim}$ telle que pour tout mot $\gamma \in \Gamma^+$ compatible avec f et $\mathcal{C}'$ tel que $\mathcal{C} \xhookrightarrow{\gamma}{}^{*}_{\mathcal{D}} \mathcal{C}'$, soit $\mathcal{C}'$ n'est pas répété, soit $\mathcal{C}' \notin \Omega$.

Pour montrer que $\mathcal{C} \notin \Omega$ nous cherchons à montrer qu'il existe une stratégie sûre de $\mathcal{DT}^{\mathcal{C}}_{\sim}$. La stratégie f n'est pas nécessairement sûre mais nous construisons à partir de f une stratégie f' sûre de la façon suivante :

- tant qu'aucun élément de Ω^c n'a été visité, f' est identique à f,
- si un élément $\mathcal{C}' \in \Omega^c$ est atteint, par définition de Ω il existe une stratégie sûre $f_{\mathcal{C}'}$ de $\mathcal{DT}^{\mathcal{C}'}_{\sim}$. f' est alors identique à $f_{\mathcal{C}'}$.

Montrons que f' est une stratégie sûre de $\mathcal{DT}^{\mathcal{C}}_{\sim}$. Soit $\gamma \in \Gamma^\omega$ un mot compatible avec f'. Soit $\mathcal{C}_0 \xhookrightarrow{\gamma_1}_{\mathcal{D}} \mathcal{C}_1 \xhookrightarrow{\gamma_2}_{\mathcal{D}} \cdots$ l'exécution de $\mathcal{DT}^{\mathcal{C}}_{\sim}$ sur γ (nous avons que $\mathcal{C}_0 = \mathcal{C}$). Soit pour tout i, $\mathcal{C}_i$ n'est pas répété alors il n'y a rien à vérifier. Soit il existe $i \in \mathbb{N}$ tel que $\mathcal{C}_i$ est répété ; soit alors $i_0 \in \mathbb{N}$ le plus petit $j \in \mathbb{N}$ tel que $\mathcal{C}_j$ est répété. Alors $\mathcal{C}_0 \xhookrightarrow{\gamma_1}_{\mathcal{D}} \mathcal{C}_1 \xhookrightarrow{\gamma_2}_{\mathcal{D}} \cdots \xhookrightarrow{\gamma_{i_0}}_{\mathcal{D}} \mathcal{C}_{i_0}$ est compatible avec f, donc par définition de f, $\mathcal{C}_{i_0} \notin \Omega$. Alors par définition de f', il existe $i_1 \leq i_0$ tel que $\mathcal{C}_{i_1} \xhookrightarrow{\gamma_{i_1+1}}_{\mathcal{D}} \mathcal{C}_{i_1+1} \xhookrightarrow{\gamma_{i_1+2}}_{\mathcal{D}} \cdots$ est compatible avec $f_{\mathcal{C}_{i_1}}$ (i_1 est le plus petit indice tel que $\mathcal{C}_{i_1} \notin \Omega$). Il est alors facile de déduire que γ vérifie que $\{i \mid \mathcal{C}_i$ est répété$\}$ est fini ou il existe $i \in \mathbb{N}$ tel que $\mathcal{C}_i$ est sauvé car nous avons montré que cette condition est satisfaite par $\gamma_{i_1}\gamma_{i_1+1}\cdots$ et qu'elle ne dépend pas d'un préfixe fini.

f' est donc une stratégie sûre de $\mathcal{DT}^{\mathcal{C}}_{\sim}$ et nous avons montré que $\Omega \subseteq \mathcal{P}_{i+}(\Omega)$.

Cela achève la preuve que Ω est le plus grand point fixe de $\mathcal{P}_{i+}$. □

Notre but est maintenant de calculer une représentation finie de Ω ; l'idée est de calculer la suite d'approximations $SW \supseteq \mathcal{P}_{i+}(SW) \supseteq \mathcal{P}_{i+}^2(SW) \supseteq \cdots$ et d'utiliser la proposition 6.3.25 pour montrer que cette suite se stabilise.

Nous montrons tout d'abord que $\mathcal{P}_{i+}$ transforme un ensemble fermé vers le bas en un ensemble fermé vers le bas :

Lemme 6.3.30 *Soit $L \subseteq SW$ un ensemble fermé vers le bas (par rapport à $\sqsubseteq$). Alors $\mathcal{P}_{i+}(L)$ est un ensemble fermé vers le bas.*

Preuve. Soit $L \subseteq SW$ un ensemble fermé vers le bas. Soit $\mathcal{C}' \in \mathcal{P}_{i+}(L)$ et soit $\mathcal{C} \sqsubseteq \mathcal{C}'$. Montrons que $\mathcal{C} \in \mathcal{P}_{i+}(L)$.

$\mathcal{C}$ n'est pas sauvé car $\mathcal{C}'$ n'est pas sauvé et $\mathcal{C} \sqsubseteq \mathcal{C}'$.

Soit f une stratégie de $\mathcal{DT}^{\mathcal{C}}_{\sim}$. Par la proposition 6.3.10, f est une stratégie de $\mathcal{DT}^{\mathcal{C}'}_{\sim}$. Comme $\mathcal{C}' \in \mathcal{P}_{i+}(L)$ il existe un mot $\gamma \in \Gamma^+$ compatible avec f et $\mathcal{C}'_1 \in SW$ tels que $\mathcal{C}' \xhookrightarrow{\gamma}{}^*_{\mathcal{D}} \mathcal{C}'_1$, $\mathcal{C}'_1$ est répété et $\mathcal{C}'_1 \in L$. Par la proposition 6.3.10 il existe $\mathcal{C}_1 \sqsubseteq \mathcal{C}'_1$ tel que $\mathcal{C} \xhookrightarrow{\gamma}{}^*_{\mathcal{D}} \mathcal{C}_1$. Le fait d'être répété ne dépend que de la $\mathcal{A}$-composante donc $\mathcal{C}_1$ est répété, et comme L est fermé vers le bas, $\mathcal{C}_1 \in L$. Nous avons donc montré que $\mathcal{C} \in \mathcal{P}_{i+}(L)$. $\mathcal{P}_{i+}(L)$ est donc fermé vers le bas. □

Nous avons montré que $\mathcal{P}_{i+}$ transforme des ensembles fermés vers le bas en des ensembles fermés vers le bas. Pour avoir une procédure effective, il nous faut montrer que nous pouvons calculer une représentation finie de $\mathcal{P}_{i+}(L)$ à partir d'une représentation finie de L. C'est le but des prochains lemmes et propositions. Nous raisonnons sur une approximation finie $\preceq_n$ de $\preceq$ qui ne considère que les mots de longueur inférieure ou égale à n.

Nous supposons que nous connaissons une co-base finie B de l'ensemble fermé vers le bas L. Nous montrons que nous pouvons calculer une co-base finie de $\mathcal{P}_{i+}(L)$.

Définition 6.3.31 ([OW06b]) *Soit $n \geq 1$, nous notons $\preceq_n$ l'ordre partiel sur W défini par : $w \preceq_n w'$ si, et seulement si pour tout $u \in W$ tel que $|u| \leq n$, $u \preceq w$ implique $u \preceq w'$.*

Remarquons que $w \preceq w'$ implique $w \preceq_n w'$ pour tout $n \geq 1$. De plus, pour tout $n \geq |w|$, $w \preceq_n w'$ implique $w \preceq w'$.

Exemple 6.3.32 *Considérons $\Sigma = \{a, b\}$ et $\preceq$ la relation de sous-mot sur Σ^* . Nous avons que $aba \preceq_2 bab$: en effet l'ensemble des mots u de longueur inférieure ou égale à 2 tels que $u \preceq aba$ est $\{\varepsilon, a, b, ab, ba\}$ et chacun de ces mots u vérifie $u \preceq bab$. Nous avons donc que $aba \preceq_2 bab$ mais $aba \not\preceq bab$.*

Nous considérons également l'ordre partiel $\sqsubseteq_n$ sur SW défini par : $\mathcal{C} \sqsubseteq_n \mathcal{C}'$ si, et seulement si $\forall w' \in \mathcal{C}'\ \exists w \in \mathcal{C},\ w \preceq_n w'$. Pour $\mathcal{C} \in SW$, la *longueur* de $\mathcal{C}$, notée $|\mathcal{C}|$, est la longueur du plus long mot de $\mathcal{C}$.

Lemme 6.3.33 *Soit $n \in \mathbb{N}$ un entier supérieur au nombre d'horloges de $\mathcal{A}$. La relation de transition $\hookrightarrow$ de $\mathcal{T}_\sim$ est compatible vers le bas par rapport à $\preceq_n$; c'est-à-dire $u \preceq_n w$ et $w \xhookrightarrow{g,a,Y} w'$ impliquent qu'il existe $u' \preceq_n w'$ tel que $u \xhookrightarrow{g,a,Y} u'$.*

La preuve du lemme 6.3.33 est identique à celle du lemme de simulation du rapport de recherche de [OW06b].

Lemme 6.3.34 *Soit $n \in \mathbb{N}$ un entier supérieur au nombre d'horloges de $\mathcal{A}$.*

1. *si $\mathcal{C} \sqsubseteq_n \mathcal{C}'$, alors $\mathsf{Enb}_{\mathcal{DT}_\sim}(\mathcal{C}) = \mathsf{Enb}_{\mathcal{DT}_\sim}(\mathcal{C}')$.*
2. *La relation de transition $\hookrightarrow_\mathcal{D}$ de $\mathcal{DT}_\sim$ est compatible vers le bas par rapport à $\sqsubseteq_n$, c'est-à-dire $\mathcal{C}'_1 \sqsubseteq_n \mathcal{C}_1$ et $\mathcal{C}_1 \xhookrightarrow{g,a,Y}_\mathcal{D} \mathcal{C}_2$ impliquent qu'il existe $\mathcal{C}'_2 \sqsubseteq_n \mathcal{C}_2$ tel que $\mathcal{C}'_1 \xhookrightarrow{g,a,Y}_\mathcal{D} \mathcal{C}'_2$.*

En utilisant le lemme 6.3.33, la preuve du lemme 6.3.34 est identique à celle de la proposition 6.3.10.

Dans la suite nous supposons que n est un entier supérieur au nombre d'horloges de $\mathcal{A}$ et à la longueur de tout $\mathcal{C} \in B$. Remarquons que L est un ensemble fermé vers le bas par rapport à $\sqsubseteq_n$. En effet pour tout $\mathcal{C} \in SW$ et $\mathcal{C}' \in B$, $\mathcal{C}' \sqsubseteq_n \mathcal{C}$ si, et seulement si $\mathcal{C}' \sqsubseteq \mathcal{C}$.

Proposition 6.3.35 *$\mathcal{P}_{i+}(L)$ est un ensemble fermé vers le bas par rapport à $\sqsubseteq_n$.*

Par le lemme 6.3.33, la relation de transition de $\mathcal{T}_\sim$ est compatible vers le bas par rapport à $\preceq_n$. La preuve de la proposition 6.3.35 est alors identique à celle du lemme 6.3.30.

Proposition 6.3.36 *L'ensemble des $\mathcal{C} \in SW \setminus \mathcal{P}_{i+}(L)$ de longueur inférieure ou égale à $n \cdot 2^{n\cdot|\Lambda|}$ est[11] une base de $SW \setminus \mathcal{P}_{i+}(L)$.*

Preuve. Soit $\mathcal{C} \in SW \setminus \mathcal{P}_{i+}(L)$. Montrons qu'il existe $\mathcal{C}_1 \in SW \setminus \mathcal{P}_{i+}(L)$ tel que $\mathcal{C}_1 \sqsubseteq \mathcal{C}$ et $|\mathcal{C}_1| \leq n \cdot 2^{n\cdot|\Lambda|}$. Par la proposition 6.3.35, $\mathcal{C} \sqsubseteq_n \mathcal{C}_1$ implique $\mathcal{C}_1 \in SW \setminus \mathcal{P}_{i+}(L)$. Soit u un mot de $\mathcal{C}$. Suivant la technique utilisée dans [OW06b], pour tout sous-mot $u' \preceq u$ avec $|u'| = n$, nous marquons les lettres de u correspondant à u' . Il y a n telles lettres pour tout sous-mot, et il y a moins de $2^{n\cdot|\Lambda|}$ tels sous-mots. Soit $sub(u)$ le mot composé de toutes les lettres marquées de u (dans l'ordre). Nous avons que $sub(u) \preceq u$ et $u \preceq_n sub(u)$. Soit $\mathcal{C}_1 \in SW$ défini par : $\mathcal{C}_1 = \{sub(u) \mid u \in \mathcal{C}\}$. Nous avons que $\mathcal{C}_1 \sqsubseteq \mathcal{C}$, $|\mathcal{C}_1| \leq n \cdot 2^{n\cdot|\Lambda|}$, et $\mathcal{C} \sqsubseteq_n \mathcal{C}_1$. Comme $\mathcal{P}_{i+}(L)$ est un ensemble fermé vers le bas par rapport à $\sqsubseteq_n$, nécessairement $\mathcal{C}_1 \in SW \setminus \mathcal{P}_{i+}(L)$.

$\{\mathcal{C}_1 \in SW \setminus \mathcal{P}_{i+}(L) \mid |\mathcal{C}_1| \leq n \cdot 2^{n\cdot|\Lambda|}\}$ est donc une base de $SW \setminus \mathcal{P}_{i+}(L)$. □

Proposition 6.3.37 *Soit $\mathcal{C} \in SW$, il est décidable de tester si $\mathcal{C} \in \mathcal{P}_{i+}(L)$.*

[11] Nous rappelons que $\Lambda = 2^{(Q\times X\times REG_K)\cup(P\times REG_K)}$, tel que défini page 101, est l'alphabet sur lequel sont définies les configurations.

Preuve. Nous utilisons un algorithme similaire à celui de la sous-section 6.3.4. Nous déclarons *mauvaises* les configurations $\mathcal{C}' \in SW$ telles que $\mathcal{C}'$ est répété et $\mathcal{C}' \in L$. Nous construisons ensuite un arbre fini qui est une partie du dépliage de $\mathcal{DT}_\sim$ à partir de $\mathcal{C}$. La construction et la preuve de correction sont similaires à celles de la sous-section 6.3.4. □

Corollaire 6.3.38 *Une co-base finie de $\mathcal{P}_{i+}(L)$ peut être calculée à partir d'une co-base finie de L.*

Preuve. Par définition, une co-base finie de $\mathcal{P}_{i+}(L)$ est une base finie de $SW \setminus \mathcal{P}_{i+}(L)$. Soit B une co-base finie de L et soit n un entier supérieur au nombre d'horloges de $\mathcal{A}$ et à la longueur de tout $\mathcal{C} \in B$.

Par la proposition 6.3.36, l'ensemble des $\mathcal{C} \in SW$ de longueur inférieure ou égale à $n \cdot 2^{n \cdot |\Lambda|}$ tels que $\mathcal{C} \notin \mathcal{P}_{i+}(L)$ est une base de $SW \setminus \mathcal{P}_{i+}(L)$. Comme l'ensemble des $\mathcal{C} \in SW$ de longueur inférieur ou égale à $n \cdot 2^{n \cdot |\Lambda|}$ est fini, il suffit de tester pour chaque tel $\mathcal{C}$ si $\mathcal{C} \in \mathcal{P}_{i+}(L)$, ce qui est possible par la proposition 6.3.37. □

Nous pouvons maintenant achever la preuve de décidabilité des jeux temporisés Safety-MTL pour des spécifications interdites. Nous calculons des co-bases finies de la suite d'approximation $SW \supseteq \mathcal{P}_{i+}(SW) \supseteq \mathcal{P}_{i+}^2(SW) \supseteq \cdots$ et montrons que cette suite se stabilise sur Ω. Il suffit alors de tester si la configuration initiale est dans Ω.

Proposition 6.3.39 *Étant donné un jeu temporisé* Safety-MTL $\mathbb{G} = (\mathcal{A}, val, L^\omega(\varphi))$, *il est possible de décider l'existence d'une stratégie gagnante pour des spécifications interdites.*

Preuve. Par le lemme 6.3.26 et la définition de Ω, il existe une stratégie gagnante de $\mathbb{G}$ pour des spécifications interdites si, et seulement si $\{w_0\} \notin \Omega$. Par le lemme 6.3.30, l'opérateur $\mathcal{P}_{i+}$ transforme un ensemble fermé vers le bas en ensemble fermé vers le bas. La suite $SW \supseteq \mathcal{P}_{i+}(SW) \supseteq \mathcal{P}_{i+}^2(SW) \supseteq \cdots$ est donc une suite d'ensembles fermés vers le bas. Par le corollaire 6.3.38, nous pouvons construire une co-base finie de chaque $\mathcal{P}_{i+}^n(SW)$. De plus, par la proposition 6.3.25 cette suite se stabilise après un nombre fini d'itérations (le plus petit k tel que $\mathcal{P}_{i+}^k(SW) = \mathcal{P}_{i+}^{k+1}(SW)$). La valeur de stabilisation est le plus grand point fixe de $\mathcal{P}_{i+}$, c'est donc Ω par le lemme 6.3.29. Nous pouvons calculer une co-base finie de Ω, et donc décider si $\{w_0\} \in \Omega$. □

Le théorème suivant est une conséquence immédiate des propositions 6.3.2 et 6.3.39 :

Théorème 6.3.40 *Le problème de contrôle* Safety-MTL *sur les mots infinis à ressources fixées pour des spécifications interdites est décidable.*

Remarque 6.3.41 *Le problème de non-satisfaisabilité de* Safety-MTL *sur les mots infinis, qui a une complexité non-élémentaire [BMO⁺07], peut facilement être réduit au problème de contrôle* Safety-MTL *sur les mots infinis pour des spécifications interdites. La complexité du problème de contrôle* Safety-MTL *sur les mots infinis pour des spécifications interdites est donc non-élémentaire.*

Remarque 6.3.42 *Pour le problème de contrôle* Safety-MTL *pour des spécifications interdites, le problème de savoir s'il existe toujours un contrôleur à états finis est ouvert à notre connaissance.*

Troisième partie

Expressivité de MTL

Chapitre 7

Expressivité de MTL et TPTL

Les logiques temporisées MTL et TPTL ont été introduites à la fin des années 80 pour exprimer des propriétés quantitatives des systèmes temporisés [AH89, Koy90]. Dans ce chapitre, nous nous intéressons à l'expressivité relative de ces logiques. Il est évident que toute formule de MTL peut être exprimée dans TPTL car la formule de MTL $\varphi_1 \, \mathbf{U}_I \, \varphi_2$ est équivalente à la formule de TPTL $x.(\varphi_1 \, \mathbf{U} \, (\varphi_2 \wedge x \in I))$. Dans [AH92b, AH93, Hen98], Alur et Henzinger ont conjecturé que TPTL serait strictement plus expressive que MTL. Ils ont aussi conjecturé que la formule de TPTL suivante ne pourrait pas être exprimée dans MTL :

$$\mathbf{G}\,(a \Rightarrow x.\mathbf{F}\,(b \wedge \mathbf{F}\,(c \wedge \ x \leq 2))) \tag{7.1}$$

Nous résolvons cette conjecture pour les deux sémantiques, la sémantique d'actions et la sémantique continue. Nous montrons que pour la sémantique d'actions, la formule (7.1) ne peut pas être exprimée dans MTL, établissant le saut d'expressivité conjecturé. De façon surprenante, nous montrons que la formule (7.1) peut être exprimée dans MTL pour la sémantique continue ; nous proposons alors une autre formule de TPTL ne pouvant pas être exprimée dans MTL pour la sémantique continue, confirmant la conjecture :

$$x.\mathbf{F}\,(a \wedge x \leq 1 \wedge \mathbf{G}\,(x \leq 1 \Rightarrow \neg b))$$

Nos techniques permettent de déduire facilement d'autres propriétés intéressantes comme le fait qu'ajouter des modalités du passé à MTL et MITL augmente strictement leur expressivité, ou le fait que MTL est strictement plus expressive pour la sémantique continue que pour la sémantique d'actions (cela a été prouvé indépendamment dans [DP07]). Nous montrons également que la formule (7.1) ne peut être exprimée dans MITL pour la sémantique continue, ce qui est contre-intuitif car cette formule n'utilise pas de contrainte ponctuelle.

7.1 TPTL est strictement plus expressive que MTL

Dans cette section, nous montrons que TPTL est strictement plus expressive que MTL pour la sémantique d'actions et la sémantique continue.

7.1.1 La formule d'Alur et Henzinger n'est pas un bon témoin

Il a été conjecturé dans [AH92b, AH93, Hen98] que TPTL serait strictement plus expressive que MTL, et en particulier que la formule de TPTL

$$\mathbf{G}\,(a \Rightarrow x.\mathbf{F}\,(b \wedge \mathbf{F}\,(c \wedge\ x \leq 2)))$$

ne peut pas être exprimée dans MTL. La proposition suivante montre que cette formule n'est pas un bon témoin pour montrer que TPTL est strictement plus expressive que MTL pour la sémantique continue :

Proposition 7.1.1 *La formule de* TPTL $x.\mathbf{F}\,(b \wedge \mathbf{F}\,(c \wedge\ x \leq 2))$ *peut être exprimée dans* MTL *pour la sémantique continue.*

Preuve. Soit Φ la formule de TPTL $x.\mathbf{F}\,(b \wedge \mathbf{F}\,(c \wedge\ x \leq 2))$. Cette formule exprime que depuis le point courant, il y a un b suivi d'un c dans la séquence temporisée, et que le c se produit dans moins de deux unités de temps. Pour prouver la proposition, nous construisons une formule de MTL Φ' qui est équivalente à Φ pour les séquences temporisées. La formule Φ' est définie comme la disjonction de trois formules $\Phi' = \Phi'_1 \vee \Phi'_2 \vee \Phi'_3$ où :

$$\begin{cases} \Phi'_1 & = (\mathbf{F}_{\leq 1}\, b) \wedge \mathbf{F}_{[1,2]}\, c \\ \Phi'_2 & = \mathbf{F}_{\leq 1}\,(b \wedge \mathbf{F}_{\leq 1}\, c) \\ \Phi'_3 & = \mathbf{F}_{\leq 1}\,((\mathbf{F}_{\leq 1}\, b) \wedge \mathbf{F}_{=1}\, c) \end{cases}$$

Soit κ une séquence temporisée. Si $\kappa \models \Phi'$, il est évident que $\kappa \models \Phi$. Supposons maintenant que $\kappa \models \Phi$, alors il existe $0 < t_1 < t_2 \leq 2$ tels que [1] $\kappa, t_1 \models b$ et $\kappa, t_2 \models c$. Si $t_1 \leq 1$ alors κ satisfait Φ'_1 ou Φ'_2 (ou les deux) selon que t_2 est plus petit ou plus grand que 1. Si $t_1 \in]1,2]$ alors il existe une date t' de $]0,1]$ telle que $\kappa, t' \models \mathbf{F}_{\leq 1}\, b \wedge \mathbf{F}_{=1}\, c$, ce qui implique que $\kappa \models \Phi'_3$. Nous représentons les trois cas possibles sur la figure 7.1 :

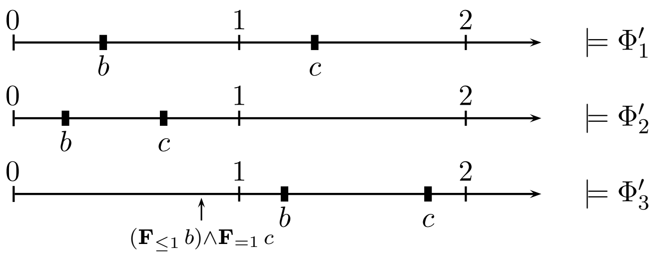


FIG. 7.1 – Traduction de la formule Φ de TPTL dans MTL

$\square$

D'après la proposition ci-dessus, nous obtenons que la formule de TPTL $\mathbf{G}\,(a \Rightarrow \Phi)$ est équivalente à la formule de MTL $\mathbf{G}\,(a \Rightarrow \Phi')$ pour les séquences temporisées.

[1] Nous ne mentionnons pas la valeur de l'horloge x car elle correspond à la date.

7.1.2 MTL est plus expressive que MITL pour la sémantique continue

La formule de MTL proposée dans la sous-section précédente utilise une contrainte *ponctuelle* $\mathbf{F}_{=1}$. Il est naturel de se demander si cela est vraiment nécessaire étant donné que la propriété initiale de TPTL n'est pas ponctuelle à priori. De manière surprenante, pour exprimer la formule Φ dans MTL, les contraintes ponctuelles sont nécessaires :

Proposition 7.1.2 *La formule $\Phi = x.(\mathbf{F}\,(b \wedge \mathbf{F}\,(c \wedge x \leq 2)))$ ne peut pas être exprimée dans* MITL *pour la sémantique continue.*

La *granularité* d'une formule φ est le plus petit dénominateur commun des constantes apparaissant dans φ. Si p est la granularité de φ, toute constante apparaissant dans φ est clairement un multiple de $\frac{1}{p}$. Nous utilisons la notation MITL_p (resp. MTL_p) pour l'ensemble des formules de MITL (resp. MTL) de granularité p.
Preuve. Nous construisons deux familles de mots temporisés vu comme des séquences temporisées $(\mathcal{A}_n)_{n\in\mathbb{N}^*}$ et $(\mathcal{B}_n)_{n\in\mathbb{N}^*}$ telles que :

(a) pour tout $n \in \mathbb{N}^*$, $\mathcal{A}_n \models \Phi$ et $\mathcal{B}_n \not\models \Phi$

(b) pour tout $p \in \mathbb{N}^*$ et tout $\varphi \in \mathsf{MITL}_p$, $\mathcal{A}_p \models \varphi \iff \mathcal{B}_p \models \varphi$.

La proposition 7.1.2 est une conséquence immédiate des propriétés (a) et (b) : si Φ a un équivalent dans MITL Ψ, soit p la granularité de Ψ. Par (a) $\mathcal{A}_p \models \Psi$ et $\mathcal{B}_p \not\models \Psi$; par (b) $\mathcal{A}_p \models \Psi \iff \mathcal{B}_p \models \Psi$ ce qui est une contradiction.

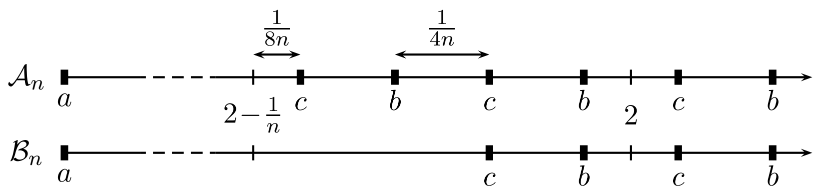


FIG. 7.2 – Les modèles $\mathcal{A}_n$ et $\mathcal{B}_n$

Les deux familles de modèles sont présentées sur la figure 7.2. Remarquons que sur les modèles $\mathcal{A}_n$ et $\mathcal{B}_n$, il n'y a pas d'action entre les dates 0 et $2-\frac{1}{n}$. Après la date $2-\frac{1}{n}$, le mot $\mathcal{A}_n$ est périodique de période $\frac{1}{2n}$: la proposition atomique c est vraie aux dates $2-\frac{7}{8n}+\frac{i}{2n}$ avec $i \geq 0$, la proposition atomique b est vraie aux dates $2-\frac{5}{8n}+\frac{i}{2n}$. Le mot $\mathcal{B}_n$ est obtenu en décalant $\mathcal{A}_n$ de $\frac{1}{2n}$ unités de temps.

Nous montrons tout d'abord que pour tout $p \in \mathbb{N}^*$, toute formule de MITL_p est uniformément vraie ou fausse sur certains intervalles de $\mathcal{A}_p$ et $\mathcal{B}_p$. Pour tous les entiers p et i avec $1 \leq i \leq 2p$, nous notons $J_{i,p}$ l'intervalle $]2-\frac{i}{p}-\frac{1}{8p}, 2-\frac{i}{p}+\frac{5}{8p}[\cap\mathbb{R}_{\geq 0}$.

Lemme 7.1.3 *Pour tous les entiers p et i avec $1 \leq i \leq 2p$, pour tout $\varphi \in \mathsf{MITL}_p$, et tout $x, y \in J_{i,p}$,*

$$\mathcal{B}_p, x \models \varphi \iff \mathcal{B}_p, y \models \varphi.$$

Preuve.[Preuve du lemme] Nous prouvons ce lemme par récurrence sur i. Nous prouvons tout d'abord l'étape de récurrence : supposons le résultat vrai jusqu'à $i-1$. Nous montrons le résultat pour i par induction sur la structure de φ. Cette induction est évidente si φ est une proposition atomique, ou si φ est la conjonction ou la négation de formules plus petites.

Le dernier cas est quand $\varphi = \varphi_1\, \mathbf{U}_I\, \varphi_2$. Dans la suite, nous écrivons q pour $\frac{1}{p}$. Comme $\varphi \in \mathsf{MITL}_p$, I est de la forme $I =]k_1q, k_2q[$, $I = [k_1q, k_2q[$, $I =]k_1q, k_2q]$ ou $I = [k_1q, k_2q]$, avec $k_1 <$

k_2. Nous montrons l'hypothèse d'induction pour les quatre cas en prouvant la propriété plus forte suivante : s'il existe $x \in J_{i,p}$ tel que $\mathcal{B}_p, x \models \varphi_1 \, \mathbf{U}_{[k_1q,k_2q]} \, \varphi_2$, alors pour tout $y \in J_{i,p}$, $\mathcal{B}_p, y \models \varphi_1 \, \mathbf{U}_{]k_1q,k_2q[} \, \varphi_2$.

Pour prouver cette implication, soit $x \in J_{i,p}$ tel que $\mathcal{B}_p, x \models \varphi_1 \, \mathbf{U}_{[k_1q,k_2q]} \, \varphi_2$. Soit $y \in J_{i,p}$, montrons que $\mathcal{B}_p, y \models \varphi_1 \, \mathbf{U}_{]k_1q,k_2q[} \, \varphi_2$.

Par définition de x, il existe $k_1q \leq t \leq k_2q$ tel que $\mathcal{B}_p, x+t \models \varphi_2$ et pour tout $0 < t' < t$, $\mathcal{B}_p, x+t' \models \varphi_1$. Par hypothèse d'induction sur φ_1, nous avons que pour tout $z \in J_{i,p}$, $\mathcal{B}_p, z \models \varphi_1$. Cela est vrai en particulier entre y et x si $y < x$.

Nous distinguons maintenant différents cas selon la position respective de x, y et t :

- Cas où $k_1q < x+t-y < k_2q$. C'est le cas où le témoin pour x est aussi correct pour y. En posant $t' = x+t-y$, nous avons que φ_1 est vraie entre y et x (si $y < x$), donc $\mathcal{B}_p, y \models \varphi_1 \, \mathbf{U}_{]k_1q,k_2q[} \, \varphi_2$.
- Cas où $x+t-y \leq k_1q$ (en particulier $x \leq y$)
  - Si $k_1 = 0$, alors $x+t \leq y < 2 - iq + \frac{5q}{8}$. Donc par hypothèse d'induction, φ_1 et φ_2 sont vraies partout dans l'intervalle $J_{i,p}$, donc $\mathcal{B}_p, y \models \varphi_1 \, \mathbf{U}_{]k_1q,k_2q[} \, \varphi_2$.
  - Si $k_1 \geq i$: nous pouvons supposer que $|x-y| < \frac{q}{2}$. En effet le cas général peut être retrouvé en considérant $z = \frac{(x+y)}{2}$ et en appliquant le lemme deux fois.
    Nous avons que $x+t > 2 - iq - \frac{q}{8} + k_1q \geq 2 - \frac{q}{8}$. Comme les modèles $\mathcal{B}_p, x+t$ et $\mathcal{B}_p, x+t+\frac{q}{2}$ sont identiques, nous avons que $\mathcal{B}_p, x+t+\frac{q}{2} \models \varphi_2$. Le point $x+t+\frac{q}{2}$ sera le témoin de $\varphi_1 \, \mathbf{U}_{]k_1q,k_2q[} \, \varphi_2$ en y. Nous devons montrer que
    1. φ_1 est satisfaite entre l'ancien témoin $(x+t)$ et le nouveau témoin $(x+t+\frac{q}{2})$: cela est vrai parce que pour tout $0 \leq z < \frac{q}{2}$, les modèles $\mathcal{B}_p, x+t+z$ et $\mathcal{B}_p, x+t+z-\frac{q}{2}$ sont identiques, et $\mathcal{B}_p, x+t+z-\frac{q}{2}$ satisfait φ_1.
    2. $t' = x+t-y+\frac{q}{2}$ est dans l'intervalle $]k_1q, k_2q[$: $t' \leq k_1q + \frac{q}{2} < k_2q$ (car $x+t-y \leq k_1q$) et $t' > t \geq k_1q$ (car $|x-y| < \frac{q}{2}$).
  - Si $0 < k_1 < i$ (ce qui implique que $i > 1$), nous prouvons l'existence d'un témoin dans l'intervalle $J_{i-k_1,p}$.
    Nous avons que $x+t > 2 - iq - \frac{q}{8} + k_1q = 2 - (i-k_1)q - \frac{q}{8}$, et $x+t = x+t-y+y < k_1q + 2 - iq + \frac{5q}{8}$, donc $x+t$ est dans $J_{i-k_1,p}$. Nous appliquons l'hypothèse de récurrence pour $i-k_1$, et obtenons que φ_1 et φ_2 sont satisfaites partout dans $J_{i-k_1,p}$. En prenant $t' = k_1q + \frac{2-iq+5\frac{q}{8}-y}{2}$, il est facile de vérifier que $k_1q < t' < k_2q$ et $y+t' \in J_{i-k_1,p}$.
- Cas où $x+t-y \geq k_2q$ (en particulier $x > y$)
  - Si $k_2 \geq i$: nous supposons à nouveau que $|x-y| < \frac{q}{2}$.
    Comme $x+t = x+t-y+y > k_2q + 2 - iq - \frac{q}{8} \geq 2 - \frac{q}{8}$ et $\mathcal{B}_p, x+t \models \varphi_2$, nous avons que $\mathcal{B}_p, x+t-\frac{q}{2} \models \varphi_2$. Nous devons encore montrer que le nouveau témoin $t' = x+t-y-\frac{q}{2}$ est dans le bon intervalle : nous avons que $t' < t \leq k_2q$ car $|x-y| < \frac{q}{2}$, et $t' \geq k_2q - \frac{q}{2} > k_1q$. De plus, comme nous l'avons montré avant, φ_1 est satisfaite entre y et x.
  - Si $k_2 < i$ (donc $i > 1$), nous construisons un autre témoin dans l'intervalle $J_{i-k_2,p}$.
    A nouveau, $x+t = x+t-y+y > 2 + k_2q - iq - \frac{q}{8}$ et $x+t < 2 - iq + 5\frac{q}{8} + k_2q$ donc $x+t$ est dans $J_{i-k_2,p}$. Nous appliquons l'hypothèse de récurrence pour $i-k_2$, et obtenons que φ_1 et φ_2 sont satisfaites partout dans $J_{i-k_2,p}$. Nous montrons facilement que $t' = k_2q - \frac{y-(2-iq-q/8)}{2}$ est un témoin de $\varphi_1 \, \mathbf{U}_{]k_1q,k_2q[} \, \varphi_2$ en y.

Le cas $i = 1$ est une conséquence de la preuve ci-dessus, car nous utilisons l'hypothèse de récurrence que pour les cas où $i > 1$. □

Nous déduisons facilement le lemme suivant :

Lemme 7.1.4 *Pour tout* $p \in \mathbb{N}^*$*, pour tout* $\varphi \in \mathsf{MITL}_p$*,* $\mathcal{A}_p, 0 \models \varphi \iff \mathcal{B}_p, 0 \models \varphi$.

Preuve.[Preuve du lemme] Soit φ une formule de MITL_p. Alors

$$\begin{aligned}
\mathcal{A}_p, 0 \models \varphi &\iff \mathcal{B}_p, \frac{1}{2p} \models \varphi \text{ car les suffixes à partir de} \mathcal{A}_p, 0 \text{ et } \mathcal{B}_p, \frac{p}{2} \text{ sont les mêmes} \\
&\iff \forall x \in \left[0, \frac{5}{8p}\right[\ ,\ \mathcal{B}_p, x \models \varphi \text{ par le lemme 7.1.3 pour } i = 2p \\
&\iff \mathcal{B}_p, 0 \models \varphi
\end{aligned}$$

□

Ce lemme achève la preuve de la proposition 7.1.2. □

Comme corollaire de la proposition 7.1.2, nous obtenons le théorème suivant :

Théorème 7.1.5 MTL *est strictement plus expressive que* MITL *pour la sémantique continue.*

7.1.3 TPTL et MTL pour la sémantique d'actions

Nous avons montré précédemment que la formule d'Alur et Henzinger n'était pas un bon témoin pour montrer que TPTL est strictement plus expressive que MTL pour la sémantique continue. Nous montrons dans cette sous-section que pour la sémantique d'actions, cette formule est cependant un bon témoin :

Proposition 7.1.6 *La formule de* TPTL $\Phi = x.(\mathbf{F}\ (b \wedge \mathbf{F}\ (c \wedge x \leq 2)))$ *n'a pas d'équivalent dans* MTL *pour la sémantique d'actions.*

Preuve. Nous conservons les notations de la sous-section 7.1.2, et en particulier nous considérons à nouveau les familles de modèles (maintenant considérées comme des mots temporisés) $(\mathcal{A}_n)_{n\in\mathbb{N}^*}$ et $(\mathcal{B}_n)_{n\in\mathbb{N}^*}$ décrites sur la figure 7.2. Comme précédemment MTL_p désigne le fragment de MTL contenant les formules de granularité p. Comme dans la sous-section précédente nous prouvons que :

(a) pour tout $p \in \mathbb{N}^*$, $\mathcal{A}_p \models \Phi$ et $\mathcal{B}_p \not\models \Phi$

(b) pour tout $p \in \mathbb{N}^*$ et tout $\varphi \in \mathsf{MTL}_p$, $\mathcal{A}_p \models \varphi \iff \mathcal{B}_p \models \varphi$

La propriété (a) est évidente. Pour prouver la propriété (b), nous montrons tout d'abord le lemme facile suivant :

Lemme 7.1.7 *Pour tout* $p \in \mathbb{N}^*$*, pour tout* $k, k' \geq 1$ *tels que* $k = k' \mod 2$*, et pour toute formule* $\varphi \in \mathsf{MTL}_p$*,*

$$\mathcal{A}_p, k \models \varphi \iff \mathcal{B}_p, k \models \varphi \iff \mathcal{A}_p, k' \models \varphi \iff \mathcal{B}_p, k' \models \varphi.$$

Ce résultat est évident car les suffixes à partir de $\mathcal{A}_p, k$, $\mathcal{B}_p, k$, $\mathcal{A}_p, k'$, et $\mathcal{B}_p, k'$ sont les mêmes.

Nous prouvons maintenant le lemme suivant :

Lemme 7.1.8 *Pour tout $p \in \mathbb{N}^*$ et toute formule $\varphi \in \mathsf{MTL}_p$, $\mathcal{A}_p, 0 \models \varphi \iff \mathcal{B}_p, 0 \models \varphi$.*

Nous raisonnons par induction sur la structure de φ. Le cas des propositions atomiques est facile, de même que l'étape d'induction pour la conjonction et la négation. Nous posons $q = \frac{1}{p}$.

Supposons maintenant que $\varphi = \varphi_1 \, \mathbf{U}_I \, \varphi_2$. Remarquons que pour tout $k \in \mathbb{N}$ il n'y a pas d'action aux dates kq dans $\mathcal{A}_p$ ou $\mathcal{B}_p$. Nous avons donc que pour tout $k_1, k_2 \in \mathbb{N}$,

$$\mathcal{A}_p \models \varphi_1 \, \mathbf{U}_{[k_1q,k_2q]} \, \varphi_2 \iff \mathcal{A}_p \models \varphi_1 \, \mathbf{U}_{]k_1q,k_2q]} \, \varphi_2$$
$$\iff \mathcal{A}_p \models \varphi_1 \, \mathbf{U}_{[k_1q,k_2q[} \, \varphi_2 \iff \mathcal{A}_p \models \varphi_1 \, \mathbf{U}_{]k_1q,k_2q[} \, \varphi_2$$

et de même pour $\mathcal{B}_p$. Sans perte de généralité nous supposons donc que $I =]k_1q, k_2q[$.

Dans ce qui suit, nous écrivons τ_i pour la date associée à la position i dans $\mathcal{A}_p$, et τ'_j pour la date associée à la position j dans $\mathcal{B}_p$.

- Supposons que $\mathcal{A}_p, 0 \models \varphi$, et montrons que $\mathcal{B}_p, 0 \models \varphi$. Nous savons qu'il existe $i > 0$ avec $\tau_i \in I$, $\mathcal{A}_p, i \models \varphi_2$, et tel que pour tout $0 < k < i$, $\mathcal{A}_p, k \models \varphi_1$. Nous distinguons deux cas :
  - Si $i \geq 3$: nous prenons $j = i - 2$; alors $\tau'_j = \tau_i$, et $\tau'_j \in I$. Par le lemme 7.1.7, nous avons que $\mathcal{B}_p, j \models \varphi_2$. Comme $\mathcal{A}_p, 1$ et $\mathcal{A}_p, 2$ satisfont φ_1, ce lemme assure également que pour tout $k > 0$, $\mathcal{B}_p, k \models \varphi_1$. Donc $\mathcal{B}_p, 0 \models \varphi_1 \, \mathbf{U}_I \, \varphi_2$
  - Si $1 \leq i \leq 2$: alors $\tau_i \in \{2 - \frac{7q}{8}, 2 - \frac{5q}{8}\}$, ce qui implique $k_2q \geq 2$. En prenant $j = i$ et en appliquant le lemme 7.1.7, nous obtenons que $\mathcal{B}_p, j \models \varphi_2$ et pour tout $0 < k < j$, $\mathcal{B}_p, k \models \varphi_1$. Comme $j = i$, nous avons $\tau'_j = \tau_i + \frac{q}{2}$, donc $\tau'_j \geq \tau_i > k_1q$ et $\tau'_j \leq 2 - \frac{q}{8} \leq 2 \leq k_2p$. Donc $\mathcal{B}_p, 0 \models \varphi_1 \, \mathbf{U}_{]k_1q,k_2q[} \, \varphi_2$, et en particulier $\mathcal{B}_p, 0 \models \varphi_1 \, \mathbf{U}_I \, \varphi_2$
- Réciproquement supposons que $\mathcal{B}_p, 0 \models \varphi$. Alors il existe $j > 0$ tel que $\tau'_j \in I$, $\mathcal{B}_p, j \models \varphi_2$, et pour tout $0 < k < j$, $\mathcal{B}_p, k \models \varphi_1$. Nous distinguons deux cas :
  - Si $j \geq 3$: nous prenons alors $i = j + 2$. Dans ce cas $\tau_i = \tau'_j$ et $\tau_i \in I$. Par le lemme 7.1.7, nous avons que $\mathcal{A}_p, i \models \varphi_2$. À nouveau, comme $\mathcal{B}_p, 1$ et $\mathcal{B}_p, 2$ satisfont φ_1, le lemme 7.1.7 asssure que pour tout $k > 0$, $\mathcal{A}_p, k \models \varphi_1$. Donc $\mathcal{A}_p, 0 \models \varphi_1 \, \mathbf{U}_I \, \varphi_2$
  - Si $1 \leq j \leq 2$: alors $\tau'_j \in \{2 - \frac{3q}{8}, 2 - \frac{q}{8}\}$, ce qui implique que $k_1q \leq 2 - q$. Nous prenons $i = j$: le lemme 7.1.7 assure que $\mathcal{A}_p, i \models \varphi_2$ et que pour tout $0 < k < i$, $\mathcal{A}_p, k \models \varphi_1$. Nous avons $\tau_i = \tau'_j - \frac{q}{2}$, donc $\tau_i < \tau'_j \leq k_2p$ et $\tau_i \geq 2 - \frac{7q}{8} > 2 - q \geq k_1q$. D'où $\mathcal{A}_p, 0 \models \varphi_1 \, \mathbf{U}_{]k_1q,k_2q[} \, \varphi_2$, et $\mathcal{A}_p, 0 \models \varphi_1 \, \mathbf{U}_I \, \varphi_2$.

□

Comme corollaire direct de la proposition 7.1.6, nous obtenons le théorème suivant : :

Théorème 7.1.9 TPTL *est strictement plus expressive que* MTL *pour la sémantique d'actions.*

Comme la formule $\mathbf{F}_{\leq 2} \, (c \wedge \top \, \mathbf{S} \, b)$ de MITL+Past distingue également les familles $(\mathcal{A}_p)_{p \in \mathbb{N}}$ et $(\mathcal{B}_p)_{p \in \mathbb{N}}$, nous avons le corollaire suivant :

Corollaire 7.1.10 *Pour la sémantique d'actions,* MTL+Past *et* MITL+Past *sont respectivement strictement plus expressives que* MTL *et* MITL.

Le résultat pour MITL a déjà été prouvé par des techniques différentes dans [AH92b]. À notre connaissance, le résultat pour MTL n'était pas connu avant. Remarquons que le corollaire 7.1.10 établit une importante différence entre le cadre temporisé et le cadre non temporisé où il est bien connu qu'ajouter des modalités du passé n'augmente pas l'expressivité de LTL [Kam68, GPSS80].

7.1.4 TPTL et MTL pour la sémantique continue

Comme nous l'avons vu, la formule utilisée pour la sémantique d'actions ne peut être un témoin pour la sémantique continue. Nous utilisons une autre formule et montrons la proposition suivante :

Proposition 7.1.11 *La formule de* TPTL $\Phi = x.\mathbf{F}\,(a \wedge x \leq 1 \wedge \mathbf{G}\,(x \leq 1 \Rightarrow \neg b))$ *n'a pas d'équivalent dans* MTL *pour la sémantique continue.*

Preuve. Soit $p \in \mathbb{N}^*$ et $q = \frac{1}{p}$. Quitte à augmenter la hauteur temporelle d'une formule Ψ de MTL, nous pouvons supposer que Ψ utilise uniquement des contraintes de la forme $\sim q$, avec $\sim \in \{<, =, >\}$. Nous définissons $\mathsf{MTL}^{-}_{p,n}$ le fragment de MTL composé des formules utilisant uniquement des contraintes $\sim q$ et dont la hauteur temporelle est au plus n.

Nous construisons deux familles de mots temporisés vus comme des séquences temporisées $\mathcal{A}_{p,n}$ et $\mathcal{B}_{p,n}$ telles que :

(a) $\mathcal{A}_{p,n}, 0 \models \Phi$ et $\mathcal{B}_{p,n}, 0 \not\models \Phi$

(b) $\mathcal{A}_{p,n}$ et $\mathcal{B}_{p,n}$ ne peuvent être distingués par une formule de $\mathsf{MTL}^{-}_{p,n-3}$.

Nous définissons tout d'abord $\mathcal{A}_{p,n}$. Sur cette séquence temporisée, a est vraie exactement aux dates $\frac{q}{4n} + \alpha\frac{q}{2n}$, où $\alpha \in \mathbb{N}$. La proposition atomique b est vraie exactement aux dates $(\alpha+1)\cdot\frac{q}{2} - \frac{4q}{6n}$, où $\alpha \in \mathbb{N}$.

$\mathcal{B}_{p,n}$ a exactement les mêmes a que $\mathcal{A}_{p,n}$, alors que b est vraie dans $\mathcal{B}_{p,n}$ aux dates $(\alpha+1)\cdot\frac{q}{2} - \frac{q}{6n}$, où $\alpha \in \mathbb{N}$.

Les portions entre 0 et $\frac{q}{2}$ des deux séquences temporisées sont représentées sur la figure 7.3. Les deux séquences temporisées sont en fait périodiques de période $\frac{q}{2}$.

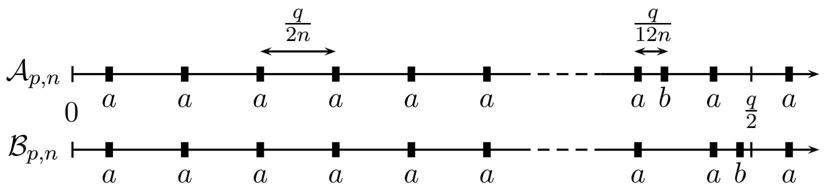


FIG. 7.3 – Les modèles $\mathcal{A}_{p,n}$ et $\mathcal{B}_{p,n}$

Le lemme suivant est évident car pour chaque équivalence les suffixes sont les mêmes :

Lemme 7.1.12 *Pour tout entier strictement positif p, pour tout entier n, pour tout réel positif x, et pour toute formule φ de* MTL*, en posant $q = \frac{1}{p}$, nous avons les propriétés suivantes :*

$$\mathcal{A}_{p,n}, x \models \varphi \quad\Longleftrightarrow\quad \mathcal{B}_{p,n}, x + \frac{q}{2n} \models \varphi \tag{7.2}$$

$$\mathcal{A}_{p,n}, x \models \varphi \quad\Longleftrightarrow\quad \mathcal{A}_{p,n}, x + \frac{q}{2} \models \varphi \tag{7.3}$$

$$\mathcal{B}_{p,n}, x \models \varphi \quad\Longleftrightarrow\quad \mathcal{B}_{p,n}, x + \frac{q}{2} \models \varphi \tag{7.4}$$

Nous prouvons le lemme suivant :

Lemme 7.1.13 *Soit $p \in \mathbb{N}^*$, et $q = \frac{1}{p}$. Pour tout $k \leq n$, pour toute formule $\varphi \in \mathsf{MTL}^-_{p,k}$, pour tout $x \in \left[0, \frac{q}{2} - \frac{(k+2)q}{2(n+3)}\right[$, pour tout $\alpha \in \mathbb{N}$, nous avons*

$$\mathcal{A}_{p,n+3}, \alpha\frac{q}{2} + x \models \varphi \qquad \Longleftrightarrow \qquad \mathcal{B}_{p,n+3}, \alpha\frac{q}{2} + x \models \varphi$$

Preuve. Nous raisonnons par récurrence sur k. Pour améliorer la lisibilité nous écrivons $\mathcal{A}$ et $\mathcal{B}$ pour $\mathcal{A}_{p,n+3}$ et $\mathcal{B}_{p,n+3}$ respectivement et posons $\delta = \frac{q}{2(n+3)}$.

- Le cas $k = 0$ est facile, car φ ne peut être qu'une proposition atomique et pour toutes les dates dans l'intervalle considéré, $\mathcal{A}$ et $\mathcal{B}$ ont les mêmes propositions atomiques.
- Supposons le résultat vrai pour k et montrons le pour $k+1$. Soit $\varphi \in \mathsf{MTL}^-_{p,k+1}$, remarquons que nous pouvons supposer φ de hauteur temporelle égale à $k+1$ sinon le résultat est déjà vrai par hypothèse de récurrence.
  - Si $\varphi = \varphi_1 \mathbf{U}_{=q} \varphi_2$: soit $x \in \left[0, \frac{q}{2} - ((k+1)+2)\delta\right[$ et $\alpha \in \mathbb{N}$, et supposons que $\mathcal{A}, \alpha\frac{q}{2} + x \models \varphi_1 \mathbf{U}_{=q} \varphi_2$. Alors φ_2 est vraie à la date $(\alpha+2)\frac{q}{2} + x$, et φ_1 est vraie pour toutes les dates intermédiaires. En appliquant l'hypothèse de récurrence, nous obtenons que $\mathcal{B}, (\alpha+2)\frac{q}{2} + x \models \varphi_2$. Nous obtenons également que $\mathcal{B}$ vérifie φ_1 pour toutes les dates entre $\alpha\frac{q}{2} + x$ et $\alpha\frac{q}{2} + x + \delta$. Par l'équation (7.2), φ_1 est également vérifiée dans $\mathcal{B}$ entre les dates $\alpha\frac{q}{2}+x+\delta$ et $(\alpha+2)\frac{q}{2}+x$. Il vient donc que $\mathcal{B}, \alpha\frac{q}{2}+x \models \varphi$.
    Réciproquement supposons que $\mathcal{B}, \alpha\frac{q}{2} + x \models \varphi_1 \mathbf{U}_{=q} \varphi_2$. Par hypothèse de récurrence $\mathcal{A}, (\alpha+2)\frac{q}{2} + x \models \varphi_2$. Par l'équation (7.2), nous savons que φ_1 est vraie dans $\mathcal{B}$ entre $\alpha\frac{q}{2} + x$ et $(\alpha+2)\frac{q}{2} + x - \delta$. Enfin, l'équation (7.3) assure que φ_1 est également vraie dans $\mathcal{B}$ entre $(\alpha+2)\frac{q}{2} + x - \delta$ et $(\alpha+2)\frac{q}{2} + x$, ce qui clôt ce cas.
  - Supposons que $\varphi = \varphi_1 \mathbf{U}_{<q} \varphi_2$: soit $x \in \left[0, \frac{q}{2} - ((k+1)+2)\delta\right[$ et $\alpha \in \mathbb{N}$ et supposons que $\mathcal{A}, \alpha\frac{q}{2} + x \models \varphi_1 \mathbf{U}_{<q} \varphi_2$.
    - Si le témoin pour φ_2 est entre $\alpha\frac{q}{2}+x$ et $(\alpha+1)\frac{q}{2}+x$, en appliquant l'équation (7.2), nous obtenons que $\mathcal{B}, \alpha\frac{q}{2} + x + \delta \models \varphi_1 \mathbf{U}_{<\frac{q}{2}} \varphi_2$. L'hypothèse de récurrence assure que φ_1 est vraie dans $\mathcal{B}$ entre $\alpha\frac{q}{2} + x$ et $\alpha\frac{q}{2} + x + \delta$, d'où nous déduisons que $\mathcal{B}, \alpha\frac{q}{2} + x \models \varphi_1 \mathbf{U}_{<q} \varphi_2$.
    - Si le témoin est entre $(\alpha+1)\frac{q}{2}+x$ et $(\alpha+2)\frac{q}{2}+x$, par l'équation (7.3), il y a également un témoin entre $\alpha\frac{q}{2} + x$ et $(\alpha+1)\frac{q}{2} + x$ et nous appliquons le cas précédent.

    Réciproquement, supposons que $\mathcal{B}, \alpha\frac{q}{2} + x \models \varphi_1 \mathbf{U}_{<q} \varphi_2$. Nous distinguons ici aussi deux cas :
    - Si le témoin pour φ_2 est entre $\alpha\frac{q}{2} + x$ et $\alpha\frac{q}{2} + x + \delta$, il suffit d'appliquer l'hypothèse de récurrence à φ_1 et φ_2.
    - Sinon il suffit d'appliquer l'équation (7.2).
  - Enfin, supposons que $\varphi = \varphi_1 \mathbf{U}_{>q} \varphi_2$: soit $x \in \left[0, \frac{q}{2} - ((k+1)+2)\delta\right[$, $\alpha \in \mathbb{N}$, et supposons que $\mathcal{A}, \alpha\frac{q}{2} + x \models \varphi_1 \mathbf{U}_{>q} \varphi_2$. En appliquant l'équation (7.2) et l'hypothèse de récurrence à φ_1, nous obtenons que $\mathcal{B}, \alpha\frac{q}{2} + x \models \varphi_1 \mathbf{U}_{>q} \varphi_2$.
    Réciproquement, supposons que $\mathcal{B}, \alpha\frac{q}{2} + x \models \varphi_1 \mathbf{U}_{>q} \varphi_2$. Si le témoin pour φ_2 est après $(\alpha+2)\frac{q}{2} + x + \delta$, il suffit d'appliquer l'équation (7.2). Sinon, l'équation (7.3) assure que nous pouvons trouver un autre témoin pour φ_2 après $(\alpha+2)\frac{q}{2} + x + \delta$. Cela achève la preuve du lemme.

□

En appliquant le lemme précédent avec $k = n$ et $\alpha = x = 0$, nous trouvons que toute formule de $\mathsf{MTL}^-_{p,n}$ ne peut distinguer $\mathcal{A}_{p,n+3}$ et $\mathcal{B}_{p,n+3}$. Alors la formule Φ ne peut avoir d'équivalent dans MTL, car cette formule devrait satisfaire les équations (a) et (b). Nous avons donc prouvé la proposition 7.1.11. $\square$

Nous déduisons le théorème suivant :

Théorème 7.1.14 TPTL *est strictement plus expressive que* MTL *pour la sémantique continue.*

Remarquons que la formule $x.\mathbf{F}\,(a \wedge x \leq 1 \wedge \mathbf{G}\,(x \leq 1 \Rightarrow \neg b))$ n'utilise pas la modalité $\mathbf{U}$, donc le fragment de TPTL utilisant uniquement les modalités $\mathbf{F}$ et $\mathbf{G}$ est lui aussi strictement plus expressif que le fragment correspondant de MTL. Ce n'est pas le cas du fragment de TPTL utilisant uniquement la modalité $\mathbf{F}$ (voir la section 7.2).

Il est clair que la formule $\mathbf{F}_{=1}\,(\neg b\,\mathbf{S}\,a)$ de MTL+Past distingue les deux familles de modèles $(\mathcal{A}_{p,n})_{p\in\mathbb{N},n\in\mathbb{N}}$ et $(\mathcal{B}_{p,n})_{p\in\mathbb{N},n\in\mathbb{N}}$. La formule plus élaborée de MITL+Past

$$\mathbf{F}_{\geq 1}\,((\mathbf{F}^{-1}_{\geq -1}\,((\mathbf{G}^{-1}\,\neg a)) \wedge \neg b\,\mathbf{S}\,a) \tag{7.5}$$

distingue également ces modèles : la sous-formule $\mathbf{F}^{-1}_{\geq -1}\,(\mathbf{G}^{-1}\,\neg a)$ assure qu'il y a un point pas trop éloigné dans le passé (au plus une unité de temps) tel que a n'a jamais été vrai dans le passé. Dans les deux modèles, ce point est nécessairement entre les dates 0 et $\frac{q}{4n}$, et $\mathbf{F}^{-1}_{\geq -1}\,(\mathbf{G}^{-1}\,\neg a)$ est vraie exactement entre les dates 0 et $1+\frac{q}{4n}$. Donc la formule (7.5) exprime qu'il y a un point entre les dates 1 et $1+\frac{q}{4n}$ où $\neg b\,\mathbf{S}\,a$ doit être vérifiée. Cette formule est satisfaite dans $\mathcal{A}_{p,n}$, pour tout n et p, mais n'est satisfaite pour aucun $\mathcal{B}_{p,n}$. Nous avons donc le corollaire suivant :

Corollaire 7.1.15 MTL+Past *et* MITL+Past *sont respectivement strictement plus expressives que* MTL *et* MITL *pour la sémantique continue.*

A notre connaissance, ce sont les premiers résultats d'expressivité pour des logiques temporisées linéaires utilisant le passé pour la sémantique continue.

7.2 Fragment existentiel de MTL et TPTL

$\mathsf{TPTL}_{\mathbf{F}}$ est le fragment de TPTL qui utilise uniquement la modalité $\mathbf{F}$ (et pas la modalité plus générale $\mathbf{U}$) et restreint la négation aux propositions atomiques. Formellement, $\mathsf{TPTL}_{\mathbf{F}}$ est définie comme suit :

$$\varphi ::= p \mid \neg p \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \mathbf{F}\,\varphi \mid x \sim c \mid x.\varphi$$

où p est une proposition atomique, x une horloge, $c \in \mathbb{Q}$ et $\sim \in \{\leq, <, =, >, \geq\}$.

Un exemple de formule de $\mathsf{TPTL}_{\mathbf{F}}$ est $x.\mathbf{F}\,(b \wedge \mathbf{F}\,(c \wedge x \leq 2))$ (voir la sous-section 7.1.1). De même nous définissons le fragment $\mathsf{MTL}_{\mathbf{F}}$ de MTL où seule la modalité $\mathbf{F}$ est autorisée :

$$\varphi ::= p \mid \neg p \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \mathbf{F}_I\,\varphi$$

où p est une proposition atomique et $I \in \mathcal{I}_{\mathbb{Q}}$.

D'après la sous-section 7.1.3, nous savons que pour la sémantique d'actions, $\mathsf{TPTL}_{\mathbf{F}}$ est strictement plus expressive que $\mathsf{MTL}_{\mathbf{F}}$, car la formule $x.\mathbf{F}\,(b \wedge \mathbf{F}\,(c \wedge x \leq 2))$ n'a pas d'équivalent dans MTL (et donc dans $\mathsf{MTL}_{\mathbf{F}}$). Par contre, pour la sémantique continue, nous avons prouvé que la formule ci-dessus peut être exprimée dans $\mathsf{MTL}_{\mathbf{F}}$ (voir la sous-section 7.1.1). Nous généralisons maintenant la construction de la sous-section 7.1.1 et montrons que $\mathsf{TPTL}_{\mathbf{F}}$ et $\mathsf{MTL}_{\mathbf{F}}$ ont la même expressivité pour la sémantique continue.

Théorème 7.2.1 $\mathsf{TPTL}_{\mathbf{F}}$ *et* $\mathsf{MTL}_{\mathbf{F}}$ *ont la même expressivité pour la sémantique continue.*

Preuve. Nous supposons sans perte de généralité que toutes les contraintes apparaissant dans des formules de $\mathsf{TPTL}_{\mathbf{F}}$ sont des entiers. Pour toute formule de $\mathsf{TPTL}_{\mathbf{F}}$, nous construisons une formule équivalente de $\mathsf{MTL}_{\mathbf{F}}$ pour la sémantique continue. La construction se fait en six étapes.

1. Forme normale pour les formules de $\mathsf{TPTL}_{\mathbf{F}}$.

Quitte à ajouter de nouvelles horloges, nous supposons que toute occurrence d'une modalité $\mathbf{F}$ est sous la portée directe d'une remise à zéro « $x.$ », et que toute horloge x apparaissant dans la formule est remise à zéro au plus une fois. Nous considérons donc des formules de la logique définie par :

$$\varphi ::= p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid x \sim c \mid x.\mathbf{F}\,\varphi \tag{7.6}$$

telle que toute horloge apparaît au plus une fois dans une remise à zéro « $x.\mathbf{F}$ ».

Nous construisons maintenant récursivement une forme normale pour les formules de $\mathsf{TPTL}_{\mathbf{F}}$ qui est en quelque sorte une forme normale disjonctive. Nous appelons *atome* une proposition atomique ou sa négation.

Définition 7.2.2 *Une formule de* $\mathsf{TPTL}_{\mathbf{F}}$ *est* simple *si c'est un atome, une contrainte d'horloges ou si elle est de la forme :*

$$x.\mathbf{F}\left(\bigwedge_{k=1}^{h_1} a_k \wedge \left(\bigwedge_{k=1}^{h_2} x_{i_k} \sim_k c_k\right) \wedge \bigwedge_{k=1}^{h_3} \varphi_k\right)$$

où a_k sont des atomes, x_k des horloges, $\sim_k \in \{\leq, <, =, >, \geq\}$, c_k sont des rationnels positifs, et φ_k sont des formules simples de $\mathsf{TPTL}_{\mathbf{F}}$ $x_{j_k}.\mathbf{F}\,\psi_k$.

Le lemme suivant est évident en utilisant la propriété que $x.\mathbf{F}\,(\varphi_1 \vee \varphi_2)$ est équivalent à $(x.\mathbf{F}\,\varphi_1) \vee (x.\mathbf{F}\,\varphi_2)$.

Lemme 7.2.3 *Toute formule de* $\mathsf{TPTL}_{\mathbf{F}}$ *est équivalente à une combinaison booléenne positive de formules simples de* $\mathsf{TPTL}_{\mathbf{F}}$.

Le problème initial se ramène donc à construire pour toute formule simple de $\mathsf{TPTL}_{\mathbf{F}}$, une formule équivalente de $\mathsf{MTL}_{\mathbf{F}}$.

2. Des formules simples de $\mathsf{TPTL}_{\mathbf{F}}$ aux systèmes d'inéquations diagonales.

Cette étape consiste à transformer récursivement une formule simple de $\mathsf{TPTL}_{\mathbf{F}}$ en un système d'inéquations, où chaque sous-formule $x.\mathbf{F}\,\varphi$ est représentée par la date y à laquelle φ est vérifiée. Cela génère des conditions entre y et les autres dates et horloges déjà construites.

Nous définissons tout d'abord *les systèmes d'inéquations diagonales*[2] qui seront associés aux formules de $\mathsf{TPTL}_{\mathbf{F}}$.

Définition 7.2.4 *Soit X un ensemble d'horloges et Y un ensemble fini de variables disjoint de X. Un* système $\mathcal{S}$ *sur X et Y est un couple $(V, \mathcal{J})$ où $V\colon Y \to \mathsf{MTL}_{\mathbf{F}}$ associe à toute variable $y \in Y$ une formule de $\mathsf{MTL}_{\mathbf{F}}$ $V(y)$, et $\mathcal{J}$ est une ensemble d'inéquations (diagonales) de la forme $x - x' \sim c$ ou $x \sim c$ où x, x' sont des éléments de $X \cup Y$, $\sim \in \{<, \leq, =, \geq, >\}$, et $c \in \mathbb{Z}$ est un entier relatif.*

Intuitivement, un tel système $\mathcal{S}$ représente une propriété sur les séquences temporisées où les formules de $\mathsf{MTL}_{\mathbf{F}}$ données par V doivent être satisfaites aux dates satisfaisant les contraintes données par le système d'inéquations $\mathcal{J}$.

Par abus de notation, si $\mathcal{S} = (V, \mathcal{J})$ est un système sur X et Y, pour tout $y \in Y$, nous écrivons $\mathcal{S}(y)$ pour $V(y)$, et si e est une inéquation de $\mathcal{J}$, nous notons $e \in \mathcal{S}$.

Soit $\mathcal{S}$ un système sur X et Y, κ une séquence temporisée, $v\colon Y \to \mathbb{R}_{\geq 0}$ une valuation, et $\alpha\colon X \to \mathbb{R}_{\geq 0}$ une interprétation des horloges de X. Nous disons que $\kappa, v, \alpha \vdash \mathcal{S}$ si l'interprétation $\alpha'\colon X \cup Y \to \mathbb{R}_{\geq 0}$, qui étend naturellement v et α satisfait les conditions suivantes :

$$\forall e \in \mathcal{S},\ \alpha' \models e \quad \text{et} \quad \forall y \in Y,\ \kappa, v(y) \models \mathcal{S}(y).$$

La relation de satisfaction est alors définie par[3] :

$$\begin{aligned}\kappa, t, \alpha \models \mathcal{S} \quad \text{si} \quad & \exists v\colon Y \to \mathbb{R}_{\geq 0} \text{ tel que } \kappa, v, \alpha \vdash \mathcal{S}, \\ & \qquad \forall y \in Y,\ v(y) \geq t \text{ et } \exists y_0 \in Y,\ v(y_0) = t.\end{aligned}$$

La conjonction de deux systèmes $\mathcal{S} = (V, \mathcal{J})$ et $\mathcal{S}' = (V', \mathcal{J}')$, notée $\mathcal{S} \wedge \mathcal{S}'$, est le système $(V'', \mathcal{J} \cup \mathcal{J}')$ où $V''\colon y \mapsto V(y) \wedge V'(y)$.

Soit φ une formule simple de $\mathsf{TPTL}_{\mathbf{F}}$ avec pour horloges X_φ. Nous expliquons comment construire inductivement un système $\mathcal{S}_\varphi$ sur X_φ et un ensemble de variables Y_φ tel que

$$\kappa, 0, \alpha \models_c \varphi \quad \text{ssi} \quad \kappa, 0, \alpha \models \mathcal{S}_\varphi. \tag{7.7}$$

- Si φ est un atome, l'ensemble des variables Y_φ contient une unique variable y, le système n'a pas d'inéquations, et $\mathcal{S}(y) = \varphi$.
- Si φ est une contrainte $x \sim c$, l'ensemble Y_φ contient une unique variable y, le système a une unique inéquation $y \sim c$, et $\mathcal{S}(y) = \top$.
- Nous supposons que $\varphi = x.\mathbf{F}\ \left(\bigwedge_{k=1}^{h_1} a_k \wedge \left(\bigwedge_{k=1}^{h_2} x_{i_k} \sim_k c_k\right) \wedge \bigwedge_{k=1}^{h_3} \varphi_k\right)$, où φ_k est une formule simple de $\mathsf{TPTL}_{\mathbf{F}}$ de la forme $x_{j_k}.\mathbf{F}\ \psi_k$. Nous supposons que nous avons déjà construit pour tout $1 \leq k \leq h_3$, un système $\mathcal{S}_{\varphi_k}$ sur X_{φ_k} et Y_{φ_k} qui correspond à φ_k dans le sens de l'équivalence (7.7). Nous construisons le système $\mathcal{S}_\varphi$ comme suit. L'ensemble des variables Y_φ est $\bigcup_{k=1}^{h_3} Y_{\varphi_k} \cup \{y\}$, où y est une variable fraîche représentant la date à laquelle la sous-formule $\bigwedge_{k=1}^{h_1} a_k \wedge \left(\bigwedge_{k=1}^{h_2} x_{i_k} \sim_k c_k\right) \wedge \bigwedge_{k=1}^{h_3} \varphi_k$ sera vérifiée. Le système $\mathcal{S}_\varphi$ est alors défini par la conjonction suivante :

$$\mathcal{S}_\varphi = \bigwedge_{1 \leq k \leq h_3} \mathcal{S}_{\varphi_k}[x_{j_k} \leftarrow y] \ \wedge \ \begin{cases} V : y \mapsto \bigwedge_{k=1}^{h_1} a_k \\ \mathcal{J} = \{y > x\} \cup \{y - x_{i_k} \sim_k c_k \mid 1 \leq k \leq h_2\} \end{cases}$$

  où $\mathcal{S}_{\varphi_k}[x_{j_k} \leftarrow y]$ est le système $\mathcal{S}_{\varphi_k}$ dans lequel la variable x_{j_k} a été remplacée par y.

[2]Nous remercions Pierre-Alain Reynier pour la traduction de ce terme.

[3]Comme le système est évalué au temps t, au moins une des variables de la valuation sera envoyée sur t.

Exemple 7.2.5 *Pour la formule* $x_1.\mathbf{F}\,(a \wedge x_2.\mathbf{F}\,(b \wedge x_1 \leq 2))$*, le système construit par la transformation inductive présentée plus haut est le suivant :*

$$\mathcal{S} = \left\{ \begin{array}{l} V\colon \begin{array}{lll} y_1 & \mapsto & a \\ y_2 & \mapsto & b \end{array} \\ \mathcal{J} = \left\{ \begin{array}{l} y_2 - x_1 \leq 2 \\ y_2 > y_1 \\ y_1 > x_1 \end{array} \right\} \end{array} \right.$$

La variable x_1 *représente le temps initial, c'est-à-dire la date à laquelle la formule doit être vérifiée.*

Le lemme technique suivant établit la correction de la construction :

Lemme 7.2.6 $\kappa, t, \alpha[x \mapsto t] \models \varphi \iff \kappa, t, \alpha[x \mapsto t] \models \mathcal{S}_\varphi$.

Preuve. Nous raisonnons par induction sur la structure de φ.

Le cas des atomes et des contraintes d'horloges est évident. Soit φ une formule simple de $\mathsf{TPTL}_{\mathbf{F}}$ de la forme $x.\mathbf{F}\left(\bigwedge_{k=1}^{h_1} a_k \wedge \left(\bigwedge_{k=1}^{h_2} x_{i_k} \sim_k c_k\right) \wedge \bigwedge_{k=1}^{h_3} \varphi_k\right)$ où φ_k est de la forme $x_{j_k}.\mathbf{F}\,\psi_k$.

$$\begin{array}{ll} \kappa, t, \alpha[x \mapsto t] \models \varphi & \\ \stackrel{(1)}{\iff} & \exists\,\beta > t \text{ tel que } \left\{ \begin{array}{l} \kappa, \beta \models \bigwedge_{k=1}^{h_1} a_k \\ \forall\, 1 \leq k \leq h_2,\ \beta - \alpha[x \mapsto t](x_{i_k}) \sim_k c_k \\ \forall\, 1 \leq k \leq h_3,\ (\kappa, \beta, \alpha[x \mapsto t]) \models x_{j_k}.\mathbf{F}\,\psi_k \end{array} \right. \\ \stackrel{(2)}{\iff} & \exists\,\beta > t \text{ tel que } \left\{ \begin{array}{l} \kappa, \beta \models \bigwedge_{k=1}^{h_1} a_k \\ \forall\, 1 \leq k \leq h_2,\ \beta - \alpha[x \mapsto t](x_{i_k}) \sim_k c_k \\ \forall\, 1 \leq k \leq h_3,\ \exists\, v_k\colon Y_{\psi_k} \to [\beta, +\infty) \text{ tel que} \\ \qquad (\kappa, v_k, \alpha[x \mapsto t, x_{j_k} \mapsto \beta]) \vdash \mathcal{S}_{x_{j_k}.\mathbf{F}\,\psi_k} \\ \qquad \text{(par hypothèse d'induction)} \end{array} \right. \\ \stackrel{(3)}{\iff} & \exists\,\beta > t \text{ tel que } \left\{ \begin{array}{l} \kappa, \beta \models \bigwedge_{k=1}^{h_1} a_k \\ \forall\, 1 \leq k \leq h_2,\ \beta - \alpha[x \mapsto t](x_{i_k}) \sim_k c_k \\ \exists\, \overline{v_k}\colon Y_{\psi_k} \cup \{y\} \to [t, +\infty) \text{ tel que } \overline{v_k}(y) = \beta \text{ et} \\ \qquad (\kappa, \overline{v_k}, \alpha[x \mapsto t, x_{j_k} \mapsto \beta]) \vdash \mathcal{S}_{x_{j_k}.\mathbf{F}\,\psi_k}[x_{j_k} \leftarrow y] \\ \qquad \text{(parce que } x_{j_k} \text{ est remplacé par } y \text{ et la va-} \\ \qquad \text{luation } \overline{v_k} \text{ associe à } y \text{ la valeur } \beta\text{, qui est la} \\ \qquad \text{valeur associée à } x_{j_k} \text{ par l'interprétation)} \end{array} \right. \\ \stackrel{(4)}{\iff} & \exists\, v\colon Y \to [t, +\infty) \text{ (étendant chaque } v_k\text{) tel que} \\ & \left\{ \begin{array}{l} \kappa, v(y) \models \bigwedge_{k=1}^{h_1} a_k \\ v(y) - \alpha[x \mapsto t](x) > 0 \\ \forall\, 1 \leq k \leq h_2,\ v(y) - \alpha[x \mapsto t](x_{i_k}) \sim_k c_k \\ \forall\, 1 \leq k \leq h_3, \\ \qquad \kappa, v, \alpha[x \mapsto t, x_{j_k} \mapsto v(y)] \vdash \mathcal{S}_{x_{j_k}.\mathbf{F}\,\psi_k}[x_{j_k} \leftarrow y] \end{array} \right. \\ \stackrel{(5)}{\iff} & \exists\, v\colon Y \to [t, +\infty[,\ \text{tel que} \end{array}$$

$$\begin{cases} \kappa, v(y) \models \bigwedge_{k=1}^{h_1} a_k \\ v(y) - \alpha[x \mapsto t](x) > 0 \\ \forall\, 1 \leq k \leq h_2,\ v(y) - \alpha[x \mapsto t](x_{i_k}) \sim_k c_k \\ \forall\, 1 \leq k \leq h_3,\ (\kappa, v, \alpha[x \mapsto t]) \vdash \mathcal{S}_{x_{j_k}.\mathbf{F}\,\psi_k}[x_{j_k} \leftarrow y] \\ \qquad \text{(parce que l'horloge } x_{j_k} \text{ n'apparaît plus} \\ \qquad \text{dans } \mathcal{S}_{x_{j_k}.\mathbf{F}\,\psi_k}[x_{j_k} \leftarrow y]) \end{cases}$$

$$\overset{(6)}{\Longleftrightarrow} \quad \kappa, t, \alpha[x \mapsto t] \models \mathcal{S}_\varphi$$

Presque toutes les équivalences ci-dessus sont évidentes, à part l'implication réciproque $\overset{(3)}{\Longleftarrow}$. En effet, v_k envoie *a priori* Y_{φ_k} sur $[t, +\infty[$, et pas sur $[\beta, +\infty[$. Pour montrer que $v_k(y') \geq \beta$ pour $y' \in Y_{\varphi_k}$, nous appliquons le lemme facile suivant :

Lemme 7.2.7 *Soit κ, v, α tel que $\kappa, v, \alpha[x \mapsto t] \vdash \mathcal{S}_\varphi$. Alors pour tout $y' \in Y$, $v(y') \geq v(y)$.*

Cela achève la preuve du lemme 7.2.6. □

3. Quelques propriétés des systèmes.
Soit $\mathcal{S}$ un système sur Y et ψ une formule de $\mathsf{MTL}_\mathbf{F}$. Nous disons que $\mathcal{S}$ et ψ sont *équivalents* si pour toute séquence temporisée κ,

$$\kappa, 0, \mathbf{0} \models \mathcal{S} \quad \text{ssi} \quad \kappa, 0 \models \psi.$$

Notre but est donc de construire une formule ψ de $\mathsf{MTL}_\mathbf{F}$ équivalente à $\mathcal{S}_\varphi$, où φ est une formule simple de $\mathsf{TPTL}_\mathbf{F}$.

Nous disons que deux systèmes $\mathcal{S} = (V, \mathcal{J})$ et $\mathcal{S}' = (V', \mathcal{J}')$ sont *équivalents* si $V = V'$, et $\mathcal{J}$ et $\mathcal{J}'$ ont les mêmes solutions. Remarquons que deux systèmes équivalents représentent des formules de $\mathsf{TPTL}_\mathbf{F}$ équivalentes sur les séquences temporisées.

Nous prouvons facilement le lemme suivant :

Lemme 7.2.8 *Soient $\mathcal{S}_1 = (V, \mathcal{J}_1)$ et $\mathcal{S}_2 = (V, \mathcal{J}_2)$ deux systèmes. Soit $\mathcal{S} = (V, \mathcal{J})$ un système tel que l'ensemble des solutions de $\mathcal{J}$ est l'union de l'ensemble des solutions de $\mathcal{J}_1$ et $\mathcal{J}_2$. Alors*

$$\kappa, t, \alpha \models \mathcal{S} \iff \kappa, t, \alpha \models \mathcal{S}_1 \text{ ou } \kappa, t, \alpha \models \mathcal{S}_2.$$

Grâce à ce lemme, nous déduisons la propriété suivante : si φ_i est une formule de $\mathsf{MTL}_\mathbf{F}$ équivalente à un système $\mathcal{S}_i$ (pour $i \in \{1, 2\}$), alors $\varphi_1 \vee \varphi_2$ est une formule de $\mathsf{MTL}_\mathbf{F}$ équivalente à $\mathcal{S}$.

4. Réduction à des systèmes d'inéquation diagonales bornés.
Nous fixons un système $\mathcal{S} = (V, \mathcal{J})$, en supposant que $\mathcal{J} = \{x_i - x_j \prec_{i,j} m_{i,j} \mid i, j = 0 \cdots n\}$ est un ensemble de contraintes en forme normale (c'est-à-dire qu'il n'est pas possible de déduire des contraintes plus fortes) avec $x_0 = 0$. Nous supposons de plus (quitte à dupliquer le système, ajouter des contraintes de la forme $x_i \leq x_j$, et appliquer le lemme 7.2.8) que les contraintes de $\mathcal{J}$ impliquent que $x_{i-1} \leq x_i$ pour tout $0 < i \leq n$. Soit M la plus grande constante apparaissant dans $\mathcal{J}$. Pour tout $\mathsf{b}\colon \{1, \cdots, n\} \to \{\leq, >\}$, nous définissons un nouvel ensemble de contraintes $\mathcal{J}^\mathsf{b}$ où les contraintes $\{x_i - x_{i-1}\ \mathsf{b}(i)\ M \mid 1 \leq i \leq n\}$ sont ajoutées à $\mathcal{J}$. Nous prouvons les deux lemmes suivants :

Lemme 7.2.9 *$(a_i)_{0\leq i\leq n}$ est une solution de $\mathcal{J}$ si, et seulement si c'est une solution de $\mathcal{J}^{\mathsf{b}}$ pour un $\mathsf{b}\colon \{1,\cdots,n\} \to \{\leq,>\}$.*

Lemme 7.2.10 *Soit $\mathsf{b}\colon \{1,\cdots,n\} \to \{\leq,>\}$ tel que $\mathcal{J}^{\mathsf{b}}$ est consistent (c'est-à-dire $\mathcal{J}^{\mathsf{b}}$ a une solution), et soit $\equiv_{\mathsf{b}}$ la relation d'équivalence sur les indices suivante :*

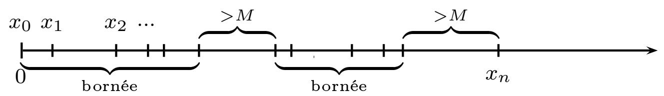
$$i \equiv_{\mathsf{b}} j \quad \textit{ssi} \quad \textit{pour tout } i \leq k < j,\ \mathsf{b}(k) = \leq.$$

Alors $\mathcal{J}^{\mathsf{b}}$ est équivalent à

$$\{x_i - x_j \prec_{i,j} m_{i,j} \mid i \equiv_{\mathsf{b}} j\} \cup \{x_i - x_{i-1}\, \mathsf{b}(i)\, M \mid 1 \leq i \leq n\}.$$

Ces lemmes sont une conséquence immédiate du fait que M est la plus grande constante apparaissant dans $\mathcal{J}$, et du fait que $x_{i-1} \leq x_i$ pour tout $0 < i \leq n$.

Le lemme 7.2.10 est décrit par la figure suivante :



Sur cette figure, tout point sur la ligne représente une variable, et une partie notée « bornée » rassemble toutes les variables dont les différences sont bornées par le système d'inéquations $\mathcal{J}^{\mathsf{b}}$. Deux parties « bornées » sont séparées par au moins M unités de temps.

D'après le lemme 7.2.9, si ψ^{b} est une formule de $\mathsf{MTL}_{\mathbf{F}}$ équivalente à $\mathcal{S}^{\mathsf{b}}$, alors la disjonction de tous les ψ^{b}, où b parcourt l'ensemble des fonctions de $\{1,\cdots,n\} \to \{\leq,>\}$, est équivalente à $\mathcal{S}$. Il nous reste à expliquer comment construire une formule équivalente à un système $\mathcal{S}^{\mathsf{b}}$.

Nous fixons un $\mathsf{b}\colon \{1,\cdots,n\} \to \{\leq,>\}$, et notons $(I_i)_{0\leq i\leq p}$ les classes d'équivalence pour $\equiv_{\mathsf{b}}$ (en ordre croissant). Pour chaque $0 \leq i \leq p$, nous notons n_i le plus grand indice de I_i. Nous supposons que nous connaissons une procédure qui construit une formule de $\mathsf{MTL}_{\mathbf{F}}$ équivalente aux systèmes $\mathcal{S} = (V, \mathcal{J})$ où $\mathcal{J}$ implique que toutes les variables sont bornées. Nous décrirons une telle procédure à l'étape 6. ci-dessous. La formule de $\mathsf{MTL}_{\mathbf{F}}$ résultante est noté $\Psi(\mathcal{S})$. Nous définissons les systèmes $(\mathcal{S}_i)_{0\leq i\leq p}$ par une récurrence décroissante sur i comme suit : $\mathcal{S}_i = (V_i, \mathcal{J}_i)$ est un système sur $\{x_j \mid j \in I_i\}$ et

$$\begin{cases} \begin{cases} V_i(x_j) = V^{\mathsf{b}}(x_j) & \text{si } i = p \text{ et } j \in I_i, \text{ ou si } j \in I_i \setminus \{n_i\} \\ V_i(x_{n_i}) = V^{\mathsf{b}}(x_{n_i}) \wedge \mathbf{F}_{>M}\, \Psi(\mathcal{S}_{i+1}) & \text{si } i \neq p \end{cases} \\ \mathcal{J}_i = \mathcal{J}^{\mathsf{b}}_{|I_i} \text{ est la restriction de } \mathcal{J}^{\mathsf{b}} \text{ aux variables } \{x_j \mid j \in I_i\} \end{cases}$$

D'après les lemmes 7.2.6 et 7.2.10, la formule ψ^{b} est équivalente à la formule $\Psi(\mathcal{S}_0)$ définie plus haut. Nous avons donc réduit notre problème initial à trouver une formule de $\mathsf{MTL}_{\mathbf{F}}$ équivalente à un système $\mathcal{S} = (V, \mathcal{J})$ où les contraintes de $\mathcal{J}$ impliquent que toutes les variables sont bornées.

5. Décomposition de systèmes d'inéquations diagonales bornés.

Nous fixons $\mathcal{S} = (V, \mathcal{J})$. Nous supposons que les variables apparaissant dans $\mathcal{J}$ sont $\{x_i \mid 0 \leq i \leq n\}$, et qu'elles sont bornées par M. Nous nous inspirons de la décomposition en régions pour les automates temporisés [AD94], et nous décomposons $\mathcal{J}$ en systèmes où les contraintes sont des régions. Intuitivement, une région spécifie dans quel intervalle élémentaire (de la forme $]c, c+1[$ ou $[c, c]$ pour $c \leq M$) se trouve la différence $x_i - x_j$. Il est alors suffisant de trouver des formules de $\mathsf{MTL}_{\mathbf{F}}$ pour les systèmes $\mathcal{S}_R = (V_R, \mathcal{J}_R)$ où $\mathcal{J}_R$ représente une région bornée : en effet, si ψ_R est une formule de $\mathsf{MTL}_{\mathbf{F}}$ équivalente au système $\mathcal{S}_R = (V, \mathcal{J}_R)$ où $\mathcal{J}_R$ contient toutes les contraintes de $\mathcal{J}$ et toutes les contraintes définissant la région R (ce qui signifie aussi que $\mathcal{J}_R$ correspond à R car R est soit inclus dans $\mathcal{J}$ ou disjoint de $\mathcal{J}$), alors la formule $\bigvee_{R \subseteq \mathcal{J}} \psi_R$ est équivalente au système $\mathcal{S}$ (en appliquant le lemme 7.2.8).

Une région R peut également être caractérisée par une valeur entière pour chaque variable x_i $(0 \leq i \leq n)$ et par des variables $(X_i)_{0 \leq i \leq p}$ (qui forment une partition de $\{x_i \mid 0 \leq i \leq n\}$) telles que[4]

- $x \in X_0$ si, et seulement si $\langle x \rangle = 0$,
- $x, y \in X_i$ si, et seulement si $\langle x \rangle = \langle y \rangle$,
- $x \in X_i$ et $y \in X_j$ avec $i < j$ impliquent $\langle x \rangle < \langle y \rangle$,

Soit $\mathcal{S}' = (V', \mathcal{J}')$ le système sur $\{X_i \mid 1 \leq i \leq p\}$ (les X_i sont vus comme des variables ici) tel que pour tout $1 \leq i \leq p$, $V'(X_i) = \bigwedge_{x \in X_i} \mathbf{F}_{=\lfloor x \rfloor} V(x)$, et $\mathcal{J}'$ est le système $0 < X_1 < \cdots < X_p < 1$. Si ψ' est une formule de $\mathsf{MTL}_{\mathbf{F}}$ équivalente à $\mathcal{S}'$, alors la formule $\left(\bigwedge_{x \in X_0} \mathbf{F}_{=\lfloor x \rfloor} V(x)\right) \wedge \psi'$ est équivalente au système $\mathcal{S}$.

6. Formules de $\mathsf{MTL}_{\mathbf{F}}$ pour les systèmes simples.

Il nous reste à montrer comment construire des formules de $\mathsf{MTL}_{\mathbf{F}}$ $\Psi_{[1 \cdots p], r}$ équivalentes aux systèmes $\mathcal{S}_{p,r} = (V, \mathcal{J}_{p,r})$ sur $\{X_i \mid 1 \leq i \leq p\}$, où r est un rationnel et $\mathcal{J}_{p,r}$ est l'ensemble des contraintes $0 < X_1 < \cdots < X_p < r$. Remarquons que par abus de notation nous supposons que nous avons une unique fonction V qui est utilisée pour tous les systèmes $\mathcal{S}_{p,r}$. Nous construisons inductivement des formules $\Psi_{[h \cdots h+k], r}$, qui traitent le cas des variables X_h à X_{h+k} sur l'intervalle $]0; r[$. Pour $k < 0$, la formule est $\top$. Pour $k = 0$, nous avons $\Psi_{[h], r} = \mathbf{F}_{<r} V(X_h)$. Pour $k+1$ variables X_h à X_{h+k}, $\Psi_{[h \cdots h+k], r}$ est la disjonction des quatre formules Φ_1 à Φ_4 décrites ci-dessous, qui distinguent les positions possibles des variables :

- s'il n'y a pas de variable dans l'intervalle $\left]0, \frac{r}{k+1}\right]$ et que toutes les variables $(X_{h+i})_{0 \leq i \leq k}$ sont dans l'intervalle $\left]\frac{r}{k+1}, r\right[$:

$$\Phi_1 = \Theta_1 \vee \mathbf{F}_{<\frac{r}{k+1}} \Theta_1$$

où

$$\Theta_1 = \mathbf{F}_{<\frac{r}{k+1}} \left(\bigvee_{i=1}^{k} \left(\left(\mathbf{F}_{=r-\frac{ir}{k+1}} V(X_{h+k}) \right) \wedge \Psi_{[h \cdots h+k-1], r-\frac{ir}{k+1}} \right) \right)$$

La formule Φ_1 distingue les différentes positions pour la dernière variable X_{h+k} : celle-ci est dans l'un des intervalles $\left[r - \frac{ir}{k+1}, r - \frac{ir}{k+1}\right]$ avec $1 \leq i < k$ ou $\left]r - \frac{ir}{k+1}, r - \frac{(i-1)r}{k+1}\right[$ avec $1 \leq i \leq k$.

Remarquons que Φ_1 n'exprime pas exactement la propriété ci-dessus : elle contient d'autres cas, mais implique toujours que $0 < X_1 < \cdots < X_p < r$. Cette remarque s'applique également pour les trois formules suivantes.

[4] Dans la suite, $\langle x \rangle$ représente la partie fractionnaire de x, et $\lfloor x \rfloor$ représente la borne inférieure de l'intervalle dans lequel x se trouve (si x est dans $[c, c]$ ou $]c, c+1[$, alors $\lfloor x \rfloor$ est c).

- s'il y a $1 \leq l \leq k$ variables dans l'intervalle $\left]0, \frac{r}{k+1}\right[$ et $k-l+1$ variables dans l'intervalle $\left]\frac{r}{k+1}, r\right[$:

$$\Phi_2 = \bigvee_{l=1}^{k} \left(\Psi_{[h\cdots h+l-1], \frac{r}{k+1}} \wedge \mathbf{F}_{=\frac{r}{k+1}} \left(\Psi_{[h+l\cdots h+k], r-\frac{r}{k+1}}\right)\right).$$

- s'il y a $0 \leq l \leq k$ variables dans l'intervalle $\left]0, \frac{r}{k+1}\right[$, une variable à la date $\frac{r}{k+1}$, et $k-l$ variables dans l'intervalle $\left]\frac{r}{k+1}, r\right[$:

$$\Phi_3 = \bigvee_{l=0}^{k} \left(\Psi_{[h\cdots h+l-1], \frac{r}{k+1}} \wedge \mathbf{F}_{=\frac{r}{k+1}} \left(V(X_{h+l}) \wedge \Psi_{[h+l+1\cdots h+k], r-\frac{r}{k+1}}\right)\right).$$

- enfin, si toutes les variables sont dans l'intervalle $\left]0, \frac{r}{k+1}\right[$:

$$\Phi_4 = \mathbf{F}_{<\frac{r}{k+1}} \left(V(X_h) \wedge \mathbf{F}_{<\frac{r}{k+1}} \left(V(X_{h+1}) \wedge (\ \cdots\ )\right)\right)$$

Il est facile de montrer par induction que la formule résultante est équivalente à $\mathcal{S}_{p,r}$. □

Exemple 7.2.11 *Nous illustrons toutes les étapes de la construction précédente sur la formule :*

$$\varphi \;=\; x.\mathbf{F}\left(a \wedge x \geq 1 \wedge \mathbf{F}\,(b \wedge x \leq 3) \wedge y.\mathbf{F}\,(\neg a \wedge x \leq 3 \wedge y > 1)\right)$$

1. *La forme normale de φ est*

$$x.\mathbf{F}\left(a \wedge x \geq 1 \wedge z.\mathbf{F}\,(b \wedge x \leq 3) \wedge y.\mathbf{F}\,(\neg a \wedge x \leq 3 \wedge y > 1)\right)$$

2. *Le système associé à cette formule simple de* $\mathsf{TPTL}_{\mathbf{F}}$ *est*

$$\mathcal{J} = \left\{ \begin{array}{ll} z_0 = 0, & z_1 - z_0 \geq 1 \\ z_2 - z_1 > 0, & z_2 - z_0 \leq 3 \\ z_3 - z_1 > 0, & z_3 - z_0 \leq 3 \\ z_3 - z_1 > 1 & \end{array} \right\} \qquad V : \left\{ \begin{array}{l} z_0 \mapsto \top \\ z_1 \mapsto a \\ z_2 \mapsto b \\ z_3 \mapsto \neg a \end{array} \right.$$

- *z_0 représente le temps initial, c'est-à-dire la date à laquelle la formule φ doit être vérifiée ;*
- *z_1 représente la date à laquelle la formule $\left(a \wedge x \geq 1 \wedge z.\mathbf{F}\,(b \wedge x \leq 3) \wedge y.\mathbf{F}\,(\neg a \wedge x \leq 3 \wedge y > 1)\right)$ doit être vérifiée ;*
- *z_2 représente la date à laquelle la formule $(b \wedge x \leq 3)$ doit être vérifiée ;*
- *z_3 représente enfin la date à laquelle la formule $(\neg a \wedge x \leq 3 \wedge y > 1)$ doit être vérifiée.*

De tels systèmes de contraintes peuvent être représentés par des matrices à différences bornées *(DBMs pour* Difference Bound Matrices*) [Bou04]. Le système d'inéquations $\mathcal{J}$ peut être représenté par la DBM suivante en forme normale :*

$$\begin{array}{c} \\ z_0 \\ z_1 \\ z_2 \\ z_3 \end{array} \begin{array}{c} \begin{array}{cccc} z_0 & z_1 & z_2 & z_3 \end{array} \\ \begin{pmatrix} = 0 & < 3 & \leq 3 & \leq 3 \\ \leq -1 & = 0 & \leq 2 & \leq 2 \\ < -1 & < 0 & = 0 & < 2 \\ < -2 & < -1 & < 1 & = 0 \end{pmatrix} \end{array}$$

4. *Le système d'inéquations ci-dessus présente une incertitude sur l'ordre relatif des variables z_2 et z_3. Nous décomposons ce système en deux sous-systèmes, un où $z_2 < z_3$, et un autre où $z_3 \leq z_2$. Nous ne considérons que le second cas dans la suite. Le système obtenu est borné, donc nous n'avons pas besoin de le décomposer plus avant.*
5. *Ce système d'inéquations représente une* zone *bornée, que nous décomposons en régions. Par exemple, il contient la région définie par les contraintes :*

$$\left\{ \begin{array}{cc} z_0 = 0 & 1 < z_1 < 2 \\ 2 < z_2 < 3 & 2 < z_3 < 3 \\ z_2 - z_1 = 1 & 0 < z_3 - z_1 < 1 \\ 0 < z_2 - z_3 < 1 & \end{array} \right\}$$

Nous ne voulons avoir que des constantes 0 *et* 1*, nous décalons donc le système ci-dessus et obtenons :*

$$\mathcal{J}' = \left\{ \begin{array}{cc} z'_0 = 0, & 0 < z'_1 < 1 \\ 0 < z'_2 < 1, & 0 < z'_3 < 1 \\ z'_1 - z'_2 = 0, & 0 < z'_1 - z'_3 < 1 \\ 0 < z'_2 - z'_3 < 1 & \end{array} \right\} \qquad V' : \left\{ \begin{array}{l} z'_0 \mapsto \top \\ z'_1 \mapsto \mathbf{F}_{=1}\, a \\ z'_2 \mapsto \mathbf{F}_{=2}\, b \\ z'_3 \mapsto \mathbf{F}_{=2}\, \neg a \end{array} \right.$$

En posant $X_0 = \{z'_0\}$, $X_1 = \{z'_1, z'_2\}$ et $X_2 = \{z'_3\}$, nous obtenons le nouveau système

$$\mathcal{J}'' = \{0 = X_0 < X_1 < X_2 < 1\} \qquad V'' : \left\{ \begin{array}{l} X_0 \mapsto \top \\ X_1 \mapsto \mathbf{F}_{=1}\, a \wedge \mathbf{F}_{=2}\, b \\ X_2 \mapsto \mathbf{F}_{=2}\, \neg a \end{array} \right.$$

6. *Nous construisons maintenant la formule correspondant au cas que nous avons sélectionné : $\Phi = \Phi_1 \vee \Phi_2 \vee \Phi_3 \vee \Phi_4$ avec*[5]

$$\begin{array}{rl} \Phi_1 = & \Theta_1 \vee \mathbf{F}_{<0.5}\, \Theta_1 \ \textit{où}\ \Theta_1 = \mathbf{F}_{=2.5}\, \neg a \wedge \mathbf{F}_{<0.5}\, (\mathbf{F}_{=1}\, a \wedge \mathbf{F}_{=2}\, b) \\ \Phi_2 = & \mathbf{F}_{<0.5}\, (\mathbf{F}_{=1}\, a \wedge \mathbf{F}_{=2}\, b) \wedge \mathbf{F}_{]2.5,3[}\, \neg a \\ \Phi_3 = & \mathbf{F}_{=0.5}\, (\mathbf{F}_{=1}\, a \wedge \mathbf{F}_{=2}\, b \wedge \mathbf{F}_{]2,2.5[}\, \neg a) \vee (\mathbf{F}_{<0.5}\, (\mathbf{F}_{=1}\, a \wedge \mathbf{F}_{=2}\, b) \wedge \mathbf{F}_{=2.5}\, \neg a) \\ \Phi_4 = & \mathbf{F}_{<0.5}\, (\mathbf{F}_{=1}\, a \wedge \mathbf{F}_{=2}\, b \wedge \mathbf{F}_{]2,2.5[}\, \neg a) \end{array}$$

Remarquons que cette formule n'est qu'une partie de la formule de $\mathsf{MTL}_{\mathbf{F}}$ *équivalente à la formule initiale φ.*

Notre construction de $\mathsf{TPTL}_{\mathbf{F}}$ à $\mathsf{MTL}_{\mathbf{F}}$ est exponentielle. Nous calculons tout d'abord la forme normale de la formule φ de $\mathsf{TPTL}_{\mathbf{F}}$ en choisissant pour chaque disjonction une des sous-formules : la forme normale est alors la disjonction de toutes les formules obtenues par de tels choix. Cela donne un nombre exponentiel de formules dont la disjonction correspond à φ, la taille de chacune de ces formules étant linéaire en la taille de φ. La réduction à des systèmes bornés produit pour chaque formule un nombre exponentiel de systèmes (dont la taille est polynomiale en celle de φ). Nous calculons alors pour chaque système la formule de MTL correspondante qui a une taille exponentielle en la taille du système. La formule de MTL pour φ est finalement la combinaison d'un nombre exponentiel de formules exponentielles, sa taille est donc simplement exponentielle.

[5] Pour améliorer la lisibilité de cet exemple, nous considérons que les opérateurs temporels ont une priorité supérieure à celle de la conjonction.

Notre construction donne une procédure pour tester la satisfaisabilité d'une formule de $\mathsf{TPTL}_{\mathbf{F}}$. La satisfaisabilité de TPTL et MTL est indécidable pour la sémantique continue [AH89], alors que la satisfaisabilité de MTL est décidable mais de complexité non primitive-récursive pour la sémantique d'actions[OW05]. Par la construction ci-dessus nous obtenons que :

Corollaire 7.2.12 *Le problème de satisfaisabilité pour* $\mathsf{TPTL}_{\mathbf{F}}$ *(et donc* $\mathsf{MTL}_{\mathbf{F}}$*) pour la sémantique continue est NP-complet.*

Preuve. Si ψ est une formule de $\mathsf{TPTL}_{\mathbf{F}}$, nous devinons pour chaque disjonction de ψ une des sous-formules, et construisons le système $\mathcal{S} = (V, \mathcal{J})$ pour la nouvelle formule qui est directement en forme normale, nous devinons alors un ordre sur les variables qui est cohérent avec les contraintes de $\mathcal{J}$, et nous résolvons finalement un simple problème de programmation linéaire. Le problème est donc dans NP. La NP-dureté est une conséquence immédiate de celle de 3SAT. $\square$

Chapitre 8

Automates *input-determined* continus

Les automates temporisés sont un modèle expressif ayant un problème de vacuité décidable ; ils ne sont néanmoins pas déterminisables ni clos par complémentaire. De ce fait il est plus difficile d'obtenir des caractérisations logiques pour cette classe de systèmes [Wil94], car par exemple, une caractérisation par une logique monadique du second ordre (MSO) non restreinte suppose la clôture par complémentaire. Les automates *event-clock* [AFH94] furent un des premiers pas vers l'identification d'une sous-classe des automates temporisés clos pour les opérations booléennes. Ces automates ont une caractérisation logique forte en terme de logique monadique du second ordre [RS97, D'S03]. Les automates *event-clock* ont des horloges implicites, dont la valeur ne dépend que du mot temporisé lu et pas du chemin pris dans l'automate.

Pour la sémantique d'actions, une classe d'automates (les automates *input-determined*) généralisant les automates *event-clock* a été proposée [DT04], permettant d'obtenir de fortes caractérisations logiques entre logiques temporisées et automates *input-determined*. Des exemples d'opérateurs *input-determined* sont l'opérateur $\lhd_a$ de [AFH96] qui mesure le temps depuis la dernière occurrence d'une action a, ou l'opérateur $\Diamond_a$ [DT04, DM05] rappelant l'opérateur $\mathbf{F}$ de MTL, qui mesure le temps jusqu'aux prochaines occurrences de l'action a. Nous établissons dans ce chapitre un cadre similaire pour la sémantique continue, nous ajoutons des invariants et des ε-transitions aux automates *input-determined* classiques. Nous montrons que cette classe d'automates est déterminisable et close pour les opérations booléennes. Tout comme les automates finis, les automates *input-determined* continus admettent une caractérisation par une logique monadique du second ordre (TMSO^c) ; de plus le fragment du premier ordre de cette logique (TFO^c) est expressivement équivalent à une extension *input-determined* naturelle de LTL (TLTL^c).

La récursivité apparaît de manière naturelle dans les logiques temporisées dans la construction inductive des formules ; pour obtenir des caractérisations entre automates et logiques temporisées, nous définissons une version *récursive* des automates *input-determined*. Comme dans le cadre non-récursif, nous montrons que les automates *input-determined* récursifs admettent une caractérisation par une logique monadique du second ordre ($\mathsf{rec\text{-}TMSO}^c$), avec une équivalence entre le fragment du premier ordre ($\mathsf{rec\text{-}TFO}^c$) et une extension *input-determined* naturelle de LTL ($\mathsf{rec\text{-}TLTL}^c$). La logique $\mathsf{rec\text{-}TLTL}^c$ correspond à la logique MTL+Past, qui est donc expressivement équivalente à $\mathsf{rec\text{-}TFO}^c$. Nous avons donc développé un formalisme logique pour MTL+Past : les automates *input-determined* récursifs caractérisent $\mathsf{rec\text{-}TMSO}^c$, et MTL+Past correspond au fragment du premier ordre de cette logique. Nous donnerons d'autres applications de ces résultats dans le chapitre 9.

8.1 Définitions

8.1.1 Fonctions à variation finie

Soit Σ un alphabet et soit $f : [0, r] \to \Sigma$ une fonction avec $r \in \mathbb{T}$. Nous notons $dur(f)$ la durée de f, dans ce cas r. Nous disons que f est une fonction *à variation finie*[1] sur Σ s'il existe une suite d'intervalles $I_0, I_1, \cdots, I_{2n}$ et des lettre $w_0, w_1, \cdots, w_{2n}$ tels que $0 \in I_0$, I_i et I_{i+1} sont adjacents pour tout i, I_i est un singleton pour tout i pair, et pour tout $t \in [0, r]$, $f(t) = w_i$ si $t \in I_i$. Nous disons alors que $(w_0, I_0) \cdots (w_{2n}, I_{2n})$ est une *représentation par intervalles* de f. Un mot $w_0 w_1 \cdots w_n$ de Σ^* est dit *canonique* si n est pair, et s'il n'existe pas de i pair tel que $0 < i < n$ et $w_{i-1} = w_i = w_{i+1}$. Une représentation par intervalles $(w_0, I_0) \cdots (w_{2n}, I_{2n})$ de f est dite *canonique* si $w_0 \cdots w_{2n}$ est canonique. Remarquons que toute fonction à variation finie a une unique représentation canonique par intervalles. Nous appelons $FVF(\Sigma)$ l'ensemble des fonctions à variation finie sur Σ.

Soit $f \in FVF(\Sigma)$ et soit $(w_0, I_0) \cdots (w_{2n}, I_{2n})$ sa représentation canonique par intervalles. La *détemporisation* de f, notée *détemp*(f), est $w_0 w_1 \cdots w_{2n}$. La détemporisation d'une fonction capture explicitement les points de discontinuités de cette fonction et les intervalles entre eux : si $F \subseteq FVF(\Sigma)$, *détemp*$(F) = \{$*détemp*$(f) \mid f \in F\}$. La *temporisation* d'un mot $w = w_0 w_1 \cdots w_n$ sur Σ, notée $temp_\Sigma(w)$ (ou simplement $temp(w)$ quand Σ est clair dans le contexte) est définie comme suit :

$$\begin{array}{ll} \text{si } |w| \text{ est impair,} & temp(w) = \emptyset \\ \text{si } |w| \text{ est pair,} & temp(w) = \{f \in FVF(\Sigma) \mid (w_0, I_0)(w_1, I_1) \cdots (w_n, I_n) \\ & \qquad\qquad \text{est une représentation par intervalles de } f\} \end{array}$$

Si $L \subseteq \Sigma^*$, $temp(L) = \{temp(w) \mid w \in L\}$.

Exemple 8.1.1 *Soit $\Sigma = \{a, b\}$ et soit $f : [0, 2] \to \Sigma$ définie par :*

$$f(x) = \begin{cases} a & si \quad 0 \leq x < 1.3 \\ b & si \quad 1.3 \leq x < 2 \\ a & si \quad x = 2 \end{cases}$$

f est une fonction à variation finie représentée sur la figure suivante :

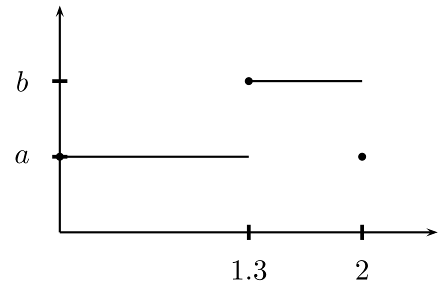


FIG. *8.1 – Fonction à variation finie*

[1]Nous omettons Σ quand l'alphabet est clair dans le contexte.

Soit $w = aaaabba \in \Sigma^*$. La fonction f est dans $temp(w)$ car une représentation par intervalles de f est $(a,[0;0])(a,]0;0,5[)(a,[0,5;0,5])(a,]0,5;1,3[)(b,[1,3;1,3])(b,]1,3;2[)(a,[2;2])$. Notons que cette représentation par intervalles n'est pas canonique (de façon intuitive, cela est dû à l'introduction «arbitraire» du point $0,5$).

Nous avons que $détemp(f) = aabba$ car f admet pour représentation canonique par intervalles $(a,[0;0])(a,]0;1,3[)(b,[1,3;1,3])(b,]1,3;2[)(a,[2;2])$.

8.1.2 Automates *input-determined* continus

Nous définissons tout d'abord les opérateurs *input-determined* et les gardes sur ces opérateurs.

Définition 8.1.2 *Soit Σ un alphabet. Un* opérateur *input-determined sur Σ est une application partielle de $(T\Sigma^* \times \mathbb{T})$ dans $2^{\mathbb{T}}$ qui est définie pour tout couple (σ, t) où $t \in [0, dur(\sigma)]$.*

Un opérateur *input-determined* identifie donc un ensemble de distances à partir d'une position dans un mot temporisé.

Étant donné un ensemble d'opérateurs *input-determined* Op, nous définissons l'ensemble des *gardes sur Op*, noté $\mathcal{G}(Op)$ inductivement de la manière suivante :

$$g ::= \top \mid \Delta^I \mid \neg g \mid g \vee g \mid g \wedge g$$

où $\Delta \in Op$ et $I \in \mathcal{I}_{\mathbb{Q}}$. Les gardes de la forme Δ^I sont dites *atomiques*. Étant donné un mot temporisé σ, nous définissons la satisfaisabilité d'une garde g aux dates $t \in [0, dur(\sigma)]$, et définissons la relation de satisfaction $\sigma, t \models g$ de la manière suivante :

$$\begin{array}{lll}
\sigma, t \models \top & & \\
\sigma, t \models \Delta^I & \text{si} & \Delta(\sigma, t) \cap I \neq \emptyset \\
\sigma, t \models \neg g & \text{si} & \sigma, t \not\models g \\
\sigma, t \models g_1 \vee g_2 & \text{si} & \sigma, t \models g_1 \text{ ou } \sigma, t \models g_2 \\
\sigma, t \models g_1 \wedge g_2 & \text{si} & \sigma, t \models g_1 \text{ et } \sigma, t \models g_2
\end{array}$$

Exemple 8.1.3 *Soit Σ un alphabet. L'opérateur $\Delta_{\mathbb{Q}}$ qui associe à (σ, t) l'ensemble $\{1\}$ si t est rationnel et l'ensemble $\{0\}$ sinon, est un opérateur* input-determined.

Pour $a \in \Sigma$, l'opérateur $\Diamond_a$, inspiré de la logique MTL*, qui associe à (σ, t) l'ensemble des distances d telles que un a se produit au temps $t+d$ dans σ est un opérateur* input-determined. *$\Diamond_a$ est formellement défini par $\Diamond_a(\sigma, t) = \{d \in \mathbb{T} \mid \exists 0 \leq i \leq |\sigma| - 1,\ \sigma_i = (a, t+d)\}$.*

L'équivalent dans le passé de $\Diamond_a$, noté $\overleftarrow{\Diamond}_a$, est également un opérateur input-determined. *Il est défini par $\overleftarrow{\Diamond}_a(\sigma, t) = \{d \in \mathbb{T} \mid \exists 0 \leq i \leq |\sigma| - 1\ \sigma_i = (a, t-d)\}$*

Pour $a \in \Sigma$, l'opérateur $\lhd_a$ input-recording*, qui associe à (σ, t) la distance de la dernière occurrence de a avant le temps t, est un opérateur* input-determined. *Il est défini par $\lhd_a(\sigma, t) = \max(\overleftarrow{\Diamond}_a(\sigma, t))$ si $\overleftarrow{\Diamond}_a(\sigma, t) \neq \emptyset$ et par $\lhd_a(\sigma, t) = \emptyset$ sinon.*

Un opérateur *input-determined* Δ sur Σ est dit *à variation finie* si pour tout $I \in \mathcal{I}_{\mathbb{Q}}$, et tout $\sigma \in T\Sigma^*$, la fonction $f_{\Delta^I} : [0, dur(\sigma)] \rightarrow \{0, 1\}$ définie par $f_{\Delta^I}(t) = 1$ si $\sigma, t \models \Delta^I$ et 0 sinon, est à variation finie.

Exemple 8.1.4 *L'opérateur $\Delta_{\mathbb{Q}}$ n'est pas à variation finie. Les opérateurs $\Diamond_a$, $\overleftarrow{\Diamond}_a$ et $\lhd_a$ sont à variation finie.*

Nous définissons maintenant la notion *d'alphabet symbolique* sur un ensemble d'opérateurs *input-determined* :

Définition 8.1.5 *Soit* Σ *un alphabet et* Op *un ensemble d'opérateurs* input-determined *sur* Σ*. Un* alphabet symbolique *sur* (Σ, Op) *est un alphabet partitionné* $\Gamma = \Gamma_1 \cup \Gamma_2$*, où* Γ_1 *est un sous-ensemble fini de* $(\Sigma \cup \{\varepsilon\}) \times \mathcal{G}(Op)$ *et* Γ_2 *un sous-ensemble fini de* $\mathcal{G}(Op)$*.*

Nous définissons l'ensemble des mots temporisés sur Σ associés à une fonction f de $FVF(\Gamma_1 \cup \Gamma_2)$, noté $tw(f)$ de la manière suivante. Si $détemp(f) \notin \Gamma_1 \cdot (\Gamma_2 \cdot \Gamma_1)^*$, alors $tw(f) = \emptyset$. Sinon un mot temporisé $\sigma = (w_0, t_0) \cdots (w_n, t_n)$ est dans $tw(f)$ si pour tout $t \in [0, dur(f)]$:

- si $f(t) = (a, g)$, pour $a \in \Sigma$ et $g \in \mathcal{G}(Op)$, alors il existe $i \in \{0, \cdots, n\}$, avec $t_i = t$ et $a_i = a$ et $\sigma, t \models g$.
- si $f(t) = (\varepsilon, g)$ ou g, pour $g \in \mathcal{G}(Op)$, alors $\sigma, t \models g$ et il n'existe pas de $i \in \{0, \cdots, n\}$ avec $t_i = t$.

Notons que pour tout f, $tw(f)$ est soit vide, soit un singleton. Si $E \subseteq FVF(\Gamma_1 \cup \Gamma_2)$, $tw(f)$ est défini par $\bigcup_{f \in E} tw(f)$.

Définition 8.1.6 *Soit* Σ *un alphabet et* Op *un ensemble d'opérateurs* input-determined *sur* Σ*. Un* automate *input-determined* continu $(AIDC)$ $\mathcal{A}$ *sur* (Σ, Op) *est un automate fini avec invariants sur un alphabet symbolique* $\Gamma = \Gamma_1 \cup \Gamma_2$ *basé sur* (Σ, Op)*.*

En tant qu'automate fini avec invariants, $\mathcal{A}$ accepte un langage symbolique $L^*_{symb}(\mathcal{A})$. Nous définissons également le *langage de fonctions* accepté par $\mathcal{A}$, noté $F(\mathcal{A})$ par $temp(L^*_{symb}(\mathcal{A}))$. Le langage temporisé accepté par $\mathcal{A}$ est $tw(F(\mathcal{A}))$, nous le notons $L^*(\mathcal{A})$.

Exemple 8.1.7 *La figure 8.2 représente un AIDC sur* $(\{a, b\}, Op)$ *avec* $Op = \{\Diamond_a \mid a \in \Sigma\}$*. Il reconnaît le langage* L_{si} *(« si » signifie* sans insertion*), qui accepte les mots temporisés pour lesquels pour tout instant entre deux a consécutifs, il n'y a pas de a ou de b une unité de temps plus tard. L'automate fonctionne de la façon suivante : si la dernière lettre lue est un b il se trouve dans l'état initial ; si la dernière lettre lue est un a et la prochaine à être lue un b, il se trouve dans l'état du bas ; enfin si la dernière lettre lue est un a et la prochaine à être lue un a, il se trouve dans l'état de droite. L'invariant de ce dernier état est* $\neg(\Diamond_a^{[1,1]} \vee \Diamond_b^{[1,1]})$ *et assure donc qu'il n'y a pas de a ou de b une unité de temps plus tard.*

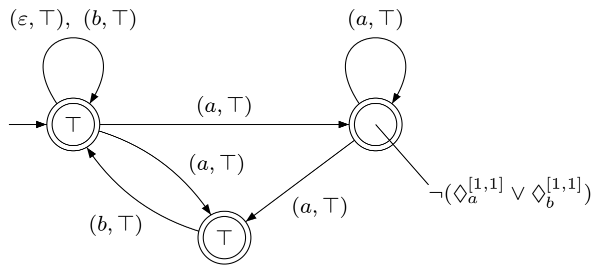


FIG. *8.2 – Exemple d'automate* input-determined *continu*

8.2 Propriétés de clôture

Dans cette section nous montrons que les automates *input-determined* continus sont clos pour l'union, l'intersection et le complémentaire. S'il est clair que leurs langages symboliques sont clos pour les opérations booléennes, ce n'est pas le cas pour leurs langages temporisés : en effet deux mots symboliques différents peuvent avoir des mots temporisés en commun. Complémenter de façon symbolique un automate *input-determined* continu ne permet donc pas d'obtenir le complémentaire de son langage temporisé.

Nous définissons la notion *d'alphabet symbolique propre* : à deux mots d'un tel alphabet correspondent des ensembles de mots temporisés distincts, nous pourrons donc complémenter un automate sur un alphabet symbolique propre en complémentant son langage symbolique.

Soit G un ensemble fini de gardes atomiques sur Op. Un *alphabet symbolique propre* sur (Σ, Op) basé sur G est un alphabet partitionné $\Gamma = \Gamma_1 \cup \Gamma_2$ où $\Gamma_1 = (\Sigma \cup \{\varepsilon\}) \times 2^G$ et $\Gamma_2 = 2^G$. Un *mot symbolique propre* est un mot sur un alphabet symbolique propre. Un mot symbolique propre $\gamma \in \Gamma$ est *canonique pur* si $\gamma \in \Gamma_1 \cdot (\Gamma_2 \cdot \Gamma_1)^*$ et γ ne contient pas de facteur de la forme $g \cdot (\varepsilon, g) \cdot g$. Pour $f \in FVF(\Gamma_1 \cup \Gamma_2)$, nous définissons $tw(f)$ en interprétant $g \subseteq G$ comme la garde $\bigwedge_{h \in g} h \wedge \bigwedge_{h \in G \setminus \{g\}} \neg h$. La propriété fondamentale des mots canoniques purs est que deux mots distincts ont des mots temporisés disjoints.

Exemple 8.2.1 *Soit $\Sigma = \{a\}$, $Op = \{\Diamond_a\}$ et $G = \{\Diamond_a^{[1,1]}\}$. L'alphabet symbolique propre est $\Gamma = \Gamma_1 \cup \Gamma_2$ où $\Gamma_1 = (\Sigma \cup \{\varepsilon\}) \times 2^G$ et $\Gamma_2 = 2^G$. Soit $f_1 : [0,2] \to \Gamma_1 \cup \Gamma_2$ telle que $f_1(0) = f_1(2) = (a, \emptyset), f_1(1) = (\varepsilon, \{\Diamond_a^{[1,1]}\})$ et $f_1(t) = \emptyset$ si $t \notin \{0,1,2\}$. Alors $tw(f_1) = \{(a,0)(a,2)\}$.*

Soit $f_2 : [0,2] \to \Gamma_1 \cup \Gamma_2$ définie par $f_2(1) = (\varepsilon, \emptyset)$ et $f_2(t) = f_1(t)$ si $t \neq 1$. Alors $tw(f_2) = \emptyset$. En effet la lettre $(\varepsilon, \emptyset)$ lue en 1 dans f_2 impose qu'aucune contrainte de G ne doit être vérifiée en ce point, en particulier la garde $\Diamond_a^{[1,1]}$.

Un *automate* input-determined *continu propre* $\mathcal{A}$ sur (Σ, Op) est un automate fini avec invariants sur un alphabet symbolique propre Γ basé sur (Σ, Op). Le langage d'un automate *input-determined* continu propre est défini comme pour un automate *input-determined* continu. Un *AIDC* propre est dit *canonique pur* si son langage symbolique ne contient que des mots canoniques purs.

Nous montrons maintenant que tout *AIDC* est équivalent à un *AIDC* canonique pur.

Lemme 8.2.2 *Soit Σ un alphabet et Op un ensemble d'opérateurs* input-determined *à variation finie. Les AIDC sur (Σ, Op) et les AIDC propres sur (Σ, Op) définissent la même classe de langages temporisés.*

Preuve. Soit $\mathcal{A}$ un *AIDC* sur un alphabet symbolique propre $\Gamma = \Gamma_1 \cup \Gamma_2$ basé sur un ensemble fini de gardes atomiques G. Nous obtenons un *AIDC* à partir de $\mathcal{A}$ en remplaçant tout $g \in G$ par $\bigwedge_{h \in g} h \wedge \bigwedge_{h \in G \setminus \{g\}} \neg h$.

Réciproquement, soit $\Gamma = \Gamma_1 \cup \Gamma_2$ un alphabet symbolique, et soit G l'ensemble des gardes atomiques utilisées dans Γ. Nous montrons maintenant que pour tout *AIDC* $\mathcal{A} = (\Gamma, Q, i, \delta, F, l)$ il existe un *AIDC* propre $\mathcal{A}'$ tel que $L^*(\mathcal{A}) = L^*(\mathcal{A}')$.

Soit $\Gamma' = \Gamma_1' \cup \Gamma_2'$ l'alphabet symbolique propre sur (Σ, Op) basé sur G. Nous construisons $\mathcal{A}' = (\Gamma, Q', i', \delta', F', l')$ basé sur $\Gamma' = \Gamma_1' \cup \Gamma_2'$ comme suit :

- $Q' = \{i'\} \cup \{(q, g) \mid q \in Q, g \subseteq G, g \Rightarrow l(q)\}$
- $\delta' = \{(i', (c, g_1), (q, g_2)) \mid (i, (c, g), q) \in \delta, g_1 \Rightarrow g\} \cup$
  $\{((p, g_1), (c, g_2), (q, g_3)) \mid (p, (c, g), q) \in \delta, g_2 \Rightarrow g\} \cup$
  $\{((p, g_1), (\varepsilon, g_2), (p, g_3)) \mid g_2 \Rightarrow l(p)\}$
- $F' = \{(f, g) \in Q' \mid f \in F\}$
- $l'(s') = \top$ et $l'((q, g)) = g$.

Cette construction consiste à dupliquer tout état $p \in Q$ en fonction de toutes les gardes symboliques propres qui satisfont $l(p)$. Par exemple si $G = \{g_1, g_2\}$ et $l(p) = g_1$, $\mathcal{A}'$ contiendra les états $(p, \{g_1, g_2\})$ et $(p, \{g_1\})$. L'état $(p, \{g_1, g_2\})$ signifie que $\mathcal{A}$ se trouve dans p et que g_1 et g_2 sont vérifiées ; et l'état $(p, \{g_1\})$ signifie que $\mathcal{A}$ se trouve dans p et que g_1 est vérifiée mais pas g_2 (car la garde symbolique propre $\{g_1\}$ signifie $g_1 \wedge \neg g_2$). Remarquons qu'il peut y avoir une transition spontanée de $(p, \{g_1, g_2\})$ à $(p, \{g_1\})$ si la garde g_1 est vérifiée. □

Lemme 8.2.3 *Soit Σ un alphabet et Op un ensemble d'opérateurs* input-determined *à variation finie. Les AIDC sur (Σ, Op) propres et les AIDC canoniques purs sur (Σ, Op) définissent la même classe de langages temporisés.*

Preuve. Un *AIDC* canonique pur est un *AIDC* propre donc l'implication réciproque est triviale.

Soit $\mathcal{A}$ un *AIDC* propre sur $\Gamma = \Gamma_1 \cup \Gamma_2$ basé sur G. Nous allons construire un *AIDC* canonique pur $\mathcal{A}'$ sur Γ tel que $L^*(\mathcal{A}) = L^*(\mathcal{A}')$.

Soit $\mathcal{A} = (\Gamma, Q, i, \delta, F, l)$. Nous définissons tout d'abord la relation δ_ε^* qui correspond intuitivement à la fermeture transitive par ε-transition pour un automate fini avec invariants, c'est-à-dire qu'il faut que tout au long du chemin le même invariant soit vérifié.

δ_ε^* est la plus petite relation incluse dans $Q \times Q$ telle que

- pour tout $q \in Q$, $(q, q) \in \delta_\varepsilon^*$
- pour tout $q, r, t \in Q$, pour tout $g \in G$ tels que $l(q) = l(r) = l(t) = g$, si $(q, (\varepsilon, g), r) \in \delta$ et $(r, t) \in \delta_\varepsilon^*$ alors $(q, t) \in \delta_\varepsilon^*$

Nous construisons maintenant l'automate $\mathcal{A}_1$ comme la fermeture arrière par δ_ε^*, c'est-à-dire $\mathcal{A}_1 = (\Gamma, Q, i, \delta_1, F_1, l)$ où :

- pour tout $q, r \in Q$, pour tout $\gamma \in \Gamma_2$, $(q, \gamma, r) \in \delta_1$ si, et seulement s'il existe $t \in Q$ tel que $(q, t) \in \delta_\varepsilon^*$ et $(t, \gamma, r) \in \delta$; et si $l(q) \neq l(r)$ alors $\gamma \notin \{\varepsilon\} \times G$
- $F_1 = \{q \in Q \mid \exists r \in F \ (q, r) \in \delta_\varepsilon^*\}$

L'automate $\mathcal{A}'$ est obtenu en retirant toutes les transitions $(q, (\varepsilon, g), r)$ telles que $l(q) = l(r) = g$; cette opération ne change pas le langage temporisé puisque nous avons fermé $\mathcal{A}_1$ par δ_ε^*. Nous avons donc que $\mathcal{A}'$ est canonique pur et reconnaît $L^*(\mathcal{A})$. □

Proposition 8.2.4 *Soit Σ un alphabet et Op un ensemble d'opérateurs* input-determined *à variation finie. Les AIDC sur (Σ, Op) et les AIDC canoniques purs sur (Σ, Op) définissent la même classe de langages temporisés.*

Preuve. C'est une conséquence immédiate des lemmes 8.2.2 et 8.2.3. □

Théorème 8.2.5 *Soit Σ un alphabet et Op un ensemble d'opérateurs* input-determined *à variation finie. La classe des AIDC sur (Σ, Op) est close par union, intersection et complémentaire.*

Preuve. La clôture par union est évidente car il suffit de faire l'union des langages symboliques. Pour la clôture par complémentaire, d'après la proposition 8.2.4 il suffit de montrer que les

AIDC canoniques purs sont clos par complémentaire. Soit $\mathcal{A}$ un *AIDC* canonique pur sur $\Gamma = \Gamma_1 \cup \Gamma_2$. Comme deux mots canoniques purs distincts ont des langages de fonctions disjoints, nous avons que $L^*(\mathcal{A})^c = tw(temp(L^*_{symb}(\mathcal{A})))^c = tw(temp(L^*_{symb}(\mathcal{A})^c))$. Or $L^*_{symb}(\mathcal{A})^c$ est un sous-ensemble régulier de $\Gamma_1 \cdot (\Gamma_2 \cdot \Gamma_1)^*$, il est donc reconnu par un *AIDC* propre.

L'intersection peut être obtenue à partir de l'union et du complémentaire. Il est également possible de construire un automate reconnaissant l'intersection de deux *AIDC* canoniques purs comme à partir de l'automate intersection. $\square$

8.3 Logique monadique du second ordre en temps continu

Dans cette section, nous interprétons la logique monadique du second ordre de Büchi sur des fonctions à variation finie et montrons que la détemporisation d'un langage de fonctions définissable dans la logique est régulier. Ce résultat nous sera utile ultérieurement dans l'étude des *AIDC*.

Nous rappelons que pour un alphabet Σ, la logique monadique du second ordre de Büchi (voir par exemple [GThW02]), que nous noterons $\mathsf{MSO}^c(\Sigma)$ est définie comme suit :

$$\varphi ::= Q_a(x) \mid x \in X \mid x < y \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x\varphi \mid \exists X\varphi$$

où $a \in \Sigma$, x et y sont des variables du premier ordre, et X est une variable du second ordre. Nous utiliserons comme convention que les lettres minuscules correspondent à des variables du premier ordre, et les lettres majuscules à des variables du second ordre.

Nous interprétons une formule de la logique sur une fonction à variation finie $f \in FVF(\Sigma)$, avec une interprétation $\mathbb{I}$ par rapport à f, qui associe pour toute variable du premier ordre x une valeur dans $[0, dur(f)]$, et pour toute variable du second ordre X un sous-ensemble fini de $[0, dur(f)]$. Si A et B sont des sous-ensembles de $\mathbb{T}$, nous utilisons la notation $A \subseteq_{fini} B$ si A est un sous-ensemble fini de B.

Pour une interprétation $\mathbb{I}$, nous utilisons la notation $\mathbb{I}[x \mapsto t]$ pour l'interprétation qui envoie x sur t et est identique à $\mathbb{I}$ pour toutes les autres variables. De même $\mathbb{I}[X \mapsto B]$ est l'interprétation qui envoie X sur B et est identique à $\mathbb{I}$ pour toutes les autres variables.

Nous définissons la sémantique de $\mathsf{MSO}^c(\Sigma)$: soit une formule $\varphi \in \mathsf{MSO}^c(\Sigma)$, $f \in FVF(\Sigma)$ et $\mathbb{I}$ une interprétation par rapport à f des variables libres de φ, la relation $f, \mathbb{I} \models \varphi$ est définie inductivement comme suit :

$$\begin{array}{lll}
f, \mathbb{I} \models \mathrm{Q}_a(x) & \text{si} & f(\mathbb{I}(x)) = a \\
f, \mathbb{I} \models x \in X & \text{si} & \mathbb{I}(x) \in \mathbb{I}(X) \\
f, \mathbb{I} \models x < y & \text{si} & \mathbb{I}(x) < \mathbb{I}(y) \\
f, \mathbb{I} \models \neg\varphi & \text{si} & f, \mathbb{I} \not\models \varphi \\
f, \mathbb{I} \models \varphi_1 \vee \varphi_2 & \text{si} & f, \mathbb{I} \models \varphi_1 \text{ ou } f, \mathbb{I} \models \varphi_2 \\
f, \mathbb{I} \models \exists x\varphi & \text{si} & \exists t \in [0, dur(f)],\ f, \mathbb{I}[x \mapsto t] \models \varphi \\
f, \mathbb{I} \models \exists X\varphi & \text{si} & \exists B \subseteq_{fini} [0, dur(f)],\ f, \mathbb{I}[X \mapsto B] \models \varphi
\end{array}$$

Pour un énoncé, c'est-à-dire une formule sans variable libre, l'interprétation ne joue pas de rôle. Nous définissons donc le langage d'un énoncé φ par $F(\varphi) = \{f \in FVF(\Sigma) \mid f \models \varphi\}$.

Le théorème suivant fait le lien entre les automates finis et MSO^c. Notons que ce résultat a été prouvé en utilisant des techniques différentes par Rabinovich dans [Rab02].

Théorème 8.3.1 *Soit φ un énoncé de $\mathsf{MSO}^c(\Sigma)$, il existe un automate fini $\mathcal{A}_\varphi$ tel que $F(\varphi) = temp(L^*_{symb}(\mathcal{A}_\varphi))$.*

Dans le reste de cette section, nous présentons une preuve de ce théorème. Nous représentons les modèles des formules qui ont des variables libres comme des fonctions sur un alphabet étendu. Nous ordonnons l'ensemble des variables : soient $x_1, x_2, \cdots$ les variables du premier ordre et $X_1, X_2, \cdots$ les variables du second ordre. Soit φ un formule avec des variables libres dans $E = \{x_{i_1}, x_{i_2}, \cdots, x_{i_m}\}$ et $F = \{X_{j_1}, X_{j_2}, \cdots, X_{j_n}\}$ où $i_1 < i_2 < \cdots < i_m$ et $j_1 < j_2 < \cdots < j_n$. Nous représentons une fonction f et une interprétation $\mathbb{I}$ comme une fonction $f^{E,F}_{\mathbb{I}} : [0, dur(f)] \to \Sigma \times \{0,1\}^{m+n}$ définie par $f^{E,F}_{\mathbb{I}}(t) = (f(t), b_1, \cdots, b_m, c_1, \cdots, c_n)$ où $b_k = 1$ si $\mathbb{I}(x_{i_k}) = t$ et $c_k = 1$ si $t \in \mathbb{I}(X_{j_k})$. Une telle fonction est appelée (E,F)-modèle. L'ensemble des (E,F)-modèles de φ, noté $\mathcal{M}^{E,F}(\varphi)$ est l'ensemble des fonctions $f^{E,F}_{\mathbb{I}}$ telles que $f, \mathbb{I} \models \varphi$.

Nous prouvons le lemme suivant qui nous aidera dans la construction de $\mathcal{A}_\varphi$:

Lemme 8.3.2 *Soit Σ un alphabet et φ une formule de $\mathsf{MSO}^c(\Sigma)$ avec des variables libres incluses dans $E = \{x_{i_1}, x_{i_2}, \cdots, x_{i_m}\}$ et $F = \{X_{j_1}, X_{j_2}, \cdots, X_{j_n}\}$. Soient E' et F' des sous-ensembles de variables du premier et du second ordre qui contiennent E et F respectivement. Soit $\mathcal{A}$ un automate fini reconnaissant détemp$(\mathcal{M}^{E,F}(\varphi))$. Alors il existe un automate $\mathcal{A}'$ reconnaissant détemp$(\mathcal{M}^{E',F'}(\varphi))$.*

Remarquons tout d'abord que pour tout E, F il existe un automate fini $\mathcal{A}^{E,F}_{canon}$ qui reconnaît l'ensemble des mots canoniques sur $\Sigma \times \{0,1\}^{|E|+|F|}$ et un automate fini $\mathcal{A}^{E,F}_{valid}$ qui reconnaît l'ensemble des mots sur $\Sigma \times \{0,1\}^{|E|+|F|}$ qui pour toute variable du premier ordre de E ont un unique 1 dans la composante correspondante, et tel que ce 1 apparaisse en position paire.

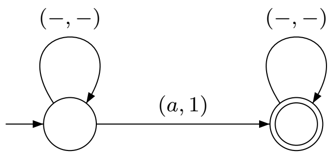
Preuve.[Preuve du lemme] Soit $\mathcal{A}_1$ un automate fini reconnaissant les mots de longueur impaire de $L^*_{symb}(\mathcal{A})$. Soit $\mathcal{A}_2 = (\Sigma, Q_2, I_2, \delta_2, F_2)$ l'automate reconnaissant $\{a_0 b_1^{k_1} a_1 \cdots b_l^{k_l} a_l \mid a_0 b_1 a_1 \cdots b_l a_l \in L^*(\mathcal{A}_1)$ et $\forall\ 1 \leq i \leq l\ k_i$ est impair$\}$. $\mathcal{A}_2$ correspond à *décanoniser* les mots de $\mathcal{A}_1$ (dans le but de pouvoir insérer de nouveaux points où $\mathbb{I}$ vaut 1 quand on construira $\mathcal{A}'$), c'est-à-dire répliquer les parties continues un nombre impair de fois.

Nous construisons ensuite l'automate $\mathcal{A}_3 = (\Sigma \times \{0,1\}^{|E'|+|F'|}, Q_2, I_2, \delta_3, F_2)$ à partir de $\mathcal{A}_2$ en remplaçant chaque transition de $\mathcal{A}_2$ par toutes ses extensions possibles : si $q, q' \in Q_2$, $a \in \Sigma$, $v \in \{0,1\}^{|E'|+|F'|}$, $(q, (a, v), q') \in \delta_3$ si, et seulement si $(q, a, q') \in \delta_2$.

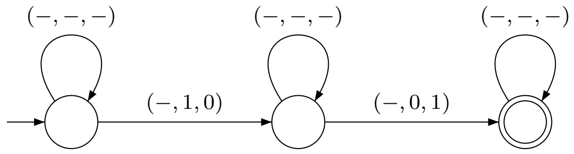
Soit $\mathcal{A}'$ l'intersection de $\mathcal{A}_3$ avec $\mathcal{A}^{E',F'}_{canon}$ et $\mathcal{A}^{E',F'}_{valid}$; nous avons alors que le langage reconnu par $\mathcal{A}'$ est *détemp*$(\mathcal{M}^{E',F'}(\varphi))$. □

Soit φ une formule de $\mathsf{MSO}^c(\Sigma)$, nous construisons inductivement un automate $\mathcal{A}^{E,F}_\varphi$ où E et F sont précisément les variables libres de φ, dont le langage symbolique est *détemp*$(\mathcal{M}^{E,F}(\varphi))$.

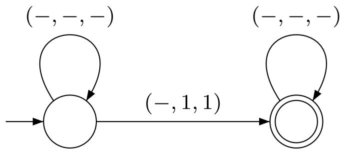
1. $\varphi = Q_a(x)$: l'automate $\mathcal{A}^{\{x\},\emptyset}_\varphi$ est l'intersection de $\mathcal{A}^{\{x\},\emptyset}_{canon}$, $\mathcal{A}^{\{x\},\emptyset}_{valid}$ et l'automate suivant (le symbole $-$ correspondant à n'importe quel symbole pouvant se trouver en cette position) :



2. $\varphi = x < y$: l'automate $\mathcal{A}_{\varphi}^{\{x,y\},\emptyset}$ est l'intersection de $\mathcal{A}_{canon}^{\{x,y\},\emptyset}$, $\mathcal{A}_{valid}^{\{x,y\},\emptyset}$ et l'automate suivant :



3. $\varphi = x \in X$: l'automate $\mathcal{A}_{\varphi}^{\{x\},\{X\}}$ est l'intersection de $\mathcal{A}_{canon}^{\{x\},\{X\}}$, $\mathcal{A}_{valid}^{\{x\},\{X\}}$ et l'automate suivant :



4. $\varphi = \neg\eta$: soit E et F les variables libres de η. $\mathcal{A}_{\varphi}^{E,F}$ est l'intersection de $\mathcal{A}_{canon}^{E,F}$, $\mathcal{A}_{valid}^{E,F}$ et de l'automate complémentaire de $\mathcal{A}_{\eta}^{E,F}$.
5. $\varphi = \eta \vee \psi$: soit (E_η, F_η) les variables libres de η, (E_ψ, F_ψ) les variables libres de ψ. Soit $E = E_\eta \cup E_\psi$ et $F = F_\eta \cup F_\psi$. Par hypothèse d'induction, il existe un automate $\mathcal{A}_{\eta}^{E_\eta,F_\eta}$ reconnaissant $\textit{détemp}(\mathcal{M}^{E_\eta,F_\eta}(\eta))$. Par le lemme 8.3.2, il existe un automate $\mathcal{A}_{\eta}^{E,F}$ reconnaissant $\textit{détemp}(\mathcal{M}^{E,F}(\eta))$. De même pour ψ. $\mathcal{A}_{\varphi}^{E,F}$ est l'union de $\mathcal{A}_{\eta}^{E,F}$ et $\mathcal{A}_{\psi}^{E,F}$.
6. $\varphi = \exists x\eta$: soit (E_η, F_η) les variables libres de η. Soit $E = E_\eta \backslash \{x\}$ et $F = F_\eta$. Par hypothèse d'induction, il existe un automate $\mathcal{A}_{\eta}^{E_\eta,F_\eta}$ pour η.

   Nous projetons tout d'abord la composante en x des étiquettes des transitions de $\mathcal{A}_{\eta}^{E_\eta,F_\eta}$, soit $\mathcal{A}_1$ l'automate obtenu. Soit $\mathcal{A}_2$ l'automate reconnaissant le langage $\{a_0 b_1 a_1 \cdots b_l a_l \mid a_0 b_1^{k_1} a_1 \cdots b_l^{k_l} a_l \in L^*(\mathcal{A}_1)$ et $\forall 1 \leq i \leq l\ k_i$ est impair$\}$ [2]. $\mathcal{A}_{\varphi}^{E,F}$ est l'intersection de $\mathcal{A}_{canon}^{E,F}$, $\mathcal{A}_{valid}^{E,F}$ et $\mathcal{A}_2$.
7. $\varphi = \exists X\eta$: ce cas est similaire au précédent.

Nous pouvons achever la preuve du théorème 8.3.1 : si φ est close nous avons que $F(\varphi) = temp(L_{symb}^*(\mathcal{A}_{\varphi}^{\emptyset,\emptyset}))$.

□

8.4 Une caractérisation logique des *AIDC*

Dans cette section, nous donnons une caractérisation logique des *AIDC* par une logique monadique du second ordre paramétrée par un ensemble d'opérateurs *input-determined*. Soit Σ un alphabet et *Op* un ensemble d'opérateurs *input-determined* sur Σ. Nous définissons la

[2] Cette deuxième étape consiste à *canoniser* le langage de $\mathcal{A}_1$, en effet l'endroit où la variable x était vraie a pû créer une discontinuité qui n'en est plus une après projection. Par exemple si $E_\eta = \{x\}$ et $F_\eta = \emptyset$, le mot $(a,0)(a,1)(a,0)$ sera projeté en aaa, qui n'est pas un mot canonique. L'étape de canonisation consiste à transformer aaa en a.

syntaxe de la *logique monadique du second ordre temporisée* sur (Σ, Op), notée $\mathsf{TMSO}^c(\Sigma, Op)$. Les formules de $\mathsf{TMSO}^c(\Sigma, Op)$ sont définies inductivement comme suit :

$$\varphi ::= Q_a(x) \mid \Delta^I(x) \mid x \in X \mid x < y \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x\varphi \mid \exists X\varphi$$

où $a \in \Sigma$, $\Delta \in Op$, $I \in \mathcal{I}_{\mathbb{Q}}$, x et y sont des variables du premier ordre et X est une variable du second ordre.

Nous interprétons cette logique sur des mots temporisés de $T\Sigma^*$. Soit φ une formule de $\mathsf{TMSO}^c(\Sigma, Op)$, soit $\sigma = (a_1, t_1)\cdots(a_n, t_n)$ un mot temporisé de $T\Sigma^*$, et soit $\mathbb{I}$ une interprétation des variables libres de φ par rapport à σ, qui envoie une variable du premier ordre x sur $t \in [0, dur(\sigma)]$ et une variable du second ordre X sur $B \subseteq_{\mathit{fini}} [0, dur(\sigma)]$, nous définissons la relation $\sigma, \mathbb{I} \models \varphi$ inductivement comme suit :

$$\begin{array}{lll}
\sigma, \mathbb{I} \models \mathrm{Q}_a(x) & \text{si} & \exists 1 \leq i \leq n,\ a_i = a, t_i = \mathbb{I}(x) \\
\sigma, \mathbb{I} \models \Delta^I(x) & \text{si} & \Delta(\sigma, \mathbb{I}(x)) \cap I \neq \emptyset \\
\sigma, \mathbb{I} \models x \in X & \text{si} & \mathbb{I}(x) \in \mathbb{I}(X) \\
\sigma, \mathbb{I} \models x < y & \text{si} & \mathbb{I}(x) < \mathbb{I}(y) \\
\sigma, \mathbb{I} \models \neg\varphi & \text{si} & \sigma, \mathbb{I} \not\models \varphi \\
\sigma, \mathbb{I} \models \varphi_1 \vee \varphi_2 & \text{si} & \sigma, \mathbb{I} \models \varphi_1 \text{ ou } \sigma, \mathbb{I} \models \varphi_2 \\
\sigma, \mathbb{I} \models \exists x\varphi & \text{si} & \exists t \in [0, dur(\sigma)],\ \sigma, \mathbb{I}[x \mapsto t] \models \varphi \\
\sigma, \mathbb{I} \models \exists X\varphi & \text{si} & \exists B \subseteq_{\mathit{fini}} [0, dur(\sigma)] :\ \sigma, \mathbb{I}[X \mapsto B] \models \varphi
\end{array}$$

Pour un énoncé φ de $\mathsf{TMSO}^c(\Sigma, Op)$, nous définissons le langage temporisé de φ comme $L^*(\varphi) = \{\sigma \in T\Sigma^* \mid \sigma \models \varphi\}$. Nous montrons maintenant que TMSO^c caractérise les *AIDC*.

Théorème 8.4.1 *Soit Σ un alphabet et Op un ensemble d'opérateurs* input-determined *à variation finie basés sur Σ. Soit $L \subseteq T\Sigma^*$. L est reconnu par un AIDC sur (Σ, Op) si, et seulement si L est définissable par un énoncé de $\mathsf{TMSO}^c(\Sigma, Op)$.*

Preuve. Nous consacrons le reste de cette section à la preuve de ce théorème.
Implication directe. Nous montrons que tout langage définissable par un *AIDC* propre sur (Σ, Op) est définissable par un énoncé de $\mathsf{TMSO}^c(\Sigma, Op)$.

Soit $\mathcal{A} = (\Gamma, Q, i, \delta, F, l)$ un *AIDC* sur (Σ, Op). Nous construisons une formule $\varphi_{\mathcal{A}}$ telle que $L^*(\mathcal{A}) = L^*(\varphi_{\mathcal{A}})$. Comme dans la preuve du théorème de Büchi, cette formule vérifie l'existence d'une exécution acceptante de $\mathcal{A}$ sur un mot temporisé. Soit $\delta = \{e_1, \cdots, e_m\}$ l'ensemble des transitions de $\mathcal{A}$.

Nous définissons l'ensemble $consec \subseteq \delta \times \delta$, où $(e, e') \in consec$ si, et seulement s'il existe $q \in Q$ tel que $e = (p, \gamma, q)$ et $e' = (q, \gamma', r)$. Nous définissons des abréviations qui nous aiderons dans la construction de la formule $\varphi_{\mathcal{A}}$:

- $\mathit{premier}(x) = \neg\exists y(y < x)$,
- $\mathit{dernier}(x) = \neg\exists y(x < y)$,
- $\mathit{suivant}(x, y, X) = x \in X \wedge y \in X \wedge \neg\exists w(x < w \wedge w < y \wedge w \in X)$, et
- $\mathit{entre}(x, y, z) = x < y \wedge y < z$.

Nous utilisons $\mathit{action}(x)$ pour $\bigvee_{a\in\Sigma} Q_a(x)$. Étant donné une garde g sur Op, nous utiliserons la notation $g(x)$ pour dénoter la formule de TMSO^c obtenue en remplaçant tout Δ^I de g par $\Delta^I(x)$.

Les variables du second ordre $X_{e_1}, \cdots, X_{e_m}$ sont utilisées pour capturer les points du mot temporisé correspondant aux transitions $e_1, \cdots, e_m$ respectivement. X représente l'union de $X_{e_1}, \cdots, X_{e_m}$.

$\varphi_{\mathcal{A}}$ est alors définie par :

$$\exists X \exists X_{e_1} \cdots \exists X_{e_m} (\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4 \wedge \varphi_5 \wedge \varphi_6 \wedge \varphi_7 \wedge \varphi_8) \text{ où :}$$

1. φ_1 signifie que X est l'union de $X_{e_1}, \cdots, X_{e_m}$:

$$\forall x (\bigvee_{e \in \delta} x \in X_e \Leftrightarrow x \in X).$$

2. φ_2 signifie que les ensembles $X_{e_1}, \cdots, X_{e_m}$ sont disjoints :

$$\forall x \bigwedge_{i,j \in \{1,\cdots,m\}, i \neq j} (x \in X_{e_i} \Rightarrow x \notin X_{e_j}).$$

3. φ_3 assure qu'il y a une transition partant de l'état initial au temps 0 :

$$\forall x (premier(x) \Rightarrow \bigvee_{(i,\gamma,q) \in \delta} x \in X_{(i,\gamma,q)}).$$

4. φ_4 signifie que la derniére action du mot temporisé correspond à une transition arrivant dans un état final :

$$\forall x (dernier(x) \Rightarrow \bigvee_{(q,\gamma,f) \in \delta, f \in F} x \in X_{(q,\gamma,f)}).$$

5. φ_5 assure que deux points consécutifs de X appartiennent à deux transitions consécutives :

$$\forall x \forall y (suivant(x, y, X) \Rightarrow \bigvee_{e,e' \in consec} (x \in X_e \wedge y \in X_{e'})).$$

6. φ_6 assure que si un point du mot temporisé correspond à une transition étiquetée (a, g) où $a \in \Sigma$, alors le mot temporisé en ce point contient la lettre a et satisfait la garde g :

$$\forall x \bigwedge_{(p,(a,g),q) \in \delta} (x \in X_{(p,(a,g),q)} \Rightarrow (Q_a(x) \wedge g(x))).$$

7. φ_7 assure que si un point du mot temporisé correspond à une transition étiquetée (ε, g), alors le mot temporisé en ce point ne contient aucune action et satisfait la garde g :

$$\forall x \bigwedge_{(p,(\varepsilon,g),q) \in \delta} (x \in X_{(p,(\varepsilon,g),q)} \Rightarrow (\neg action(x) \wedge g(x))).$$

8. φ_8 signifie que tout point situé entre deux points correspondant à deux transitions consécutives ne contient aucune action et satisfait la garde de l'état correspondant :

$$\forall x \forall y \forall z \big((suivant(y, z, X) \wedge entre(y, x, z)) \Rightarrow$$

$$(\bigwedge_{(p,a,q) \in \delta} (y \in X_{(p,a,q)} \Rightarrow (\neg action(x) \wedge [l(q)](x))))\big).$$

Finalement nous avons que $L^*(\mathcal{A}) = L^*(\varphi_{\mathcal{A}})$.

Implication réciproque. Nous transformons une formule de TMSO^c en une formule de MSO^c et utilisons le théorème 8.3.1 pour obtenir un automate fini sur Γ.

Soit $\sigma = (a_0, t_0)(a_1, t_1) \cdots (a_n, t_n) \in T\Sigma^*$ et G un ensemble fini de gardes atomiques sur Op. Nous définissons la fonction $f_\sigma^G : [0, dur(\sigma)] \to 2^G$ telle que pour tout $t \in [0, dur(\sigma)]$, $f_\sigma^G(t) = \{h \mid h \in G, \sigma, t \models h\}$.

Soit $(g_0, I_0)(g_1, I_1) \cdots (g_{2m}, I_{2m})$ la représentation canonique par intervalles de f_σ^G. Soit $\Gamma = \Gamma_1 \cup \Gamma_2$ l'alphabet symbolique propre sur (Σ, Op) basé sur G. Nous définissons la fonction $f_\sigma^\Gamma : [0, dur(\sigma)] \to \Gamma$ comme suit. Soit $t \in [0, dur(\sigma)]$ et soit $t \in I_j$. Si $t = t_i$ pour $1 \leq i \leq n$, alors $f_\sigma^\Gamma(t) = (a_i, g_j)$, sinon $f_\sigma^\Gamma(t) = (\varepsilon, g_j)$ si j est pair, et $f_\sigma^\Gamma(t) = g_j$ si j est impair. Notons que $\sigma \in tw(f_\sigma^\Gamma)$.

Soit $\varphi \in \mathsf{TMSO}^c(\Sigma, Op)$ et soit $G = \{\Delta^I \mid \Delta^I(x) \text{ est une sous-formule de } \varphi\}$. Soit $\Gamma = \Gamma_1 \cup \Gamma_2$ l'alphabet symbolique propre sur (Σ, Op) basé sur G.

Nous définissons maintenant la transformation *tmso-mso*, qui associe à une formule de $\mathsf{TMSO}^c(\Sigma, Op)$ une formule φ de $\mathsf{MSO}^c(\Gamma)$. *tmso-mso* est définie inductivement comme suit :

$$\begin{array}{lcl}
\textit{tmso-mso}(Q_a(x)) & = & \bigvee_{(a,g)\in\Gamma} Q_{(a,g)}(x) \\
\textit{tmso-mso}(\Delta^I(x)) & = & \bigvee_{(c,g)\in\Gamma, \Delta^I \in g} Q_{(c,g)}(x) \vee \bigvee_{g\in\Gamma, \Delta^I \in g} Q_g(x) \\
\textit{tmso-mso}(x \in X) & = & x \in X \\
\textit{tmso-mso}(x < y) & = & x < y \\
\textit{tmso-mso}(\neg\psi) & = & \neg \textit{tmso-mso}(\psi) \\
\textit{tmso-mso}(\eta \vee \psi) & = & \textit{tmso-mso}(\eta) \vee \textit{tmso-mso}(\psi) \\
\textit{tmso-mso}(\exists x\, \psi) & = & \exists x \ \textit{tmso-mso}(\psi) \\
\textit{tmso-mso}(\exists X\, \psi) & = & \exists X \ \textit{tmso-mso}(\psi)
\end{array}$$

Le lemme suivant se prouve facilement par induction sur la structure de φ :

Lemme 8.4.2 *Soit $\sigma \in T\Sigma^*$ et soit $\mathbb{I}$ une interprétation des variables libres de φ par rapport à σ. Soit $f \in FVF(\Gamma)$ telle que $\sigma \in tw(f)$. Alors $\sigma, \mathbb{I} \models \varphi$ si, et seulement si $f, \mathbb{I} \models \textit{tmso-mso}(\varphi)$.*

Proposition 8.4.3 *Soit φ un énoncé de $\mathsf{TMSO}^c(\Sigma, Op)$. Alors $L^*(\varphi) = tw(F(\textit{tmso-mso}(\varphi)))$.*

Preuve.[Preuve de la proposition] Soit $\textit{tmso-mso}(\varphi) = \tilde{\varphi}$. Supposons que $\sigma \in tw(F(\tilde{\varphi}))$. Alors il existe f telle que $\sigma \in tw(f)$ et $f \in F(\tilde{\varphi})$. Nous avons que $f \models \tilde{\varphi}$, ce qui implique que $\sigma \models \varphi$ par le lemme 8.4.2. Par conséquent $\sigma \in L^*(\varphi)$.

Supposons que $\sigma \in L^*(\varphi)$. $\sigma \models \varphi$, et comme $\sigma \in tw(f_\sigma^\Gamma)$, $f_\sigma^\Gamma \models \tilde{\varphi}$. Donc $f_\sigma^\Gamma \in F(\tilde{\varphi})$. D'où, $\sigma \in tw(F(\tilde{\varphi}))$. □

Nous pouvons maintenant finir la preuve de l'implication réciproque du théorème 8.4.1 en suivant le diagramme ci-dessous :

$$\begin{tikzcd}
\mathsf{TMSO}^c - \varphi \arrow[r, dotted] \arrow[d] & AIDC - \mathcal{A}' \\
\mathsf{MSO}^c - \tilde{\varphi} \arrow[r] & \mathrm{AF} - \mathcal{A}_{\tilde{\varphi}} \arrow[u]
\end{tikzcd}$$

Par la proposition 8.4.3, $L^*(\varphi) = tw(F(\tilde{\varphi}))$, où $\tilde{\varphi} = \textit{tmso-mso}(\varphi)$. Par le théorème 8.3.1 il existe un automate fini $\mathcal{A}_{\tilde{\varphi}}$ tel que $temp(L^*_{symb}(\mathcal{A}_{\tilde{\varphi}})) = F(\tilde{\varphi})$. Nous avons donc $L^*(\varphi) =$

$tw(temp(L^*_{symb}(\mathcal{A}_{\tilde{\varphi}})))$. Nous pouvons supposer que $L^*_{symb}(\mathcal{A}_{\tilde{\varphi}}) \subseteq \Gamma_1 \cdot (\Gamma_2 \cdot \Gamma_1)^*$ car les mots n'appartenant pas à $\Gamma_1 \cdot (\Gamma_2 \cdot \Gamma_1)^*$ ne génèrent aucun mot temporisé. Nous pouvons donc donner un *AIDC* $\mathcal{A}'$ tel que $L^*_{symb}(\mathcal{A}') = L^*_{symb}(\mathcal{A}_{\tilde{\varphi}})$. Nous avons que $L^*(\varphi) = L^*(\mathcal{A}')$.

Ceci achève la preuve du théorème 8.4.1.

$\square$

8.5 Logique de temps linéaire temporisée

La logique LTL est expressivement équivalente à la logique du premier ordre [GPSS80]. Dans cette section nous montrons un résultat analogue pour une logique de temps linéaire basée sur un ensemble d'opérateurs *input-determined*.

Cette logique, appelée $\mathsf{TLTL}^c(\Sigma, Op)$, est paramétrée par un alphabet Σ et un ensemble d'opérateurs *input-determined* Op sur Σ.

Les formules de TLTL^c sont données par :

$$\varphi ::= a \mid \Delta^I \mid \varphi \,\mathbf{U}\, \varphi \mid \varphi \,\mathbf{S}\, \varphi \mid \neg\varphi \mid \varphi \vee \varphi$$

où $a \in \Sigma$, $\Delta \in Op$ et $I \in \mathcal{I}_{\mathbb{Q}}$. Nous interprétons les formules de $\mathsf{TLTL}^c(\Sigma, Op)$ sur les mots temporisés sur Σ. Soit φ une formule de $\mathsf{TLTL}^c(\Sigma, Op)$. Soit $\sigma \in T\Sigma^*$, avec $\sigma = (a_1, t_1) \cdots (a_n, t_n)$, et soit $t \in [0, dur(\sigma)]$. Alors la relation $\sigma, t \models \varphi$ est définie inductivement comme suit :

$$\begin{array}{lll}
\sigma, t \models a & \text{si} & \exists i \in \mathbb{N},\ a_i = a \text{ et } t_i = t \\
\sigma, t \models \Delta^I & \text{si} & \Delta(\sigma, t) \cap I \neq \emptyset \\
\sigma, t \models \varphi \,\mathbf{U}\, \psi & \text{si} & \exists t < t' \leq dur(\sigma), \sigma, t' \models \psi, \text{ et } \forall t < t'' < t', \sigma, t'' \models \varphi \vee \psi \\
\sigma, t \models \varphi \,\mathbf{S}\, \psi & \text{si} & \exists 0 \leq t' < t, \sigma, t' \models \psi, \text{ et } \forall t' < t'' < t, \sigma, t'' \models \varphi \vee \psi \\
\sigma, t \models \neg\varphi & \text{si} & \sigma, t \not\models \varphi \\
\sigma, t \models (\varphi \vee \psi) & \text{si} & \sigma, t \models \varphi \text{ ou } \sigma, t \models \psi.
\end{array}$$

Le langage temporisé défini par une formule φ de $\mathsf{TLTL}^c(\Sigma, Op)$ est donné par $L^*(\varphi) = \{\sigma \in T\Sigma^* \mid \sigma, 0 \models \varphi\}$.

Nous montrons que TLTL^c est expressivement équivalente au fragment du premier ordre de TMSO^c. Nous notons $\mathsf{TFO}^c(\Sigma, Op)$ le fragment du premier ordre de $\mathsf{TMSO}^c(\Sigma, Op)$ (c'est-à-dire le fragment obtenu en retirant la quantification sur les variables du second ordre). Les logiques TLTL^c et TFO^c sont expressivement équivalentes dans le sens suivant :

Théorème 8.5.1 *Soit Σ un alphabet et Op un ensemble d'opérateurs* input-determined *à variation finie sur Σ. Un langage temporisé $L \subseteq T\Sigma^*$ est définissable par une formule de $\mathsf{TLTL}^c(\Sigma, Op)$ si, et seulement s'il est définissable par un énoncé de $\mathsf{TFO}^c(\Sigma, Op)$.*

Preuve.
Implication directe. Soit φ une formule de $\mathsf{TLTL}^c(\Sigma, Op)$, nous pouvons lui associer une formule φ' de $\mathsf{TFO}^c(\Sigma, Op)$ avec une unique variable libre du premier ordre x telle que $\sigma, t \models \varphi$ si, et seulement si $\sigma, [x \mapsto t] \models \varphi'$. La construction de φ' se fait inductivement de la manière suivante :

- pour les formules atomiques a et Δ^I, nous prenons φ' égale à $Q_a(x)$ et $\Delta^I(x)$ respectivement,

– pour l'hypothèse d'induction, en supposant que nous avons déjà traduit φ et ψ en φ' et ψ' respectivement. Nous prenons deux variables fraîches y et z, et nous traduisons $\varphi \,\mathbf{U}\, \psi$ en

$$\exists y(x < y \wedge \psi'[x \mapsto y] \wedge \forall z((x < z \wedge z < y) \Rightarrow (\varphi'[x \mapsto z] \vee \psi'[x \mapsto z]))),$$

– la traduction de $\mathbf{S}$ est similaire.

Nous avons que si φ' est la traduction de φ alors $\sigma, t \models \varphi$ si, et seulement si $\sigma, [x \mapsto t] \models \varphi'$. Par conséquent $\sigma, 0 \models \varphi$ si, et seulement si σ satisfait l'énoncé φ'' donné par $\forall x(premier(x) \Rightarrow \varphi')$. Il vient que $L^*(\varphi) = L^*(\varphi'')$.

Implication réciproque. Pour cette direction nous utilisons le théorème de Kamp pour la logique classique LTL^c [Kam68].

Nous rappelons que la syntaxe de $\mathsf{LTL}^c(\Sigma)$ est donnée par :

$$\varphi ::= a \mid \varphi \,\mathbf{U}\, \varphi \mid \varphi \,\mathbf{S}\, \varphi \mid \neg\varphi \mid \varphi \vee \varphi$$

où $a \in \Sigma$.

Cette logique est interprétée sur les fonctions $f \in FVF(\Sigma)$. Soit $t \in [0, dur(f)]$ et $\varphi \in \mathsf{LTL}^c(\Sigma)$, la relation $f, t \models \varphi$ est définie inductivement comme suit :

$$\begin{array}{lll}
f,t \models a & \text{si} & f(t) = a \\
f,t \models \varphi \,\mathbf{U}\, \psi & \text{si} & \exists t < t' \leq dur(f), f, t' \models \psi, \text{ et } \forall t < t'' < t', f, t'' \models \varphi \vee \psi \\
f,t \models \varphi \,\mathbf{S}\, \psi & \text{si} & \exists 0 \leq t' < t, f, t' \models \psi, \text{ et } \forall t' < t'' < t, f, t'' \models \varphi \vee \psi \\
f,t \models \neg\varphi & \text{si} & f, t \not\models \varphi \\
f,t \models \varphi \vee \psi & \text{si} & f, t \models \varphi \text{ ou } f, t \models \psi
\end{array}$$

Le langage défini par une formule φ de $\mathsf{LTL}^c(\Sigma)$ est $F(\varphi) = \{f \in FVF(\Sigma) \mid f, 0 \models \varphi\}$. Nous notons $\mathsf{FO}^c(\Sigma)$ le fragment du premier ordre de $\mathsf{MSO}^c(\Sigma)$. Alors le résultat de Kamp [Kam68] est le suivant :

Théorème 8.5.2 ([Kam68]) *Soit Σ un alphabet. La logique $\mathsf{LTL}^c(\Sigma)$ est expressivement équivalente à $\mathsf{FO}^c(\Sigma)$.*

Soit φ un énoncé de $\mathsf{TFO}^c(\Sigma, Op)$. Soit $\Gamma = \Gamma_1 \cup \Gamma_2$ un alphabet symbolique propre sur (Σ, Op) basé sur $G = \{\Delta^I \mid \Delta^I(x)$ est une sous-formule de $\varphi\}$. Soit $\sigma \in T\Sigma^*$, la fonction *tmso-mso* transforme une formule de $\mathsf{TFO}^c(\Sigma, Op)$ en une formule de $\mathsf{FO}^c(\Gamma)$. D'après le lemme 8.4.2, nous avons que $f_\sigma^\Gamma \models \textit{tmso-mso}(\varphi)$ si, et seulement si $\sigma \models \varphi$.

Soit $\tilde{\varphi} = \textit{tmso-mso}(\varphi)$. Par le résultat de Kamp, il existe une fonction *fo-ltl* qui transforme une formule de $\mathsf{FO}^c(\Gamma)$ en une formule équivalente de $\mathsf{LTL}^c(\Gamma)$. D'où $F(\tilde{\varphi}) = F(\textit{fo-ltl}(\tilde{\varphi}))$ et donc $f_\sigma^\Gamma \models \tilde{\varphi}$ si, et seulement si $f_\sigma^\Gamma \models \textit{fo-ltl}(\tilde{\varphi})$.

Soit $\psi = \textit{fo-ltl}(\tilde{\varphi})$. Nous définissons la fonction *ltl-tltl* qui transforme une formule de $\mathsf{LTL}^c(\Gamma)$ en une formule de $\mathsf{TLTL}^c(\Sigma, Op)$:

Soit $a \in \Sigma$, $g \in G$ et $(a, g) \in \Gamma$. Soit $\psi_g = \bigwedge_{h \in g} h \wedge \bigwedge_{h \in G \setminus \{g\}} \neg h$. Pour améliorer la lisibilité, nous notons $\hat{\varphi}$ pour *ltl-tltl*(φ).

$$\begin{array}{lll}
\textit{ltl-tltl}((a,g)) & = & a \wedge \psi_g \\
\textit{ltl-tltl}((\varepsilon,g)) & = & (\neg \bigvee_{a\in\Sigma} a) \wedge \psi_g \wedge \neg((\psi_g \,\mathbf{S}\, \psi_g) \wedge (\psi_g \,\mathbf{U}\, \psi_g)) \\
\textit{ltl-tltl}(g) & = & (\neg \bigvee_{a\in\Sigma} a) \wedge \psi_g \wedge (\psi_g \,\mathbf{S}\, \psi_g) \wedge (\psi_g \,\mathbf{U}\, \psi_g) \\
\textit{ltl-tltl}(\psi \,\mathbf{U}\, \eta) & = & \hat{\psi} \,\mathbf{U}\, \hat{\eta} \\
\textit{ltl-tltl}(\psi \,\mathbf{S}\, \eta) & = & \hat{\psi} \,\mathbf{S}\, \hat{\eta} \\
\textit{ltl-tltl}(\neg\psi) & = & \neg\hat{\psi} \\
\textit{ltl-tltl}(\psi \vee \eta) & = & \hat{\psi} \vee \hat{\eta}
\end{array}$$

Le lemme suivant se montre facilement par induction sur la structure de ψ :

Lemme 8.5.3 *Soit ψ une formule de $\mathsf{LTL}^c(\Gamma)$ et $\sigma \in T\Sigma^*$. Alors $f_\sigma^\Gamma, t \models \psi$ si, et seulement si $\sigma, t \models ltl\text{-}tltl(\psi)$.*

Finalement nous avons que :

$$\begin{array}{rcll} \sigma \models \varphi & \Leftrightarrow & f_\sigma^\Gamma \models \mathit{tmso\text{-}mso}(\varphi) & \text{par le lemme 8.4.2} \\ & \Leftrightarrow & f_\sigma^\Gamma \models \mathit{fo\text{-}ltl}(\mathit{tmso\text{-}mso}(\varphi)) & \text{par le théorème de Kamp} \\ & \Leftrightarrow & \sigma \models \mathit{ltl\text{-}tltl}(\mathit{fo\text{-}ltl}(\mathit{tmso\text{-}mso}(\varphi))) & \text{par le lemme 8.5.3} \end{array}$$

Donc $L^*(\varphi) = L^*(\mathit{ltl\text{-}tltl}(\mathit{fo\text{-}ltl}(\mathit{tmso\text{-}mso}(\varphi))))$, ce qui achève la preuve du théorème 8.5.1.
$\square$

8.6 Automates *input-determined* continus récursifs

Nous considérons maintenant des automates *input-determined récursifs* (ou *hiérarchiques*), une idée introduite dans [HRS99], qui permet d'augmenter le pouvoir des logiques temporelles en passant des formules comme argument aux opérateurs, ainsi que de caractériser l'expressivité des logiques temporisées où la récursivité apparaît de manière naturelle.

Mots temporisés pointés

Un *mot temporisé pointé* sur Σ est un couple (σ, t), où $\sigma \in T\Sigma^*$ et $t \in [0, dur(\sigma)]$. Un *langage temporisé pointé* sur Σ est un ensemble de mots temporisés pointés sur Σ.

Un mot temporisé pointé sur Σ peut être vu comme un mot temporisé sur l'alphabet $\Sigma' = (\Sigma \cup \{\varepsilon\}) \times \{0, 1\}$ (la seconde composante valant toujours 0 sauf au temps pointé où elle vaut 1). Si σ' est un mot temporisé sur Σ', nous notons $premier(\sigma')$ le mot temporisé obtenu à partir de σ' en effaçant la composante $\{0, 1\}$ de σ' et les ε restants.

Un mot temporisé σ' sur Σ' est dit *valide* s'il contient exactement un symbole de $(\Sigma \cup \{\varepsilon\}) \times \{1\}$, et tel que son dernier symbole est dans $\Sigma \times \{0, 1\}$. Un tel mot temporisé représente un mot temporisé pointé (σ, t), où t est le temps de l'unique action dont la deuxième composante est un 1. Nous notons $valid(T\Sigma'^*)$ l'ensemble des mots temporisés valides sur Σ'.

Nous notons $fw : T\Sigma'^* \to T\Sigma^*$ l'application partielle définie sur $valid(T\Sigma'^*)$ qui associe à un mot temporisé σ' sur Σ' le mot temporisé pointé (σ, t) correspondant. Nous étendons cette fonction aux langages temporisés sur Σ de manière naturelle.

Opérateurs *input-determined* récursifs

Un *opérateur* input-determined *récursif* Δ sur Σ est une fonction partielle de $(2^{\mathbb{T}} \times T\Sigma^* \times \mathbb{T})$ dans $2^{\mathbb{T}}$, qui est définie pour les triplets (M, σ, t) où $M \subseteq \mathbb{T}$, $\sigma \in T\Sigma^*$, et $t \in [0, dur(\sigma)]$. L'ensemble M sera typiquement donné par un automate pointé ou une formule de logique temporisée ; c'est-à-dire que M sera l'ensemble des positions dans le mot temporisé σ qui sont acceptées par l'automate pointé ou la formule. Nous précisons cette notion plus loin.

Exemple 8.6.1 *L'opérateur $\Diamond$, inspiré de* MTL *est un opérateur* input-determined *récursif; il est défini par*

$$\Diamond(M, \sigma, t) = \{t' - t \mid t' > t \text{ et } t' \in M\}$$

$\Diamond$ identifie l'ensemble des distances au point courant qui sont dans M.

Nous disons qu'un sous-ensemble M de $\mathbb{T}$ est *à variation finie* s'il existe une fonction à variation finie $f : [0, r] \to \{0, 1\}$ telle que $M \subseteq [0, r]$, et $f(t) = 1$ si, et seulement si $t \in M$. Un opérateur *input-determined* récursif Δ est dit *à variation finie* si pour tout ensemble à variation finie M, Δ_M défini par $\Delta_M(\sigma, t) = \Delta(M, \sigma, t)$ est à variation finie (voir la section 8.1).

AIDC pointés

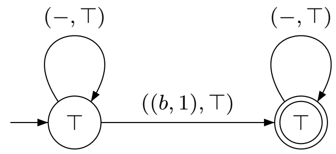
Soit Σ un alphabet et Op un ensemble d'opérateurs *input-determined*. Pour $\Delta \in Op$, nous utilisons la notation Δ' pour l'opérateur sur Σ' avec pour sémantique $\Delta'(\sigma', t) = \Delta(fw(\sigma'))$ si $\sigma' \in valid(T\Sigma'^*)$: l'opérateur Δ' ignore la composante $\{0, 1\}$ de σ' et se comporte comme Δ sur la composante Σ.

Nous notons Op' pour l'ensemble $\{\Delta' \mid \Delta \in Op\}$. Un *AIDC pointé* (*AIDCp*) sur (Σ, Op) est un *AIDC* sur (Σ', Op'). Le *langage pointé* d'un *AIDC* pointé $\mathcal{B}$ est $L^p(\mathcal{B}) = fw(L^*(\mathcal{B}))$.

Remarque 8.6.2 *Un AIDCp se comporte comme un AIDC, les gardes et invariants fonctionnent de la même manière ; la différence est qu'un AIDCp, en plus d'accepter ou non le mot d'entrée, indique une position dans ce mot. L'ensemble de ces positions acceptées sera utilisé pour définir récursivement les AIDC.*

Exemple 8.6.3 *Soit $\Sigma = \{a, b\}$ et $Op = \emptyset$. L'automate $\mathcal{B}$ suivant est un AIDCp sur (Σ, Op). Il reconnaît le langage temporisé $L^*(\mathcal{B}) = \{\sigma' \in T\Sigma'^* \mid \exists i\ \sigma'_i = ((b, 1), t)\}$. Nous serons plus intéressés par son langage temporisé pointé $L^p(\mathcal{B}) = fw(L^*(\mathcal{B})) = \{(\sigma, t) \mid \exists i\ \sigma_i = (b, t)\}$.*

$\mathcal{B}$



Un *AIDCp* $\mathcal{B}$ définit pour un mot temporisé un ensemble de positions par l'intermédiaire de son langage pointé. L'ensemble des positions de σ par rapport à $\mathcal{B}$ est $pos(\sigma, \mathcal{B}) = \{t \in [0, dur(\sigma)] \mid (\sigma, t) \in L^p(\mathcal{B})\}$. Nous associons cet ensemble de positions avec un opérateur *input-determined* récursif pour obtenir un opérateur *input-determined* classique. L'opérateur $\Delta_{\mathcal{B}}$ est défini par :

$$\forall \sigma \in T\Sigma^*\ \forall t \in [0, dur(\sigma)],\ \Delta_{\mathcal{B}}(\sigma, t) = \Delta(pos(\sigma, \mathcal{B}), \sigma, t)$$

Remarque 8.6.4 *Cette construction permet d'interpréter des AIDCp comme des opérateurs. Nous procédons ensuite récursivement en construisant des AIDC et des AIDCp utilisant ces nouveaux opérateurs. En répétant l'opération nous construisons la hiérarchie des automates* input-determined *récursifs.*

Automates *input-determined* récursifs

Nous présentons maintenant les automates *input-determined* continus récursifs et leur équivalent pointé. Ces automates sont définis niveau par niveau : les *AIDC* (resp. *AIDCp*) de niveau 0 sont des *AIDC* (resp. *AIDCp*) n'utilisant pas d'opérateur : toutes leurs gardes valent $\top$. Un ensemble fini de *AIDCp* de niveau i donne lieu comme expliqué plus haut à un ensemble d'opérateurs *input-determined*. Les *AIDC* (resp. *AIDCp*) de niveau $i+1$ sont des *AIDC* (resp. *AIDCp*) sur cet ensemble d'opérateurs (nous imposons aussi qu'ils utilisent un opérateur de niveau i, voir la définition ci-dessous).

Nous définissons les automates *input-determined* continus récursifs (*rec-AIDC*) et les automates *input-determined* continus pointés récursifs (*rec-AIDCp*) sur un alphabet Σ et un ensemble d'opérateurs *input-determined* récursifs *Rop* basé sur Σ, comme étant respectivement l'union des *rec-AIDC* et des *rec-AIDCp* de niveau i, pour $i \in \mathbb{N}$. Les *rec-AIDC* et *rec-AIDCp* de niveau i sont définis inductivement comme suit :

- Un *rec-AIDC* $\mathcal{A}$ de niveau 0 est un *AIDC* sur Σ qui utilise uniquement la garde $\top$. Un *rec-AIDCp* $\mathcal{B}$ de niveau 0 est un *AIDC* pointé sur Σ qui utilise uniquement la garde $\top$.
- Soit C un ensemble fini de *rec-AIDCp* de niveau i ou moins sur (Σ, Rop). Soit *Op* l'ensemble des opérateurs $\{\Delta_{\mathcal{B}} \mid \Delta \in Rop, \mathcal{B} \in C\}$. Nous disons qu'un opérateur $\Delta_{\mathcal{B}}$ est de niveau j si $\mathcal{B}$ est un *rec-AIDCp* de niveau j.
  Un *rec-AIDC*(Σ, Rop) de niveau $i+1$ est un *AIDC*(Σ, Op) qui utilise au moins un opérateur de niveau i. Un *rec-AIDCp*(Σ, Rop) de niveau $i+1$ est un *AIDCp*(Σ, Op) qui utilise au moins un opérateur de niveau i.

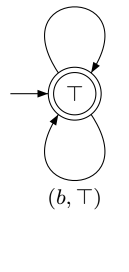
Exemple 8.6.5 *Soit* $\Sigma = \{a, b\}$ *et* $Rop = \{\Diamond\}$. *L'automate* $\mathcal{B}$ *de l'exemple 8.6.3 est un rec-AIDCp sur* (Σ, Rop) *de niveau* 0. *Il définit donc un opérateur* $\Diamond_{\mathcal{B}}$ *donné par* $\Diamond_{\mathcal{B}}(\sigma, t) = \Diamond(pos(\sigma, \mathcal{B}), \sigma, t)$. *Dans cet exemple,* $pos(\sigma, \mathcal{B}) = \{t' \in [0, dur(\sigma)] \mid \exists i,\ \sigma_i = (b, t')\}$. $\Diamond_{\mathcal{B}}$ *identifie les distances au point courant où un* b *se produit :* $\Diamond_{\mathcal{B}}(\sigma, t) = \{t' - t \mid t' > t \text{ et } \exists i,\ \sigma_i = (b, t')\}$.

L'automate $\mathcal{A}$ *suivant est un rec-AIDC sur* (Σ, Rop) *de niveau* 1. *Il reconnaît l'ensemble des mots temporisés tels que toute action* a *est suivie d'une action* b *une unité de temps plus tard. En effet si* $\mathcal{A}$ *accepte un mot temporisé* σ, *toute lecture d'un* a *au temps* t *est soumise à la garde* $\Diamond_{\mathcal{B}}^{[1,1]}$, *et donc nécessairement* $pos(\sigma, \mathcal{B})$ *doit contenir* $t+1$; *comme un* b *se produit pour toute position de* $pos(\sigma, \mathcal{B})$ *nous avons le résultat voulu.*

Plus formellement, si $\mathcal{A}$ *accepte un mot temporisé* σ *et si* $\sigma_i = (a, t)$, *nécessairement* $\sigma, t \models \Diamond_{\mathcal{B}}^{[1,1]}$. *Or nous avons que*

$$\begin{aligned}
\sigma, t \models \Diamond_{\mathcal{B}}^{[1,1]} &\Leftrightarrow \Diamond_{\mathcal{B}}(\sigma, t) \cap [1,1] \neq \emptyset \\
&\Leftrightarrow 1 \in \Diamond(pos(\sigma, \mathcal{B}), \sigma, t) \\
&\Leftrightarrow (\sigma, t+1) \in L^p(\mathcal{B}) \\
&\Leftrightarrow \exists\ 0 \leq i \leq |\sigma| - 1 \text{ tel que } \sigma_i = (b, t+1)
\end{aligned}$$

$\mathcal{A}$



8.7 Caractérisation par MSO des *AIDC* récursifs

Dans cette section nous présentons un version récursive de TMSO^c et montrons qu'elle caractérise la classe des langages temporisés définissables par *rec-AIDC*.

Cette logique est paramétrée par un alphabet Σ et un ensemble d'opérateurs récursifs *Rop*, nous la notons $\mathsf{rec\text{-}TMSO}^c(\Sigma, Rop)$. Soit Σ un alphabet et *Rop* un ensemble d'opérateurs récursifs, l'ensemble des formules de $\mathsf{rec\text{-}TMSO}^c(\Sigma, Rop)$ est défini inductivement comme suit :

$$\varphi ::= Q_a(x) \mid \Delta^I_\psi(x) \mid x < y \mid x \in X \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x \varphi \mid \exists X \varphi$$

où $a \in \Sigma$, $\Delta \in Rop$, $I \in \mathcal{I}_\mathbb{Q}$ et ψ est une formule de $\mathsf{rec\text{-}TMSO}^c(\Sigma, Rop)$ avec une unique variable libre du premier ordre.

La logique $\mathsf{rec\text{-}TMSO}^c$ est interprétée sur les mots temporisés sur Σ. Si φ ne contient pas de prédicat de la forme $\Delta^I_\psi(x)$, alors $\sigma, \mathbb{I} \models \varphi$ est défini comme pour TMSO^c.

Soit ψ une formule de $\mathsf{rec\text{-}TMSO}^c(\Sigma, Rop)$ avec une unique variable libre du premier ordre z, l'opérateur Δ_ψ est défini inductivement. Soit $pos(\sigma, \psi) = \{t \mid \sigma, [z \mapsto t] \models \psi\}$ l'ensemble des interprétations de z qui rendent ψ vraie pour σ. L'opérateur Δ_ψ est défini par $\Delta_\psi(\sigma, t) = \Delta(pos(\sigma, \psi), \sigma, t)$.

Remarquons qu'une formule de $\mathsf{rec\text{-}TMSO}^c(\Sigma, Rop)$ peut-être vue comme une formule de $\mathsf{TMSO}^c(\Sigma, Op)$, où *Op* est l'ensemble des opérateurs Δ_ψ qui ne sont pas sous la portée d'un autre opérateur Δ.

Le *niveau* d'une formule de $\mathsf{rec\text{-}TMSO}^c(\Sigma, Rop)$ est défini naturellement de la manière suivante :

- Une formule de $\mathsf{rec\text{-}TMSO}^c(\Sigma, Rop)$ de niveau 0 est une formule de $\mathsf{TMSO}^c(\Sigma, \emptyset)$.
- Soit C l'ensemble des formules de $\mathsf{rec\text{-}TMSO}^c(\Sigma, Rop)$ de niveau i ou moins avec exactement une variable libre. Alors une formule de $\mathsf{rec\text{-}TMSO}^c(\Sigma, Rop)$ de niveau $i+1$ est une formule de $\mathsf{TMSO}^c(\Sigma, Op)$, où $Op = \{\Delta_\psi \mid \Delta \in Rop, \psi \in C\}$, qui utilise au moins un opérateur de niveau i.

Notons qu'une variable z libre dans ψ n'est pas libre dans $\Delta^I_\psi(x)$. Un énoncé φ de $\mathsf{rec\text{-}TMSO}^c(\Sigma, Rop)$ définit un langage temporisé $L^*(\varphi) = \{\sigma \in T\Sigma^* \mid \sigma \models \varphi\}$. Une formule ψ de $\mathsf{rec\text{-}TMSO}^c(\Sigma, Rop)$ avec une variable libre définit un langage temporisé pointé $L^p(\psi) = \{(\sigma, t) \mid \sigma, [z \mapsto t] \models \psi\}$.

Nous montrons tout d'abord l'équivalence entre les automates pointés et les formules de TMSO^c avec une variable libre.

Lemme 8.7.1 *Soit L un langage temporisé pointé sur un alphabet Σ. Soit Op un ensemble d'opérateurs* input-determined *à variation finie basés sur Σ. Alors il existe un AIDCp $\mathcal{B}$ sur (Σ, Op) tel que $L = L^p(\mathcal{B})$ si, et seulement s'il existe une formule ψ de $\mathsf{TMSO}^c(\Sigma, Op)$ avec une variable libre telle que $L = L^p(\psi)$.*

Preuve.
Implication directe. Soit $\mathcal{B} = (Q, i, \delta, F, l)$ un *AIDCp* sur (Σ, Op) tel que $L = L^p(\mathcal{B})$. Alors $\mathcal{B}$ est un *AIDC* sur (Σ', Op').

Soit $\delta = \{e_1, \cdots, e_m\}$. Nous donnons une formule $\psi_\mathcal{B}$ de $\mathsf{TMSO}^c(\Sigma, Op)$ telle que $L^p(\mathcal{B}) = L^p(\psi_\mathcal{B})$. La construction est similaire à celle du théorème 8.4.1, les changements sont les suivants. $g(x)$ est obtenu à partir de g en remplaçant tout Δ'^I par $\Delta^I(x)$. Nous remplaçons φ_6 et φ_7 par les formules suivantes et ajoutons une formule φ_9.

$$\varphi_6 = \forall x \bigwedge_{(p,((a,i),g),q)\in\delta} (x \in X_{(p,((a,i),g),q)} \Rightarrow (Q_a(x) \wedge g(x)))$$

$$\varphi_7 = \forall x \bigwedge_{(p,(\varepsilon,g),q)\in\delta} (x \in X_{(p,(\varepsilon,g),q)} \Rightarrow (\neg action(x) \wedge g(x)))\wedge$$

$$\forall x \bigwedge_{(p,((\varepsilon,i),g),q)\in\delta} (x \in X_{(p,((\varepsilon,i),g),q)} \Rightarrow (\neg action(x) \wedge g(x)))$$

$$\varphi_9 = \forall x((\bigvee_{(p,((a,1),g),q)\in\delta} x \in X_{(p,((a,1),q),q)}) \Leftrightarrow x = z)$$

Implication réciproque. Soit ψ une formule de $\mathsf{TMSO}^c(\Sigma, Op)$ avec une variable libre z. Par le lemme 8.4.2, nous avons que $\sigma, [z \mapsto t] \models \psi$ si, et seulement si $f, [z \mapsto t] \models \textit{tmso-mso}(\psi)$, où $\sigma \in tw(f)$. Par la construction du théorème 8.3.1, nous obtenons un automate $\mathcal{A}_\psi^{(E,F)}$, où $E = \{z\}$ et $F = \emptyset$, reconnaissant les détemporisations de $f_{[z\mapsto t]}^{E,F}$, où $f, [z \mapsto t] \models \psi$. C'est un automate sur l'alphabet $\Gamma \times \{0, 1\}$, où $\Gamma = \Gamma_1 \cup \Gamma_2$ est l'alphabet symbolique propre basé sur G, les gardes atomiques de ψ. Tout symbole de Γ est de la forme (c, g, i) ou (g, i) où $c \in \Sigma \cup \{\varepsilon\}$, $g \subseteq G$ et $i \in \{0, 1\}$. Remarquons que les mots qui sont la détemporisation d'une fonction $f_{\mathbb{I}}^{(E,F)}$ commencent et finissent par un symbole de la forme (c, g, i) et alternent entre les (c, g, i) et les $(g, 0)$. Nous pouvons donc supposer que $\mathcal{A}_\psi^{(E,F)}$ ne reconnaît que des mots de cette forme. Maintenant nous transformons $\mathcal{A}_\psi^{(E,F)}$ en un *AIDCp* ; nous remplaçons les étiquettes de transition (c, g, i) par $((c, i), g'')$ et (g, i) par g'', où g'' est la garde g où les opérateurs Δ sont remplacés par Δ'. Cet automate reconnaît un sous-ensemble régulier de $(\Sigma' \times 2^G \cup 2^g)^*$; nous pouvons alors construire un *AIDCp* sur (Σ, Op) reconnaissant ce langage symbolique, et dont le langage temporisé pointé est celui de ψ. □

Nous prouvons maintenant que les formules de TMSO^c sont à variation finie. Une formule ψ de $\mathsf{TMSO}^c(\Sigma, Op)$ avec une variable libre est *à variation finie* si pour tout $\sigma \in T\Sigma^*$, $pos(\sigma, \psi)$ est un ensemble à variation finie. Il est facile de voir que si Δ un opérateur *input-determined* récursif à variation finie et ψ une formule à variation finie alors l'opérateur Δ_ψ est à variation finie.

Lemme 8.7.2 *Soit Σ un alphabet et Op un ensemble d'opérateur* input-determined *à variation finie. Toute formule ψ de $\mathsf{TMSO}^c(\Sigma, Op)$ avec une variable libre est à variation finie.*

Preuve. Soit $\sigma \in T\Sigma^*$. Soit $M = pos(\sigma, \psi)$ et $f_M : [0, dur(\sigma)] \to \{0, 1\}$ telle que $f_M(t) = 1$ si, et seulement si $t \in M$. Montrons que f_M est à variation finie. Soit G l'ensemble des gardes atomiques de ψ et soit $\Gamma = \Gamma_1 \cup \Gamma_2$ l'alphabet symbolique propre sur (Σ, Op) basé sur G. Nous montrons que la représentation canonique par intervalles de f_σ^Γ est[3] plus fine que celle de f_M.

Formellement, soit $(w_0, I_0) \cdots (w_{2n}, I_{2n})$ la représentation canonique par intervalles de f_σ^Γ. Montrons que si $t_1, t_2 \in I_i$ alors $t_1 \in pos(\sigma, \psi)$ si, et seulement si $t_2 \in pos(\sigma, \psi)$.

$t_1 \in pos(\sigma, \psi) \Leftrightarrow \sigma, [z \mapsto t_1] \models \psi$ par définition

[3] La définition de f_σ^Γ se trouve page 146

$\Leftrightarrow \quad f^{\Gamma}_{\sigma}, [z \mapsto t_1] \models \mathit{tmso\text{-}mso}(\psi)$ par le lemme 8.4.2
$\Leftrightarrow \quad \mathit{d\acute{e}temp}(f^{\Gamma}_{\sigma\,[z \mapsto t_1]}) \in L^*_{symb}(\mathcal{A}_{tmso\text{-}mso(\psi)})$ par le lemme 8.3.2 [4]
$\Leftrightarrow \quad \mathit{d\acute{e}temp}(f^{\Gamma}_{\sigma\,[z \mapsto t_2]}) \in L^*_{symb}(\mathcal{A}_{tmso\text{-}mso(\psi)})$ car $\mathit{d\acute{e}temp}(f^{\Gamma}_{\sigma\,[z \mapsto t_1]}) = \mathit{d\acute{e}temp}(f^{\Gamma}_{\sigma\,[z \mapsto t_2]})$
$\Leftrightarrow \quad f^{\Gamma}_{\sigma}, [z \mapsto t_2] \models \mathit{tmso\text{-}mso}(\psi)$ par le lemme 8.3.2
$\Leftrightarrow \quad \sigma, [z \mapsto t_2] \models \psi$ par le lemme 8.4.2
$\Leftrightarrow \quad t_2 \in pos(\sigma, \psi)$ par définition

$\square$

Théorème 8.7.3 *Soit Σ un alphabet et soit Rop un ensemble d'opérateurs* input-determined *récursifs à variation finie sur Σ. $L \subseteq T\Sigma^*$ est reconnu par un rec-AIDC sur (Σ, Rop) si, et seulement si L est définissable par un énoncé de* rec-TMSO$^c(\Sigma, Rop)$.

Preuve. Nous montrons par récurrence sur i que pour tout i :

1. $L \subseteq T\Sigma^*$ est reconnu par un *rec-AIDC*(Σ, Rop) de niveau i si, et seulement si L est définissable par un énoncé φ de rec-TMSO$^c(\Sigma, Rop)$ de niveau i.
2. Un langage temporisé pointé L sur Σ est reconnu par un *rec-AIDCp* de niveau i sur (Σ, Rop) si, et seulement si L est définissable par une formule ψ de rec-TMSO$^c(\Sigma, Rop)$ avec une variable libre.
3. Les formules de rec-TMSO$^c(\Sigma, Rop)$ de niveau i avec une variable libre et les automates de *rec-AIDCp*(Σ, Rop) de niveau i sont à variation finie (un automate $\mathcal{B} \in$ *rec-AIDCp*(Σ, Op) est dit *à variation finie* si pour tout $\sigma \in T\Sigma^*$, $pos(\sigma, \mathcal{B})$ est à variation finie).

Les automates et formules de niveau 0 correspondent respectivement aux $AIDC(\Sigma, \emptyset)$ et aux énoncés de TMSO$^c(\Sigma, \emptyset)$, qui ont la même expressivité par le théorème 8.4.1.

De même les *rec-AIDCp*(Σ, Rop) et formules de rec-TMSO$^c(\Sigma, Rop)$ de niveau 0 avec une variable libre correspondent respectivement aux $AIDCp(\Sigma, \emptyset)$ et les formules de TMSO$^c(\Sigma, \emptyset)$ avec une variable libre, qui sont expressivement équivalents par le lemme 8.7.1.

Les formules de rec-TMSO$^c(\Sigma, Rop)$ de niveau 0 avec une variable libre sont à variation finie par le lemme 8.7.2, et donc il en est de même des *rec-AIDCp*(Σ, Rop) de niveau 0.

Nous prouvons maintenant l'étape de récurrence. Soit $\mathcal{A}$ un *rec-AIDC* de niveau $i+1$ sur (Σ, Rop). C'est un $AIDC(\Sigma, Op)$ pour un ensemble d'opérateurs Op de niveau i ou moins, l'un de ces opérateurs étant de niveau i. Les opérateurs de Op sont à variation finie, car ils proviennent d'un opérateur récursif à variation finie et de *rec-AIDCp* de niveau i ou moins, qui sont à variation finie par hypothèse de récurrence. Par le théorème 8.4.1, il existe donc un énoncé de TMSO$^c(\Sigma, Op)$ équivalent à $\mathcal{A}$.

Pour tout $\Delta_{\mathcal{B}} \in Op$, $\mathcal{B}$ est un *rec-AIDC*(Σ, Rop) de niveau j, avec $j < i+1$. Par hypothèse de récurrence, il existe une formule de rec-TMSO$^c(\Sigma, Op)$ de niveau j avec une variable libre telle que $L^p(\psi) = L^p(\mathcal{B})$, donc $pos(\sigma, \psi) = pos(\sigma, \mathcal{B})$. Les opérateurs $\Delta_{\mathcal{B}}$ et Δ_{ψ} ont donc la même sémantique, et nous pouvons remplacer $\Delta_{\mathcal{B}}$ par Δ_{ψ} dans φ pour obtenir un énoncé de rec-TMSO$^c(\Sigma, Op)$ de niveau $i+1$.

Réciproquement, soit φ une formule de rec-TMSO$^c(\Sigma, Rop)$ de niveau $i+1$. C'est une formule de TMSO$^c(\Sigma, Op)$ pour un ensemble d'opérateurs Op de niveau i ou moins. Ces

[4] $f^{\Gamma}_{\sigma\,[z \mapsto t_1]}$ désigne le $(\{z\}, \emptyset)$-modèle associé à f^{Γ}_{σ} et $[z \mapsto t_1]$, voir la preuve du théorème 8.3.1 page 142.

opérateurs sont à variation finie par hypothèse de récurrence. Par le théorème 8.4.1, il existe un $AIDC(\Sigma, Op)$ $\mathcal{A}$ équivalent à φ. Pour tout Δ_ψ de Op, ψ est une formule de rec-TMSO$^c(\Sigma, Op)$ de niveau j avec une variable libre, avec $j < i+1$. Par hypothèse de récurrence, il existe un *rec-AIDCp*(Σ, Rop) $\mathcal{B}$ de niveau j tel que $L^p(\mathcal{B}) = L^p(\psi)$, donc $pos(\sigma, \mathcal{B}) = pos(\sigma, \psi)$. Les opérateurs Δ_ψ et $\Delta_\mathcal{B}$ ont donc la même sémantique, et nous pouvons remplacer Δ_ψ par $\Delta_\mathcal{B}$ dans $\mathcal{A}$ pour obtenir un *rec-AIDC*(Σ, Rop) de niveau $i+1$.

La preuve de l'étape de récurrence pour le deuxième point est similaire et utilise le lemme 8.7.1.

Lors de la preuve du premier point de l'étape de récurrence nous avons montré que les formules de rec-TMSO$^c(\Sigma, Op)$ de niveau $i+1$ et les automates de *rec-AIDC*(Σ, Rop) de niveau $i+1$ sont à variation finie. □

8.8 Caractérisation de rec-TLTL^c par une logique du premier ordre

Nous inspirant de [DT04], nous définissons une logique temporelle de temps linéaire récursive et montrons qu'elle est expressivement équivalente au fragment du premier ordre de rec-TMSO^c. Cette logique est similaire à TLTL^c, elle est paramétrée par un alphabet Σ et un ensemble d'opérateurs *input-determined* récursifs Rop ; nous la notons rec-TLTL$^c(\Sigma, Rop)$. La syntaxe de rec-TLTL$^c(\Sigma, Rop)$ est :

$$\varphi ::= a \mid \Delta^I_\varphi \mid \varphi\,\mathbf{U}\,\varphi \mid \varphi\,\mathbf{S}\,\varphi \mid \neg\varphi \mid \varphi \vee \varphi$$

où $a \in \Sigma$, $\Delta \in Rop$ et $I \in \mathcal{I}_\mathbb{Q}$.

La logique est interprétée sur les mots temporisés sur Σ. Si ψ ne contient pas de prédicat de la forme Δ^I_φ, alors $\sigma, \mathbb{I} \models \psi$ est défini comme pour TLTL^c. Si φ est une formule de rec-TLTL$^c(\Sigma, Rop)$, l'opérateur Δ_φ est défini par $\Delta_\varphi(\sigma, t) = \Delta(pos(\sigma, \varphi), \sigma, t)$ où $pos(\sigma, \varphi) = \{t \in \mathbb{T} \mid \sigma, t \models \varphi\}$. La relation $\sigma, t \models \Delta^I_\varphi$ est alors définie comme pour TLTL^c.

Ici aussi Δ_φ est un opérateur *input-determined*, et toute formule ψ de rec-TLTL$^c(\Sigma, Rop)$ est une formule de TLTL$^c(\Sigma, Op)$, où Op est l'ensemble des opérateurs Δ_φ apparaissant dans ψ, et n'étant pas sous la portée d'un autre opérateur Δ. Le niveau d'une formule de TLTL$^c(\Sigma, Op)$ est défini comme pour TMSO$^c(\Sigma, Op)$. Une formule de rec-TLTL^c définit un langage temporisé $L^*(\varphi) = \{\sigma \in T\Sigma^* \mid \sigma, 0 \models \varphi\}$ et un langage temporisé pointé $L^p(\varphi) = \{(\sigma, t) \mid \sigma, t \models \varphi\}$.

Soit rec-TFO$^c(\Sigma, Rop)$ le fragment du premier ordre de rec-TMSO$^c(\Sigma, Rop)$, nous avons le résultat d'expressivité suivant :

Théorème 8.8.1 *Soit Σ un alphabet et Rop un ensemble d'opérateurs* input-determined *récursifs.* rec-TLTL$^c(\Sigma, Rop)$ *est expressivement équivalente à* rec-TFO$^c(\Sigma, Rop)$.

Preuve. Nous montrons par récurrence sur i que

1. Un langage temporisé sur Σ est définissable par une formule de rec-TLTL$^c(\Sigma, Rop)$ de niveau i si, et seulement s'il est définissable par un énoncé de rec-TFO$^c(\Sigma, Rop)$ de niveau i.
2. Un langage temporisé pointé sur Σ est définissable par une formule de rec-TLTL$^c(\Sigma, Rop)$ de niveau i si, et seulement s'il est définissable par un énoncé de rec-TFO$^c(\Sigma, Rop)$ de niveau i.

3. Les formules de rec-$\mathsf{TLTL}^c(\Sigma, Rop)$ de niveau i et les formules de rec-$\mathsf{TFO}^c(\Sigma, Rop)$ de niveau i avec une variable libre sont à variation finie. Une formule φ de rec-$\mathsf{TLTL}^c(\Sigma, Op)$ est dite *à variation finie* si pour tout $\sigma \in T\Sigma^*$, $pos(\sigma, \varphi)$ est à variation finie.

Nous utilisons le théorème suivant de Kamp :

Théorème 8.8.2 ([Kam68]) *Soit Σ un alphabet. Pour toute formule η de $\mathsf{FO}^c(\Sigma)$ avec une variable libre z, il existe une formule φ de $\mathsf{LTL}^c(\Sigma)$ telle que pour tout $f \in FVF(\Sigma)$ et $t \in [0, dur(f)]$, $f, [z \mapsto t] \models \eta$ si, et seulement si $f, t \models \varphi$.*

La preuve est maintenant similaire à celle du théorème 8.7.3. □

8.9 Caractérisation de MTL+Past par une logique du premier ordre

En appliquant les résultats généraux de ce chapitre, nous montrons que pour la sémantique continue, MTL+Past est expressivement équivalente à rec-TFO^c pour un certain choix d'opérateurs *input-determined* récursifs. Un résultat similaire a été montré dans [DT04] pour la sémantique d'actions.

La logique $\mathsf{MTL{+}Past}(\Sigma)$ correspond aux formules de MTL+Past utilisant des variables propositionnelles appartenant à l'alphabet Σ, nous rappelons sa syntaxe :

$$\varphi ::= a \mid \varphi \wedge \varphi \mid \neg\varphi \mid \varphi\, \mathbf{U}_I\, \varphi \mid \varphi\, \mathbf{S}_I\, \varphi$$

où $a \in \Sigma$ et $I \in \mathcal{I}_{\mathbb{Q}}$.

Nous remarquons tout d'abord que $\mathsf{MTL{+}Past}(\Sigma)$ est expressivement équivalente à sa sous-logique où les modalités $\mathbf{U}_I$ et $\mathbf{S}_I$ sont remplacées par les modalités $\mathbf{U}$, $\mathbf{S}$, $\mathbf{F}_I$ et $\mathbf{F}_I^{-1}$. Nous donnons ci-dessous la traduction[5] de $\mathbf{U}_I$, la traduction de $\mathbf{S}_I$ est similaire :

$$\varphi\, \mathbf{U}_I\, \psi = \begin{cases} (\mathbf{F}_I\, \psi) \wedge (\neg \mathbf{F}_{<a}\, ((\neg\varphi) \wedge \neg\psi)) \wedge \mathbf{F}_{=a}\, (\psi \vee (\varphi \wedge (\varphi\, \mathbf{U}\, \psi))) & \text{si } I = [a, b\rangle, a > 0 \\ (\mathbf{F}_I\, \psi) \wedge (\neg \mathbf{F}_{\leq a}\, ((\neg\varphi) \wedge \neg\psi)) \wedge \mathbf{F}_{=a}\, (\varphi \wedge (\varphi\, \mathbf{U}\, \psi)) & \text{si } I =]a, b\rangle, a > 0 \\ (\mathbf{F}_I\, \psi) \wedge (\varphi\, \mathbf{U}\, \psi) & \text{si } I = \langle 0, b\rangle \end{cases}$$

Considérons la logique rec-$\mathsf{TLTL}^c(\Sigma, \{\diamondsuit, \diamondminus\})$ où la sémantique de $\diamondsuit$ et $\diamondminus$ est la suivante : $\diamondsuit(X, \sigma, t) = \{t' - t \mid t' > t \text{ et } t \in X\}$ et $\diamondminus(X, \sigma, t) = \{t - t' \mid t' < t \text{ et } t \in X\}$. La logique $\mathsf{MTL{+}Past}(\Sigma)$ a clairement la même expressivité que rec-$\mathsf{TLTL}^c(\Sigma, \{\diamondsuit, \diamondminus\})$, car les prédicats $\mathbf{F}_I\, \varphi$ et $\diamondsuit_\varphi^I$ (resp. $\mathbf{F}_I^{-1}\, \varphi$ et $\diamondminus_\varphi^I$) sont équivalents. De plus les opérateurs $\diamondsuit$ et $\diamondminus$ sont à variation finie. Donc par le théorème 8.8.1 nous obtenons le résultat suivant :

Théorème 8.9.1 *Soit Σ un alphabet. La logique $\mathsf{MTL{+}Past}(\Sigma)$ interprétée pour la sémantique continue est expressivement équivalente à rec-$\mathsf{TFO}^c(\Sigma, \{\diamondsuit, \diamondminus\})$.*

[5] Rappelons que dans la sémantique de MTL+Past que nous avons choisie, la formule $\varphi\, \mathbf{U}_I\, \psi$ est équivalente à $(\varphi \vee \psi)\, \mathbf{U}_I\, \psi$.

Chapitre 9

Automates *input-determined* continus sans compteurs

Dans le chapitre précédent, nous avons défini les automates *input-determined* continus et développé un formalisme logique correspondant. Ces automates sont caractérisés par la logique monadique du second ordre TMSO^c, et le fragment du premier ordre de cette logique TFO^c est expressivement équivalent à une extension naturelle de LTL (TLTL^c). Nous avons obtenus des résultats similaires dans le cadre récursif, avec pour application que $\mathsf{MTL+Past}$ est expressivement équivalente à $\mathsf{rec\text{-}TFO}^c$.

Le but de ce chapitre est d'établir une caractérisation similaire à celle de la représentation par automates finis sans compteurs de la logique du premier ordre [Kam68, MP71] dans le cadre des automates *input-determined* continus. Nous définissons les automates *input-determined* continus sans compteurs, qui correspondent aux automates *input-determined* propres (qui sont une version « déterminisée » des automates *input-determined*, voir le chapitre 8) dont l'automate fini sous-jacent est sans compteurs. Nous montrons que ces automates caractérisent TFO^c, et obtenons un formalisme similaire dans le cadre récursif.

Comme application, nous obtenons que $\mathsf{MTL+Past}$ est équivalente aux automates *input-determined* continus récursifs sans compteurs. Cette représentation nous permet de montrer une propriété des langages temporisés définissables dans $\mathsf{MTL+Past}$, nous montrons que MTL est *stable à l'infini*, c'est-à-dire qu'étant donnée une séquence périodique de mots temporisés uv^iw, la valeur de vérité d'une formule de $\mathsf{MTL+Past}$ sur ces modèles finit par se stabiliser. Ce résultat obtenu grâce à notre caractérisation de $\mathsf{MTL+Past}$ en terme d'automates, est une version plus forte d'un résultat de stabilité similaire [PD06], car il est valide même avec les modalités du passé.

9.1 Automates finis sans compteurs

Nous rappelons certains résultats sur les automates finis sans compteurs, notamment la propriété de *stabilité à l'infini* présentée dans la proposition 9.1.1. Nous montrerons dans la section 9.5 que les automates *input-determined* sans compteurs ont un équivalent temporisé de cette propriété.

Soit $\mathcal{A} = (\Sigma, Q, I, \delta, F)$ un automate fini. Un *compteur* dans $\mathcal{A}$ est une suite d'états distincts $q_0, \cdots, q_n$ avec $n \geq 1$ et un mot $u \in \Sigma^*$ tels que $q_0 \in \delta^*(q_n, u)$ et pour tout $0 \leq i \leq n-1$, $q_{i+1} \in \delta^*(q_i, u)$. Un automate fini est dit *sans compteurs* s'il n'a pas de compteur. Un langage régulier est dit *sans compteurs* s'il est reconnu par un automate fini sans compteurs.

Proposition 9.1.1 *Soit L un langage régulier sur un alphabet Σ. Les propositions suivantes sont équivalentes :*

1. *L est sans compteurs.*
2. *$\mathcal{A}_L$ est sans compteurs*[1].
3. *Il n'existe pas de mots u, v, w de Σ^*, tels que les ensembles $\{i \in \mathbb{N} \mid uv^iw \in L\}$ et $\{i \in \mathbb{N} \mid uv^iw \notin L\}$ soient tous deux infinis.*

Preuve. Nous montrons que (2) $\Rightarrow$ (1) $\Rightarrow$ (3) $\Rightarrow$ (2).

(2) $\Rightarrow$ (1) est évident.

(1) $\Rightarrow$ (3) : Soit $\mathcal{A} = (\Sigma, Q, I, \delta, F)$ un automate fini sans compteurs reconnaissant L. Supposons par l'absurde qu'il existe des mots u, v, w de Σ^*, tels que les ensembles $\{i \in \mathbb{N} \mid uv^iw \in L\}$ et $\{i \in \mathbb{N} \mid uv^iw \notin L\}$ soient tous deux infinis.

Il existe $i_0 \geq |Q|$ tel que $uv^{i_0}w \in L$. Considérons une exécution acceptante de $\mathcal{A}$ sur $uv^{i_0}w$. En considérant les états visités après la lecture de chaque v, nous remarquons qu'il existe un état $q \in Q$ tel qu'il existe une exécution de q à q étiquetée v^i pour $i > 0$. Comme $\mathcal{A}$ n'a pas de compteur, il y a nécessairement une exécution de q à q étiquetée v. Donc $uv^iw \in L$ pour tout $i \geq i_0$, ce qui est absurde.

(3) $\Rightarrow$ (2) : Nous raisonnons par contraposition. Supposons que $\mathcal{A}_L = (\Sigma, Q, i, \delta, F)$ a un compteur. Alors il existe $v \in \Sigma^*$ et des états distincts $q_0, \cdots, q_m \in Q$, avec $m \geq 1$, tels qu'il existe une exécution de q_j à q_{j+1} sur v pour tout $0 \leq j < m$ et il existe une exécution de q_m à q_0 sur v. Comme $\mathcal{A}_L$ est minimal, et que tous les états sont dans la même composante fortement connexe, ils sont tous accessibles, en particulier q_0. Soit u un mot tel qu'il existe une exécution de i à q_0 sur u. Comme $q_0 \neq q_1$, il existe $w \in \Sigma^*$ tel que l'état atteint après avoir lu w à partir de q_0 est final, et l'état atteint après avoir lu w à partir de q_1 n'est pas final. Donc $uv^{km+1}w$ est dans L et $uv^{km+2}w$ n'est pas dans L pour tout $k \in \mathbb{N}$. □

Cette caractérisation permet de déduire le corollaire suivant :

Corollaire 9.1.2 *Les langages réguliers sans compteurs sont clos par union, intersection et complémentaire.*

La définition de compteur dans un automate fini avec invariants est similaire à celle pour un automate fini, sauf que l'étiquette d'un chemin contient aussi les invariants, en particulier nous ne comptons pas l'invariant du premier état de l'exécution mais nous comptons celui du dernier.

Proposition 9.1.3 *Soit $\Sigma = \Sigma_1 \cup \Sigma_2$ un alphabet partitionné. Un langage régulier sur Σ inclus dans $\Sigma_1 \cdot (\Sigma_2 \cdot \Sigma_1)^*$ est sans compteurs si, et seulement s'il est reconnu par un automate fini avec invariants sans compteurs sur Σ.*

[1] Nous rappelons que $\mathcal{A}_L$ est l'automate minimal de L.

Preuve. Nous utilisons les constructions de la proposition 2.1.3, celles-ci n'ajoutent pas de compteur et permettent donc de conclure directement. $\square$

9.2 Logique du premier ordre continue

Dans cette section nous étendons le résultat de Mac Naughton et Papert [MP71] au cas continu : nous montrons que les automates finis sans compteurs correspondent à la logique de Büchi du premier ordre interprétée sur les fonctions réelles.

Soit Σ un alphabet, la logique monadique du second ordre de Büchi ($\mathsf{MSO}^c(\Sigma)$) est définie dans la section 8.3. Nous rappelons que $\mathsf{FO}^c(\Sigma)$ désigne le fragment du premier ordre de cette logique.

Nous montrons maintenant que les langages définissables par une formule de $\mathsf{FO}^c(\Sigma)$ correspondent aux langages réguliers sans compteurs sur Σ dans un certain sens.

Soit $\Sigma = \Sigma_1 \cup \Sigma_2$ un alphabet partitionné. Soit φ un énoncé de $\mathsf{FO}^c(\Sigma)$. Soit $FVF\text{-}alt(\Sigma) = \{f \in FVF(\Sigma) \mid détemp(f) \in \Sigma_1 \cdot (\Sigma_2 \cdot \Sigma_1)^*\}$. Nous construisons un automate fini sans compteurs A_φ tel que pour tout $f \in FVF\text{-}alt(A)$, $f \in F(\varphi)$ si, et seulement si $détemp(f) \in A_\varphi$.

Nous gardons les mêmes notations pour les (E, F)-modèles d'une formule que dans la section 8.3. Pour les formules du premier ordre, nous omettons la seconde composante F puisque c'est toujours l'ensemble vide. Soit $f \in FVF(\Sigma)$, la fonction $f^E_{\mathbb{I}}$ est un E-modèle *alternant* d'une formule φ, si $f \in FVF\text{-}alt(A)$. Nous notons $\mathcal{AM}^E(\varphi)$ l'ensemble des E-modèles alternants de φ.

Lemme 9.2.1 *Soit Σ un alphabet et φ une formule de $\mathsf{FO}^c(\Sigma)$ avec comme variables libres $E = \{x_{i_1}, \cdots, x_{i_m}\}$. Soit E' un ensemble fini de variables du premier ordre qui contient E. Soit $\mathcal{A}$ un automate fini sans compteurs sur Σ $\mathcal{A}$ reconnaissant détemp($\mathcal{AM}^E(\varphi)$). Alors il existe un automate fini sans compteurs sur Σ $\mathcal{A}'$ reconnaissant détemp($\mathcal{AM}^{E'}(\varphi)$).*

Preuve. Nous faisons la preuve dans le cas $E' = E \cup \{x_{i_{m+1}}\}$, où $i_{m+1} > i_m$; la généralisation au cas $E \subseteq E'$ est facile. Les transitions de $\mathcal{A}$ sont étiquetées par des symboles de $\Sigma \times \{0,1\}^m$. Nous construisons l'automate $\mathcal{A}_1$ sur $\Sigma \times \{0,1\}^{m+1}$ en fixant la dernière composante à 0 sur toutes les transitions.

Remarquons qu'un E'-modèle de φ est un E-modèle de φ étendu avec un 1 à une unique position. Nous construisons donc l'automate $\mathcal{A}'$ en créant deux copies de $\mathcal{A}_1$ et en devinant de façon non-déterministe une transition de la première à la seconde copie étiquetée par un 1 en dernière position. Si $(q, (a, b_1, \cdots, b_m, 0), q')$ est une transition de $\mathcal{A}_1$ alors $((q, 0), (a, b_1, \cdots, b_m, 1), (q', 1))$ est une transition de $\mathcal{A}_2$, si $a \in \Sigma_1$ ou si l'un des b_i vaut 1. Cela correspond au cas où le 1 apparaît sur un point de discontinuité du modèle. Si le 1 apparaît sur un point de continuité du modèle, cela crée un nouveau point de discontinuité, ce qui nous amène à ajouter les transitions suivantes. Si $a \in \Sigma_2$ et que tous les b_i sont nuls, nous divisons la transition en trois : de $(q, 0)$ à un nouvel état q_1 sur l'action $(a, b_1, \cdots, b_m, 0)$, de q_1 à un nouvel état q_2 sur l'action $(a, b_1, \cdots, b_m, 1)$, et de q_2 à q'_1 sur l'action$(a, b_1, \cdots, b_m, 0)$. $\square$

Proposition 9.2.2 *Soit Σ un alphabet et soit $\varphi \in \mathsf{FO}^c(\Sigma)$. Soit E l'ensemble des variables libres de φ. Alors il existe un automate fini sans compteurs $\mathcal{A}_\varphi$ sur Σ tel que pour tout $f \in$ FVF-alt(Σ) et $\mathbb{I}$ interprétation sur E, $f, \mathbb{I} \models \varphi$ si, et seulement si détemp$(f_{\mathbb{I}}^E) \in L^*_{symb}(\mathcal{A}_\varphi)$.*

Preuve. La preuve est très similaire à celle du théorème 8.3.1, sauf que les automates construits doivent être sans compteurs. Les cas $\varphi = Q_a(x)$, $\varphi = x < y$ et $\varphi = \neg\eta$ sont identiques. Le cas $\varphi = \eta \vee \psi$ est similaire en utilisant le lemme 9.2.1.

Pour le cas $\varphi = \exists x \eta$ nous devons faire une construction différente. Soit E' l'ensemble des variables libres de η et soit $E = E' \backslash \{x\}$. Pour simplifier les notations supposons que x est la dernière variable de E', le cas général est similaire. Par hypothèse d'induction, soit $\mathcal{A}_\eta^{E'} = (\Sigma, Q, i, \delta, F)$ un automate fini sans compteurs pour η. Nous pouvons supposer $\mathcal{A}_\eta^{E'}$ co-accessible.

Nous projetons $\mathcal{A}_\eta^{E'}$ sur $\Sigma \times \{0,1\}^{|E|}$ en effaçant la composante en x. Pour tous les états q_1, $q_2 \in Q$ et tout $a' \in \Sigma \times \{0,1\}^{|E|}$ nous ajoutons alors une transition (q_1, a', q_2) s'il existe $p_1, p_2 \in Q$ tels que $(q_1, (a', 0), p_1) \in \delta$, $(p_1, (a', 1), p_2) \in \delta$ et $(p_2, (a', 0), q_2) \in \delta$. Soit $\mathcal{A}_1$ l'automate obtenu.

Comme $\mathcal{A}_\eta^{E'}$ est co-accessible, un symbole ayant un 1 dans une de ses composantes ne peut faire partie d'une boucle, les opérations ci-dessus n'introduisent donc pas de compteur. $\mathcal{A}_\varphi^E$ est alors l'automate intersection de $\mathcal{A}_{canon}^{E,\emptyset}$, $\mathcal{A}_{valid}^{E,\emptyset}$ et $\mathcal{A}_1$. $\square$

9.3 Caractérisation logique des *AIDC* sans compteurs

Dans cette section, nous définissons les automates *input-determined* sans compteurs et donnons une caractérisation de ces automates par une logique du premier ordre.

Définition 9.3.1 *Soit Σ un alphabet et Op un ensemble d'opérateurs. Un automate* input-determined *canonique pur sur (Σ, Op) est dit* sans-compteurs*, noté AIDCSC, si l'automate fini avec invariants associé est sans compteurs.*

Exemple 9.3.2 *Soit $\Sigma = \{a\}$, $Op = \{\Diamond_a\}$ et $G = \{\Diamond_a^{[1,2]}\}$. L'AIDCSC sur (Σ, Op) suivant reconnaît les mots temporisés qui ont exactement une action a dans l'intervalle $[1,2]$. Sur la figure, $g_1 = \{\Diamond_a^{[1,2]}\}$ et $g_2 = \emptyset$. Cet automate est sans compteurs : d'après sa forme s'il avait un compteur, il aurait un compteur de longueur 2 passant par les deux états centraux. Or pour tout mot $(\varepsilon, g)g' \in ((\Sigma \cup \{\varepsilon\}) \times 2^G) \cdot 2^G$, $(\varepsilon, g)g'$ ne permet pas de réaliser une boucle dans l'automate.*

Le résultat principal de cette section est que les langages définissables par un énoncé de TFO^c sont les langages reconnus par des automates *input-determined* continus sans compteurs :

Théorème 9.3.3 *Soit Σ un alphabet et Op un ensemble d'opérateurs* input-determined *à variation finie sur Σ. Un langage temporisé sur Σ est reconnu par un AIDCSC sur (Σ, Op) si, et seulement s'il est définissable par un énoncé de $\mathsf{TFO}^c(\Sigma, Op)$.*

Preuve.

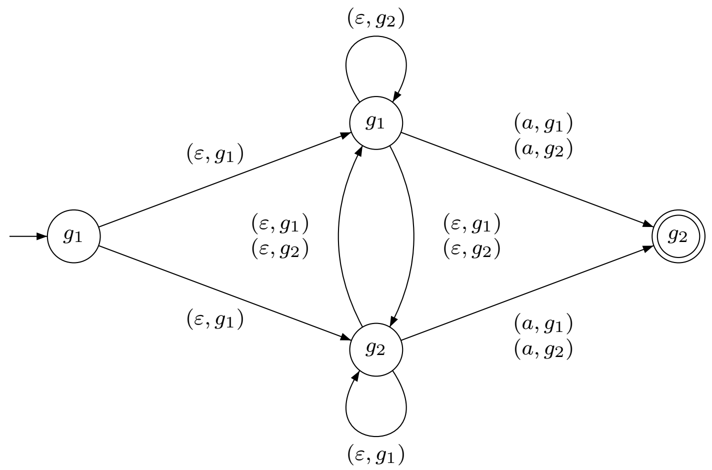


FIG. 9.1 – Automate *input-determined* sans compteurs

$$\begin{tikzcd}
AIDCSC \arrow[r, dotted] \arrow[d] & \mathsf{TLTL}^c \\
\mathsf{LTL} \arrow[r] & \mathsf{LTL}^c \arrow[u]
\end{tikzcd}$$

Implication directe. Nous montrons en fait que tout *AIDCSC* sur (Σ, Op) est équivalent à une formule de $\mathsf{TLTL}^c(\Sigma, Op)$. Par le théorème 8.5.1, $\mathsf{TLTL}^c(\Sigma, Op)$ est expressivement équivalente à $\mathsf{TFO}^c(\Sigma, Op)$, ce qui permettra de conclure.

Nous suivons le diagramme suivant :

La première transformation découle des résultats de Kamp [Kam68] et Mac-Naughton et Papert [MP71]. La troisième transformation a déjà été présentée dans la preuve du théorème 8.5.1. Nous présentons maintenant la deuxième transformation.

Soit Σ un alphabet. Nous définissons les deux formules suivantes de $\mathsf{LTL}^c(\Sigma)$:

- $\mathit{cont} = \bigvee_{a \in \Sigma}(a \wedge (a \, \mathbf{U} \, a) \wedge (a \, \mathbf{S} \, a))$ pour désigner un point de continuité du modèle,
- $\mathit{discont} = \neg \mathit{cont}$ pour désigner un point de discontinuité du modèle.

La transformation *ltl-ltlc* d'une formule de $\mathsf{LTL}(\Sigma)$ en une formule de $\mathsf{LTL}^c(\Sigma)$ est définie comme suit ($\hat{\varphi}$ désigne *ltl-ltlc*(φ)) :

$$\begin{array}{lcl}
\mathit{ltl\text{-}ltlc}(a) & = & a \\
\mathit{ltl\text{-}ltlc}(\neg\varphi) & = & \neg\hat{\varphi} \\
\mathit{ltl\text{-}ltlc}(\varphi_1 \vee \varphi_2) & = & \hat{\varphi}_1 \vee \hat{\varphi}_2 \\
\mathit{ltl\text{-}ltlc}(\varphi_1 \, \mathbf{U} \, \varphi_2) & = & \big(\mathit{discont} \Rightarrow (\hat{\varphi}_1 \, \mathbf{U} \, \hat{\varphi}_2)\big) \bigwedge \\
 & & \big(\mathit{cont} \Rightarrow ((\neg \mathit{discont}) \, \mathbf{U} \, (\mathit{discont} \wedge (\hat{\varphi}_2 \vee (\hat{\varphi}_1 \wedge \hat{\varphi}_1 \, \mathbf{U} \, \hat{\varphi}_2))))\big) \\
\mathit{ltl\text{-}ltlc}(\varphi_1 \, \mathbf{S} \, \varphi_2) & = & \big(\mathit{discont} \Rightarrow (\hat{\varphi}_1 \, \mathbf{S} \, \hat{\varphi}_2)\big) \bigwedge \\
 & & \big(\mathit{cont} \Rightarrow ((\neg \mathit{discont}) \, \mathbf{S} \, (\mathit{discont} \wedge (\hat{\varphi}_2 \vee (\hat{\varphi}_1 \wedge \hat{\varphi}_1 \, \mathbf{S} \, \hat{\varphi}_2))))\big)
\end{array}$$

La transformation *ltl-ltlc* vérifie la propriété suivante :

Lemme 9.3.4 *Soit Σ un alphabet et w un mot canonique sur Σ. Soit $f \in \mathit{temp}(w)$ ayant pour représentation canonique par intervalles $(w_0, I_0) \cdots (w_{2n}, I_{2n})$. Alors pour tout $0 \leq i \leq 2n$ et tout $t \in I_i$, $w, i \models \varphi \Leftrightarrow f, t \models \mathit{ltl\text{-}ltlc}(\varphi)$.*

Preuve.[Preuve du lemme] Nous raisonnons par induction sur φ. Le cas de base et l'induction pour les opérateurs booléens sont évidents. Montrons l'étape d'induction pour $\varphi = \varphi_1 \,\mathbf{U}\, \varphi_2$ (le cas $\varphi = \varphi_1 \,\mathbf{S}\, \varphi_2$ est similaire).

Implication directe. Soit $t \in I_i$ et supposons que $w, i \models \varphi_1 \,\mathbf{U}\, \varphi_2$.

Alors $\exists j > i$ tel que $w, i \models \varphi_2$ et $\forall i < i' < j \;\; w, i' \models \varphi_1$. Soit $t' \in I_j$. Remarquons que par hypothèse d'induction $f, t' \models \hat{\varphi}_2$, $\forall t'' \in I_j \;\; f, t'' \models \hat{\varphi}_2$ et $\forall t'' \in \cup_{i' | i < i' < j} I_{i'} \;\; f, t'' \models \hat{\varphi}_1$.

Nous distinguons deux cas :

- i est pair : alors $f, t \models \mathit{discont}$. Nous avons que $]t, t'[\subseteq \cup_{i' | i < i' < j} I_{i'} \cup I_j$. Comme nous l'avons remarqué $\forall t < t'' < t' \;\; f, t'' \models \hat{\varphi}_1 \vee \hat{\varphi}_2$, donc $f, t \models \hat{\varphi}_1 \,\mathbf{U}\, \hat{\varphi}_2$.
- i est impair : alors $f, t \models \mathit{cont}$. Soit t_1 tel que $I_{i+1} = \{t_1\}$. Nous avons que $\forall t_1 \leq t'' < t' \;\; f, t'' \models \hat{\varphi}_1 \vee \hat{\varphi}_2$. Donc $f, t_1 \models \mathit{discont} \wedge (\hat{\varphi}_2 \vee (\hat{\varphi}_1 \wedge \hat{\varphi}_1 \,\mathbf{U}\, \hat{\varphi}_2))$. De plus $\forall t < t'' < t_1 \;\; f, t'' \models \neg \mathit{discont}$, d'où $f, t \models (\neg \mathit{discont}) \,\mathbf{U}\, (\mathit{discont} \wedge (\hat{\varphi}_2 \vee (\hat{\varphi}_1 \wedge \hat{\varphi}_1 \,\mathbf{U}\, \hat{\varphi}_2)))$.

Implication réciproque. Soit $t \in I_i$ et supposons que $f, t \models \textit{ltl-ltlc}(\varphi)$. Nous distinguons deux cas :

- i est pair : alors $f, t \models \mathit{discont}$ donc $f, t \models \hat{\varphi}_1 \,\mathbf{U}\, \hat{\varphi}_2$. Il existe donc $t' > t$ tel que $f, t' \models \varphi_2$ et $\forall t < t'' < t' \; f, t'' \models \hat{\varphi}_1 \vee \hat{\varphi}_2$.
  Soit j tel que $t' \in I_j$. Par hypothèse d'induction $w, j \models \varphi_2$ et $\forall i < i' < j \;\; w, i' \models \varphi_1$, donc $w, i \models \varphi_1 \,\mathbf{U}\, \varphi_2$.
- i est impair : alors $f, t \models \mathit{cont}$ et donc $f, t \models (\neg \mathit{discont}) \,\mathbf{U}\, (\mathit{discont} \wedge (\hat{\varphi}_2 \vee (\hat{\varphi}_1 \wedge \hat{\varphi}_1 \,\mathbf{U}\, \hat{\varphi}_2)))$. Il existe alors $t_1 > t$ tel que $\forall t < t'' < t_1 \;\; f, t'' \models \neg \mathit{discont}$ et $f, t_1 \models \mathit{discont} \wedge (\hat{\varphi}_2 \vee (\hat{\varphi}_1 \wedge \hat{\varphi}_1 \,\mathbf{U}\, \hat{\varphi}_2))$. Nécessairement $I_{i+1} = \{t_1\}$.
  Soit $f, t_1 \models \hat{\varphi}_2$ alors par hypothèse d'induction $w, i+1 \models \varphi_2$ et $w, i \models \varphi_1 \,\mathbf{U}\, \varphi_2$. Sinon $f, t_1 \models \hat{\varphi}_1$ (donc par hypothèse d'induction $w, i+1 \models \varphi_1$) et $f, t_1 \models \hat{\varphi}_1 \,\mathbf{U}\, \hat{\varphi}_2$. $i+1$ est alors pair et nous pouvons appliquer le cas i pair pour obtenir que $w, i+1 \models \varphi_1 \,\mathbf{U}\, \varphi_2$. Nous déduisons que $w, i \models \varphi_1 \,\mathbf{U}\, \varphi_2$.

□

Nous pouvons achever la première moitié de la preuve du théorème 9.3.3 qui repose sur le lemme suivant.

Lemme 9.3.5 *Soit Σ un alphabet et Op un ensemble d'opérateurs* input-determined *à variation finie sur Σ. Soit $\mathcal{A}$ un AIDCSC sur (Σ, Op). Alors il existe une formule φ de $\mathsf{TLTL}^c(\Sigma, Op)$ telle que $L^*(\mathcal{A}) = L^*(\varphi)$.*

Preuve.[Preuve du lemme] Soit $\mathcal{A}$ un *AIDCSC* sur (Σ, Op), soit Γ l'alphabet symbolique propre associé. $L^*_{symb}(\mathcal{A})$ est reconnu par un automate fini sans compteurs sur Γ. Par [MP71] et [Kam68], il existe une formule φ_1 de $\mathsf{LTL}(\Gamma)$ telle que $L^*_{symb}(\mathcal{A}) = L^*(\varphi_1)$. Comme tout $w \in L^*_{symb}(\mathcal{A})$ est canonique, par le lemme 9.3.4, nous avons que $\mathit{temp}(L^*(\varphi_1)) = F(\varphi_2)$ où $\varphi_2 = \textit{ltl-ltlc}(\varphi_1)$. Nous utilisons alors la transformation *ltl-tltl* qui associe à φ_2 une formule φ telle que $L^*(\varphi) = \mathit{tw}(F(\varphi_2))$ si la détemporisation des fonctions de $F(\varphi_1)$ est composée de mots canoniques purs, ce qui est le cas car $\mathcal{A}$ est un *AIDCSC*. Nous avons donc que $L^*(\mathcal{A}) = L^*(\varphi)$. □

Implication réciproque. Cette implication repose sur le lemme suivant.

Lemme 9.3.6 *Soit* Σ *un alphabet et* Op *un ensemble d'opérateurs* input-determined *à variation finie sur* Σ. *Soit* φ *un énoncé de* $\mathsf{TFO}^c(\Sigma, Op)$, *alors il existe un AIDCSC* $\mathcal{A}$ *sur* (Σ, Op) *tel que* $L^*(\varphi) = L^*(\mathcal{A})$.

Preuve.[Preuve du lemme] Soit φ un énoncé de $\mathsf{TFO}^c(\Sigma, Op)$. Soit $G = \{\Delta^I \mid \Delta^I(x)$ est une sous-formule de $\varphi\}$ et soit $\Gamma = \Gamma_1 \cup \Gamma_2$ l'alphabet symbolique propre basé sur G. Par la proposition 8.4.3 il existe un énoncé $\tilde{\varphi}$ de $\mathsf{FO}^c(\Gamma)$ tel que $L^*(\varphi) = tw(F(\tilde{\varphi}))$.

D'après la proposition 9.2.2, il existe un automate fini sans compteurs $\mathcal{A}'$ tel que $F(\tilde{\varphi}) \cap FVF\text{-}alt(\Gamma) = temp(L^*_{symb}(\mathcal{A}'))$. De plus $L^*_{symb}(\mathcal{A}') \subseteq \Gamma_1 \cdot (\Gamma_2 \cdot \Gamma_1)^*$, donc par la proposition 9.1.3 un existe un automate fini avec invariants sans compteurs $\mathcal{A}$ tel que $L^*_{symb}(\mathcal{A}) = L^*_{symb}(\mathcal{A}')$. Nous avons alors la suite d'égalités $L^*(\varphi) = tw(F(\tilde{\varphi})) = tw(F(\tilde{\varphi}) \cap FVF\text{-}alt(\Gamma)) = tw(temp(L^*_{symb}(\mathcal{A}'))) = L^*(\mathcal{A})$. □

Ceci achève la preuve du théorème 9.3.3.

□

9.4 *AIDC* récursifs sans compteurs

Nous considérons maintenant des *AIDCSC* récursifs comme nous l'avions fait pour les *AIDC* dans la section 8.6. Nous avions montré que les *AIDC* récursifs étaient équivalents aux formules de $\mathsf{rec\text{-}TMSO}^c$; nous montrons ici que les *AIDCSC* récursifs correspondent au fragment du premier ordre de cette logique.

La définition des *rec-AIDCSC* (resp. *rec-AIDCSCp*)) est similaire à celle des *rec-AIDC* (resp. *rec-AIDCSCp*)) sauf qu'à chaque niveau les automates considérés doivent être sans compteurs (voir la section 8.6). Les *rec-AIDCSC* sont caractérisés par $\mathsf{rec\text{-}TFO}^c$ dans le sens suivant :

Théorème 9.4.1 *Soit* Σ *un alphabet et* Rop *un ensemble d'opérateurs* input-determined *récursifs à variation finie sur* Σ. *Soit* L *un langage temporisé sur* Σ. L *est reconnu par un rec-AIDCSC*(Σ, Rop) *si, et seulement si* L *est définissable par un énoncé de* $\mathsf{rec\text{-}TFO}^c(\Sigma, Rop)$.

Pour prouver le théorème 9.4.1, nous procédons comme pour la preuve du théorème 8.7.3 ; nous prouvons tout d'abord un équivalent *sans compteurs* du lemme 8.7.1 : les *AIDCSCp* correspondent aux formules de TFO^c avec une variable libre. La preuve du théorème 9.4.1 est alors strictement similaire à celle du théorème 8.7.3.

Lemme 9.4.2 *Soit* Σ *un alphabet et* Op *un ensemble d'opérateurs* input-determined *à variation finie sur* Σ. *Un langage temporisé pointé sur* Σ *est accepté par un AIDCSCp sur* (Σ, Op) *si, et seulement s'il est définissable par une formule de* $\mathsf{TFO}^c(\Sigma, Op)$ *avec une variable libre.*

Preuve.
Implication directe. Soit $\mathcal{B}$ un *AIDCSCp* sur (Σ, Op). Montrons qu'il existe une formule φ de $\mathsf{TFO}^c(\Sigma, Op)$ avec une variable libre tel que $L^p(\mathcal{B}) = L^p(\varphi)$.

$\mathcal{B}$ peut être vu comme un *AIDCSC* sur $(\Sigma \times \{0,1\}, Op)$. Par le théorème 9.3.3 il existe un énoncé ψ de $\mathsf{TFO}^c(\Sigma \times \{0,1\}, Op)$ tel que $L^*(\mathcal{B}) = L^*(\psi)$.

A partir de ψ nous construisons une formule $\varphi(z)$ de $\mathsf{TFO}^c(\Sigma, Op)$ avec une variable libre comme suit : pour toute variable x et pour tout $a \in \Sigma$ nous remplaçons $Q_{(a,1)}(x)$ par $Q_a(x) \wedge x = z$ et $Q_{(a,0)}(x)$ par $Q_a(x) \wedge \neg(x = z)$.

Nous rapellons que nous notons Σ' pour $(\Sigma \cup \{\varepsilon\}) \times \{0, 1\}$. Si σ' est un mot temporisé sur Σ', nous notons $premier(\sigma')$ le mot temporisé obtenu à partir de σ' en effaçant la composante $\{0, 1\}$ de σ' et les ε restants.

Rappelons que $valid(T\Sigma'^*)$ est l'ensemble des mots temporisés sur Σ' qui contiennent un unique symbole de $(\Sigma \cup \{\varepsilon\}) \times \{1\}$, et tels que leur dernier symbole est dans $\Sigma \times \{0, 1\}$.

Une induction immédiate sur ψ montre que :

$$\forall \sigma' \in valid(T\Sigma'^*)\ \forall \mathbb{I}\quad \sigma', \mathbb{I} \models \psi \Leftrightarrow premier(\sigma'), \mathbb{I}[z \mapsto dur(\sigma')] \models \varphi(z)$$

Nous en déduisons que $fw(L^*(\psi)) = L^p(\varphi(z))$; et donc $L^p(\mathcal{B}) = fw(L^*(\mathcal{B})) = fw(L^*(\psi)) = L^p(\varphi(z))$.

Implication réciproque. Soit φ une formule de $\mathsf{TFO}^c(\Sigma, Op)$ avec une variable libre. Montrons qu'il existe un *AIDCSCp* $\mathcal{B}$ sur (Σ, Op) tel que $L^p(\varphi) = L^p(\mathcal{B})$.

Soit z la variable libre de φ. Nous transformons $\varphi(z)$ en une formule de FO^c avec une variable libre, nous appliquons ensuite la proposition 9.2.2 pour obtenir un *AIDCSCp*.

Soit $G = \{\Delta^I \mid \Delta^I(x)$ est une sous-formule de $\varphi(z)\}$ et soit $\Gamma = \Gamma_1 \cup \Gamma_2$ l'alphabet symbolique propre basé sur G.

tmso-mso(φ) est une formule de $\mathsf{FO}^c(\Gamma)$ avec une variable libre. Soit $\mathcal{B}_{\varphi(z)}$ l'automate fini sans compteurs donné par la proposition 9.2.2.

$L^p(\varphi(z)) = L^p(\mathcal{B}_{\varphi(z)})$. En effet

$$\begin{array}{rcl}
(\sigma, t) \in L^p(\varphi(z)) & \Leftrightarrow & \sigma, [z \mapsto t] \models \varphi \\
& \Leftrightarrow & f^{\Gamma}_{\sigma}, [z \mapsto t] \models tmso\text{-}mso(\varphi)(z) \text{ par le lemme 8.4.2} \\
& \Leftrightarrow & d\acute{e}temp(f^{\Gamma}_{\sigma,[z\mapsto t]}) \in L^*_{symb}(\mathcal{B}_{\varphi(z)}) \text{ par la proposition 9.2.2} \\
& \Leftrightarrow & f^{\Gamma}_{\sigma,[z\mapsto t]} \in temp(L^*_{symb}(\mathcal{B}_{\varphi(z)})) \\
& \Leftrightarrow & (\sigma, t) \in fw(tw(temp(L^*_{symb}(\mathcal{B}_{\varphi(z)})))) \\
& \Leftrightarrow & (\sigma, t) \in L^p(\mathcal{B}_{\varphi(z)})
\end{array}$$

□

Comme application du théorème 9.4.1, nous déduisons une caractérisation de MTL+Past. D'après le théorème 8.9.1, MTL+Past(Σ) est expressivement équivalente à rec-$\mathsf{TFO}^c(\Sigma, \{\Diamond, \overleftarrow{\Diamond}\})$, le théorème 9.4.1 permet donc d'établir le résultat suivant :

Théorème 9.4.3 *Soit Σ un alphabet,* MTL+Past(Σ) *est expressivement équivalente à rec-AIDCSC*$(\Sigma, \{\Diamond, \overleftarrow{\Diamond}\})$.

9.5 Stabilité à l'infini de MTL+Past

Dans la section 9.4 nous avons montré que tout langage définissable dans MTL+Past (pour la sémantique continue) est reconnu par un *AIDC* récursif sans compteurs. Nous utilisons maintenant cette caractérisation pour montrer la *stabilité à l'infini* de MTL+Past ; c'est à dire que si un langage L est définissable dans MTL+Past, toute suite périodique de mots temporisés finira par rester dans L, ou finira par rester à l'extérieur de L. Ce résultat est connu pour LTL+Past dans le cadre non-temporisé [SC85, Mar03a] et pour MTL (sans passé) dans le cadre temporisé [PD06].

Dans cette section, Σ désigne un alphabet et *Rop* un ensemble d'opérateurs *input-determined* récursifs à variation finie.

Définition 9.5.1 *Une* suite périodique temporisée *est un triplet* $s = (u, v, w) \in (T\Sigma^*)^3$*, qui représente la suite de mots temporisés* $(s_i)_{i \geq 0}$ *où* $s_i = uv^i w$.

Un langage temporisé sur Σ *est dit* stable à l'infini *par rapport à une suite périodique temporisée* s *s'il existe* $i_0 \in \mathbb{N}$ *tel que soit* $\forall i \geq i_0,\ s_i \in L$*, soit* $\forall i \geq i_0,\ s_i \notin L$*. Un langage temporisé sur* Σ *est dit* stable à l'infini *s'il est stable à l'infini par rapport à toutes les suites périodiques temporisées.*

Théorème 9.5.2 *Soit* Σ *un alphabet et* φ *une formule de* MTL+Past(Σ) *interprétée pour la sémantique continue. Alors* $L^*(\varphi)$ *est stable à l'infini.*

Nous consacrons le reste de cette section à la preuve de ce théorème. Par le théorème 9.4.3, il suffit de montrer que tout langage reconnu par un *rec-AIDCSC*$(\Sigma, \{\Diamond, \overleftarrow{\Diamond}\})$ est stable à l'infini.

Nous présentons tout d'abord le concept de zone centrale : cela représente l'ensemble des points *au centre* d'une suite périodique temporisée. Une *zone centrale* est un couple $Z = (l, r)$ avec $l, r \in \mathbb{T}$. Étant donné un mot temporisé w sur Σ, $Z(w)$ est défini par $Z(w) =]l, dur(w) - r[$.

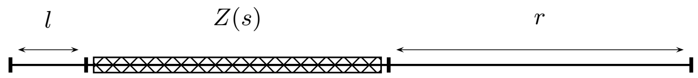


FIG. 9.2 – Une zone centrale

Un langage temporisé pointé L sur Σ est dit *correct* par rapport à une suite périodique temporisée $s = (u, v, w)$ s'il existe une zone centrale Z et un entier i_0 tels que les trois conditions suivantes sont vérifiées :

$$\forall i\ \forall i' \geq i\ \forall t \in Z(s_i),\ (s_i, t) \in L \Leftrightarrow (s_i, t + dur(v)) \in L \text{ et } (s_i, t) \in L \Leftrightarrow (s_{i'}, t) \in L \tag{9.1}$$

$$\forall i \geq i_0\ \forall i' \geq i\ \forall t < l,\ (s_i, t) \in L \Leftrightarrow (s_{i'}, t) \in L \tag{9.2}$$

$$\forall i \geq i_0\ \forall i' \geq i\ \forall t < r,\ (s_i, dur(s_i) - t) \in L \Leftrightarrow (s_{i'}, dur(s_{i'}) - t) \in L \tag{9.3}$$

Intuitivement, la première partie de la condition (9.1) signifie qu'à l'intérieur de la zone centrale, l'appartenance de (s_i, t) à L est périodique de période $dur(v)$. La deuxième partie de la condition (9.1) et les conditions (9.2) et (9.3) signifient que l'appartenance de s_i à L ne dépend plus de i pour $i \geq i_0$ (et est donc *stable* à partir de i_0).

Un langage temporisé pointé L sur Σ est dit *correct* s'il est correct par rapport à toute suite périodique temporisée. Notons qu'une garde Δ^I définit le langage temporisé pointé $\{(\sigma, t) \mid \sigma, t \models \Delta^I\}$. Nous disons qu'un *AIDCSCp* (resp. une garde) est *correct* par rapport à une suite périodique temporisée si le langage associé l'est. De même, nous disons qu'un *AIDCSCp* (resp. une garde) est *correct* si le langage associé l'est.

Étant donné un alphabet symbolique propre Γ et un mot temporisé w, nous notons γ_w^Γ l'unique mot symbolique $\gamma \in \Gamma^*$ tel que $w \in tw(\gamma)$.

Lemme 9.5.3 *Soit G un ensemble fini de gardes atomiques et Γ l'alphabet symbolique propre sur (Σ, Rop) basé sur G. Soit s une suite périodique temporisée. Si pour tout $g \in G$, g est correct par rapport à s alors il existe un entier i_0 et $\gamma_1, \gamma_2, \gamma_3 \in \Gamma^*$ tels que pour tout $i \geq i_0$, $\gamma^\Gamma_{s_i} = \gamma_1 \gamma_2^{i-i_0} \gamma_3$.*

Remarquons que ce lemme montre que si pour tout $g \in G$, g est correct par rapport à s, alors tout *AIDCSCp* basé sur G est correct par rapport à s.

Preuve. Soit $G = \{g_1, \cdots, g_k\}$. Comme $g_1, \cdots, g_k$ sont correctes, il existe des zones centrales $Z_1, \cdots, Z_k$ et des entiers $i_1, \cdots, i_k$ tels que les propriétés (9.1), (9.2) et (9.3) sont vérifiées.

Nous définissons le maximum d'un nombre fini de zones centrales comme le maximum composante par composante. Soit $Z = \max\limits_{1 \leq j \leq k} Z_j$ et $i = \max\limits_{1 \leq j \leq k} i_j$. La zone centrale Z et l'entier i satisfont les propriétés (9.1), (9.2) et (9.3) pour tout $g \in G$.

Soit i_0 un entier tel que $i_0 \geq i$ et $Z(s_{i_0}) \neq \emptyset$. Soit $\gamma_1 \in \Gamma^*$ tel qu'il existe un préfixe p_{i_0} de s_{i_0} avec $dur(p_{i_0}) \in Z(s_{i_0})$ et $p_{i_0} \in tw(\gamma_1)$. Soit γ_3 tel que $\gamma^\Gamma_{s_i} = \gamma_1\gamma_3$. Soit γ_2 tel que $\gamma^\Gamma_{s_{i+1}} = \gamma_1\gamma_2\gamma_3$. Alors par les propriétés (9.1), (9.2) et (9.3), $\forall i \geq i_0, \gamma^\Gamma_{s_i} = \gamma_1\gamma_2^{i-i_0}\gamma_3$. □

Lemme 9.5.4 *Soit $\mathcal{B}$ un AIDCSCp canonique propre sur (Σ, Rop) et $I \in \mathcal{I}_\mathbb{Q}$. Si $\mathcal{B}$ est correct, alors les gardes $\Diamond^I_\mathcal{B}$ et $\overleftarrow{\Diamond}^I_\mathcal{B}$ sont correctes.*

Preuve. Soit $s = (u, v, w)$ une suite périodique temporisée. Nous montrons que $\Diamond^I_\mathcal{B}$ est correcte par rapport à s, le cas $\overleftarrow{\Diamond}^I_\mathcal{B}$ est similaire.

Nous devons montrer qu'il existe une zone centrale $Z = (l, r)$ et un entier i_0 tels que :

$$\forall i\ \forall i' \geq i\ \forall t \in Z(s_i),\ s_i, t \models \Diamond^I_\mathcal{B} \Leftrightarrow s_i, t + dur(v) \models \Diamond^I_\mathcal{B} \text{ et } s_i, t \models \Diamond^I_\mathcal{B} \Leftrightarrow s_{i'}, t \models \Diamond^I_\mathcal{B} \tag{9.1}$$

$$\forall i \geq i_0\ \forall i' \geq i\ \forall t < l,\ s_i, t \models \Diamond^I_\mathcal{B} \Leftrightarrow s_{i'}, t \models \Diamond^I_\mathcal{B} \tag{9.2}$$

$$\forall i \geq i_0\ \forall i' \geq i\ \forall t < r,\ s_i, dur(s_i) - t \models \Diamond^I_\mathcal{B} \Leftrightarrow s_{i'}, dur(s_{i'}) - t \models \Diamond^I_\mathcal{B} \tag{9.3}$$

$\mathcal{B}$ est correct par hypothèse. Soit $Z_\mathcal{B} = (l_\mathcal{B}, r_\mathcal{B})$ une zone centrale pour $\mathcal{B}$. Soit M un réel strictement plus grand que les bornes de I. Nous définissons $Z = (l_\mathcal{B}, r_\mathcal{B} + M + dur(v))$.

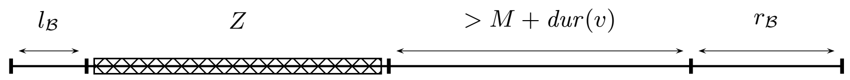
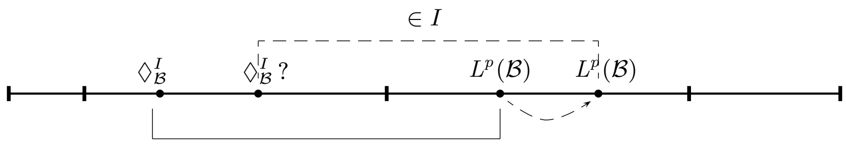


FIG. 9.3 – Choix de Z

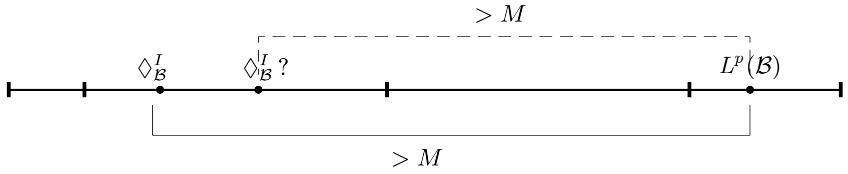
Soient i, t tels que $t \in Z(s_i)$ et supposons que $s_i, t \models \Diamond^I_\mathcal{B}$. Nous montrons que $s_i, t + dur(v) \models \Diamond^I_\mathcal{B}$ (la réciproque est similaire).

Nous avons que $\exists t_1 \geq t$ avec $(s_i, t_1) \in L^p(\mathcal{B})$ et $t_1 - t \in I$. Nous distinguons deux cas :

- $t_1 \in Z_\mathcal{B}(s_i)$
  Comme $\mathcal{B}$ est correct $(s_i, t_1 + dur(v)) \in L^p(\mathcal{B})$ et nous avons que $t_1 + dur(v) - (t + dur(v)) = t_1 - t \in I$. Donc $s_i, t + dur(v) \models \Diamond^I_\mathcal{B}$



– $t_1 \notin Z_{\mathcal{B}}(s_i)$
  Alors $t_1 - t > M$, donc nécessairement $I =]c, +\infty[$ ou $I = [c, +\infty[$. Comme $t_1 - (t + dur(v)) > M$, $s_i, t + dur(v) \models \Diamond^I_{\mathcal{B}}$.



Un raisonnement similaire montre que $s_{i'}, t \models \Diamond^I_{\mathcal{B}}$.

Nous montrons maintenant comment choisir i_0 et prouvons la propriété (9.2) (la propriété (9.3) se montre de façon similaire). Prenons $i_0 \in \mathbb{N}$ tel que $i_0 \geq i_{\mathcal{B}}$ et $dur(Z(y_{i_0})) \geq M$ (de telle façon à ce que le temps à l'intérieur de la zone centrale soit plus grand que M pour $i \geq i_0$).

Soient $i \geq i_0, i' \geq i, t < l$ et supposons que $s_i, t \models \Diamond^I_{\mathcal{B}}$. Alors il existe $t_1 \geq t$ avec $(s_i, t_1) \in L^p(\mathcal{B})$ et $t_1 - t \in I$.

Nous distinguons deux cas :

– $t_1 \leq dur(s_i) - r$
  Comme $\mathcal{B}$ est correct $(s_{i'}, t_1) \in L^p(\mathcal{B})$ et donc $s_{i'}, t \models \Diamond^I_{\mathcal{B}}$
– $t_1 > dur(s_i) - r$
  Alors $t_1 - t > M$ et nécessairement $I =]c, +\infty[$ ou $I = [c, +\infty[$.
  Soit $t'_1 = dur(s_i) - t_1$. Nous avons $(s_i, dur(s_i) - t'_1) \in L^p(\mathcal{B})$, donc comme $\mathcal{B}$ est correct $(s_{i'}, dur(s_{i'}) - t'_1) \in L^p(\mathcal{B})$. De plus $(dur(s_{i'}) - t'_1) - t \geq (dur(s_i) - t'_1) - t = t_1 - t > M$, d'où $s_{i'}, t \models \Diamond^I_{\mathcal{B}}$.

$\square$

Nous achevons la preuve du théorème 9.5.2 : soit L reconnu par un *rec-AIDCSC* $\mathcal{A}$ et soit s une suite périodique temporisée. Soit Γ l'alphabet symbolique propre basé sur les gardes atomiques de $\mathcal{A}$; par les lemmes 9.5.3 et 9.5.4, il existe un entier i_0 et des mots symboliques $\gamma_1, \gamma_2, \gamma_3 \in \Gamma^*$ tels que pour tout $i \geq i_0$, $\gamma^\Gamma_{s_i} = \gamma_1 \gamma_2^{i-i_0} \gamma_3$. Soit n le nombre d'état de $\mathcal{A}$. γ_2 n'est pas un compteur de $\mathcal{A}$ donc pour $i \geq i_0 + n$, les exécutions de $\mathcal{A}$ sur s_i terminent toutes dans le même état. L est donc stable à l'infini par rapport à s.

Remarque 9.5.5 *Le théorème 9.5.2 montre que les formules de* MTL+Past *pour la sémantique continue sont stables à l'infini. Ce résultat est vrai également pour les formules de* MTL+Past *pour la sémantique d'actions, car la sémantique continue est plus expressive (voir la proposition 5.2.2). De telles propriétés permettent d'obtenir des résultats d'expressivité [PD06].*

Chapitre 10

Conclusion

Ce travail de recherche avait pour but d'étudier des aspects d'expressivité et de contrôle dans le cadre des logiques temporisées. Nous récapitulons maintenant les résultats obtenus et donnons quelques perspectives.

Systèmes o-minimaux

Nous avons introduit le problème de contrôle pour les systèmes dynamiques. Nous avons montré que les techniques classiques pour les jeux non-temporisés à nombre infini d'états ne pouvaient s'appliquer dans le cadre du contrôle hybride car deux états bisimilaires peuvent ne pas être équivalents du point de vue du contrôle temporisé. Nous avons utilisé une technique d'encodage des trajectoires par des mots, plus fine que la bisimulation, et montré que cet encodage était correct pour le contrôle temporisé hybride. Nous avons ensuite appliqué ces méthodes au cas des systèmes o-minimaux et montré que leur problème de contrôle est décidable, des stratégies gagnantes pouvant être explicitement construites de manière définissable. Dans un cadre de contrôle o-minimal avec une hypothèse d'observation partielle de la dynamique, nous avons obtenu des résultats similaires, c'est-à-dire la décidabilité du problème de contrôle et la définissabilité de stratégies gagnantes.

Nous avons également enrichi le modèle des systèmes o-minimaux avec des fonctions de coût et avons considéré deux problème naturels sur ces systèmes pondérés : le problème de contrôle optimal, et le *model-checking* de WCTL. Un tel système hybride n'est pas o-minimal, en particulier les techniques d'encodage par des mots développées précédemment ne permettent pas de résoudre ces problèmes ; nous avons donc développé des techniques spécifiques aux systèmes o-minimaux pondérés. Nous avons montré que le problème de contrôle optimal et le *model-checking* de WCTL sont décidables pour les systèmes o-minimaux pondérés, ce qui renforce l'attrait de ce modèle, car ces problèmes sont indécidables pour les automates temporisés pondérés [BBR04, BBR05, BBM06].

Contrôle pour des spécifications MTL

Nous avons considéré le problème de contrôle temporisé où la spécification est donnée par une formule de logique temporisée. Nous avons considéré des logiques pour lesquelles le problème de satisfaisabilité est décidable, en particulier MTL sur les mots finis et Safety-MTL sur les mots infinis. Nous avons montré que le problème de contrôle pour ces spécifications est indécidable dans le cas général. Nous avons alors considéré le problème de contrôle à ressources fixées, qui est algorithmiquement plus facile. Les techniques existantes sur le contrôle temporisé à ressources fixées ne pouvant s'appliquer, nous nous sommes inspirés des techniques d'ordre partiel [AN01, OW04, OW05] pour obtenir la décidabilité du contrôle à ressources fixées pour des spécifications MTL sur les mots finis et Safety-MTL sur les mots infinis.

Décidabilité et expressivité de MTL et TPTL

Nous avons proposé une preuve originale de l'indécidabilité de la satisfaisabilité de TPTL et MTL+Past pour la sémantique d'actions et la sémantique continue, et MTL pour la sémantique continue. Cette preuve que nous espérons plus simple, permet d'identifier les propriétés qui, si elles sont exprimables dans une logique temporelle, mènent à son indécidabilité.

Nous avons montré que TPTL était strictement plus expressive que MTL, confirmant une conjecture énoncée dans [AH92b]. Nous avons montré que la formule de TPTL proposée dans [AH92b] ne pouvait en effet pas être exprimée dans MTL pour la sémantique d'actions. Par contre, nous avons montré que cette formule pouvait être exprimée dans MTL pour la sémantique continue. Nous avons cependant donné une autre formule de TPTL qui ne peut être exprimée dans MTL pour la sémantique continue, confirmant ainsi la conjecture. Les formules que nous avons proposées pour montrer ces résultats nous ont permis de déduire d'autres propriétés intéressantes des logiques temporisées comme le fait qu'ajouter des modalités du passé à MTL et MITL augmente strictement leur expressivité, ou le fait que MTL est strictement plus expressive pour la sémantique continue que pour la sémantique d'actions.

Un nouveau formalisme d'automates pour les logiques temporisées

Nous avons défini une classe générique d'automates paramétrés par des opérateurs pour la sémantique continue, les automates *input-determined* continus. Nous avons montré que ces automates sont clos pour les opérations booléennes et ont de fortes propriétés logiques : ils sont expressivement équivalents à une logique monadique du second ordre paramétrée par des opérateurs, et le fragment du premier ordre de cette logique correspond à une extension de LTL. Nous avons étendu ces résultats dans un cadre récursif plus proche des logiques temporisées. En considérant des opérateurs particuliers, nous avons caractérisé la logique MTL+Past par une logique du premier ordre. Nous avons alors identifié une sous-classe d'automates correspondant à cette logique du premier ordre, les automates *input-determined* sans compteurs. Cette représentation nous a permis de montrer une propriété de stabilité à l'infini de MTL+Past.

Perspectives

Plusieurs directions sont envisageables pour continuer ces travaux. Dans le cadre des systèmes o-minimaux, nous pouvons considérer d'autres propriétés pour le *model-checking* comme MTL ainsi que d'autres conditions de victoire pour le contrôle comme TATL. L'étude des systèmes pondérés pourrait être étendue à celle du problème de consommation moyenne comme dans [BBL04]. Certains problèmes de contrôle n'ont pas été traités : le cadre d'observation partielle des actions (en plus de l'observation partielle de la dynamique) ; ainsi que la prise en compte de la convergence du temps, les techniques pour traiter ce problème dans le cadre temporisé ne pouvant s'appliquer aux systèmes o-minimaux.

Nous pouvons étudier le problème de contrôle dans le cadre des automates temporisés pour d'autres propriétés. Nous pourrions en particulier considérer des logiques dont le problème de satisfaisabilité a une complexité plus basse que MTL et Safety-MTL, comme par exemple le fragment Bounded-MTL introduit dans [BMOW07] (dont le problème de satisfaisabilité est EXPSPACE-complet). Le problème de contrôle général pourrait être décidable, et il serait intéressant d'analyser le saut de complexité dû à l'interaction entre le contrôleur et l'environnement.

Dans le cadre de l'expressivité des logiques temporisées, nous avons montré que certaines formules de TPTL à une horloge ne peuvent être exprimées dans MTL. Le problème de savoir si toute formule de TPTL a un équivalent utilisant une horloge est ouvert. Nous pouvons également tenter d'appliquer à d'autres opérateurs les résultats génériques obtenus dans l'étude des automates *input-determined* continus. Le cadre logique que nous avons établi permet d'obtenir des résultats d'expressivité concernant MTL+Past comme nous l'avons montré ; en considérant d'autres opérateurs nous pouvons retrouver les logiques MITL ou EventClockTL [HRS98]. Nous pouvons tenter d'utiliser les caractérisations notamment par automates que nous avons obtenues pour obtenir des résultats d'expressivité ou d'autres applications comme la stabilité à l'infini de MTL+Past.

Bibliographie

[ABd03] Parosh Aziz Abdulla, Ahmed Bouajjani, and Julien d'Orso. Deciding monotonic games. In *Proc. 17th International Workshop on Computer Science Logic (CSL'03)*, volume 2803 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2003.

[ABM04] Rajeev Alur, Mikhail Bernadsky, and P. Madhusudan. Optimal reachability in weighted timed games. In *Proc. 31st International Colloquium on Automata, Languages and Programming (ICALP'04)*, volume 3142 of *Lecture Notes in Computer Science*, pages 122–133. Springer, 2004.

[ACH+95] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1) :3–34, 1995.

[Acz88] Peter Aczel. *Non-Well-Founded Sets*, volume 14 of *CSLI Lecture Notes*. Center for the Study of Language and Information, Stanford University, 1988.

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

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

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

[AFH96] Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1) :116–146, 1996.

[AFH+03] Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rapuk Majumdar, and Mariëlla Stoelinga. The element of surprise in timed games. In *Proc. 14th International Conference on Concurrency Theory (CONCUR'03)*, volume 2761 of *Lecture Notes in Computer Science*, pages 142–156. Springer, 2003.

[AH89] Rajeev Alur and Thomas Henzinger. A really temporal logic. In *Proc. 30th Annual Symposium on Foundations of Computer Science (FOCS'89)*, pages 164–169. IEEE Computer Society Press, 1989.

[AH92a] Rajeev Alur and Thomas A. Henzinger. Back to the future : towards a theory of timed regular languages. In *Proc. 33rd Annual Symposium on Foundations* *of Computer Science (FOCS'92)*, pages 177–186. IEEE Computer Society Press, 1992.

[AH92b] Rajeev Alur and Thomas A. Henzinger. Logics and models of Real-Time : a survey. In *Proc. REX Workshop 1991*, volume 600 of *Lecture Notes in Computer Science*, pages 74–106. Springer, 1992.

[AH93] Rajeev Alur and Thomas A. Henzinger. Real-time logics : Complexity and expressiveness. *Information and Computation*, 104(1) :35–77, 1993.

[AH94] Rajeev Alur and Thomas A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1) :181–204, 1994.

[AHLP00] Rajeev Alur, Thomas A. Henzinger, Gerardo Lafferriere, and George J. Pappa. Discrete abstractions of hybrid systems. *Proc. of the IEEE*, 88 :971–984, 2000.

[ALM05] Rajeev Alur, Salvatore La Torre, and P. Madhusudan. Perturbed timed automata. In *Proc. 8th International Workshop on Hybrid Systems : Computation and Control (HSCC'05)*, volume 3414 of *Lecture Notes in Computer Science*, pages 70–85. Springer, 2005.

[ALP01] Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal paths in weighted timed automata. In *Proc. 4th International Workshop on Hybrid Systems : Computation and Control (HSCC'01)*, volume 2034 of *Lecture Notes in Computer Science*, pages 49–62. Springer, 2001.

[AM04] Rajeev Alur and P. Madhusudan. Decision problems for timed automata : A survey. In *Proc. 4th International School on Formal Methods for the Design of Computer, Communication and Software Systems : Real Time (SFM-04 :RT)*, volume 3185 of *Lecture Notes in Computer Science*, pages 122–133. Springer, 2004.

[AMPS98] Eugene Asarin, Oded Maler, Amir Pnueli, and Joseph Sifakis. Controller synthesis for timed automata. In *Proc. IFAC Symposium on System Structure and Control*, pages 469–474. Elsevier Science, 1998.

[AN00] Parosh Aziz Abdulla and Aletta Nylén. Better is better than well : On efficient verification of infinite-state systems. In *Proc. 15th Annual Symposium on Logic in Computer Science (LICS'00)*, pages 132–140. IEEE Computer Society Press, 2000.

[AN01] Parosh Aziz Abdulla and Aletta Nylén. Timed Petri nets and bqos. In *Proc. 22nd International Conference on Application and Theory of Petri Nets (ICATPN'01)*, volume 2075 of *Lecture Notes in Computer Science*, pages 53–70. Springer, 2001.

[AVW03] André Arnold, Aymeric Vincent, and Igor Walukiewicz. Games for synthesis of controllers with partial observation. *Theoretical Computer Science*, 1(303) :7–34, 2003.

[BBBR06] Patricia Bouyer, Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On the optimal reachability problem. Submitted, 2006.

[BBC06a] Patricia Bouyer, Laura Bozzelli, and Fabrice Chevalier. Controller synthesis for MTL specifications. In *Proc. 17th International Conference on Concurrency Theory (CONCUR'06)*, volume 4137 of *Lecture Notes in Computer Science*, pages 450–464. Springer, 2006.

[BBC06b] Patricia Bouyer, Thomas Brihaye, and Fabrice Chevalier. Control in o-minimal hybrid systems. In *Proc. 21st Annual Symposium on Logic in Computer Science (LICS'06)*, pages 367–378. IEEE Computer Society Press, 2006.

[BBC07] Patricia Bouyer, Thomas Brihaye, and Fabrice Chevalier. Weighted o-minimal hybrid systems are more decidable than weighted timed automata ! In *Proc. Symposium on Logical Foundations of Computer Science (LFCS'07)*, Lecture Notes in Computer Science, New-York, NY, USA, 2007. Springer. To appear.

[BBL04] Patricia Bouyer, Ed Brinksma, and Kim G. Larsen. Staying alive as cheaply as possible. In *Proc. 7th International Workshop on Hybrid Systems : Computation and Control (HSCC'04)*, volume 2993 of *Lecture Notes in Computer Science*, pages 203–218. Springer, 2004.

[BBM06] Patricia Bouyer, Thomas Brihaye, and Nicolas Markey. Improved undecidability results on weighted timed automata. *Information Processing Letters*, 98(5) :188–194, 2006.

[BBR04] Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. Model-checking for weighted timed automata. In *Proc. Joint Conference on Formal Modelling and Analysis of Timed Systems and Formal Techniques in Real-Time and Fault Tolerant System (FORMATS+FTRTFT'04)*, volume 3253 of *Lecture Notes in Computer Science*, pages 277–292. Springer, 2004.

[BBR05] Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On optimal timed strategies. In *Proc. 3rd International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'05)*, volume 3821 of *Lecture Notes in Computer Science*, pages 49–64. Springer, 2005.

[BC06] Patricia Bouyer and Fabrice Chevalier. On the control of timed and hybrid systems. *EATCS Bulletin*, 89 :79–96, 2006.

[BC07] Patricia Bouyer and Fabrice Chevalier. On conciseness of extensions of timed automata. *Journal of Automata, Languages and Combinatorics*, 10(4) :393–405, 2007. To appear.

[BCD05] Patricia Bouyer, Fabrice Chevalier, and Deepak D'Souza. Fault diagnosis using timed automata. In *Proc. 8th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'05)*, volume 3441 of *Lecture Notes in Computer Science*, pages 219–233. Springer, 2005.

[BCFL04] Patricia Bouyer, Franck Cassez, Emmanuel Fleury, and Kim G. Larsen. Optimal strategies in priced timed game automata. In *Proc. 24th Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS'04)*, volume 3328 of *Lecture Notes in Computer Science*, pages 148–160. Springer, 2004.

[BCKT05] Patricia Bouyer, Fabrice Chevalier, Moez Krichen, and Stavros Tripakis. Observation partielle des systèmes temporisés. In *Journal Européen des Systèmes Automatisés (Actes du 5ème Colloque sur la Modélisation des Systèmes Réactifs (MSR'05))*, volume 39, pages 381–393. Hermès, 2005.

[BCL05] Patricia Bouyer, Franck Cassez, and François Laroussinie. Modal logics for timed control. In *Proc. 16th International Conference on Concurrency Theory (CONCUR'05)*, volume 3821 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2005.

[BCM05] Patricia Bouyer, Fabrice Chevalier, and Nicolas Markey. On the expressiveness of TPTL and MTL. In *Proc. 25th Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS'05)*, volume 3821 of *Lecture Notes in Computer Science*, pages 432–443. Springer, 2005.

[BDMP03] Patricia Bouyer, Deepak D'Souza, P. Madhusudan, and Antoine Petit. Timed control with partial observability. In *Proc. 15th International Conference on Computer Aided Verification (CAV'03)*, volume 2725 of *Lecture Notes in Computer Science*, pages 180–192. Springer, 2003.

[BFH+01] Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim G. Larsen, Paul Pettersson, Judi Romijn, and Frits Vaandrager. Minimum-cost reachability for priced timed automata. In *Proc. 4th International Workshop on Hybrid Systems : Computation and Control (HSCC'01)*, volume 2034 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2001.

[BLM07] Patricia Bouyer, Kim G. Larsen, and Nicolas Markey. Model-checking one-clock priced timed automata. In *Proc. 10th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'07)*, volume 4423 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2007.

[BLMR06] Patricia Bouyer, Kim G. Larsen, Nicolas Markey, and Jacob I. Rasmussen. Almost optimal strategies in one-clock priced timed automata. In *Proc. 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS'06)*, volume 4337 of *Lecture Notes in Computer Science*, pages 345–356. Springer, 2006.

[BM05] Thomas Brihaye and Christian Michaux. On the expressiveness and decidability of o-minimal hybrid systems. *Journal of Complexity*, 21(4) :447–478, 2005.

[BMO+07] Patricia Bouyer, Nicolas Markey, Joël Ouaknine, Philippe Schnoebelen, and James Worrell. On the complexity of termination in faulty channel machines. Submitted, 2007.

[BMOW07] Patricia Bouyer, Nicolas Markey, Joël Ouaknine, and James Worrell. The cost of punctuality. In *Proc. 22nd Annual Symposium on Logic in Computer Science (LICS'07)*. IEEE Computer Society Press, 2007.

[BMRT04] Thomas Brihaye, Christian Michaux, Cédric Rivière, and Christophe Troestler. On o-minimal hybrid systems. In *Proc. 7th International Workshop on Hybrid Systems : Computation and Control (HSCC'04)*, volume 2993 of *Lecture Notes in Computer Science*, pages 219–233. Springer, 2004.

[Bou04] Patricia Bouyer. Forward analysis of updatable timed automata. *Formal Methods in System Design*, 24(3) :281–320, 2004.

[Bri05] Thomas Brihaye. Words and bisimulation of dynamical systems. *Journal of Automata, Languages and Combinatorics*, 2005. To appear.

[Bri06] Thomas Brihaye. *Verification and Control of O-Minimal Hybrid Systems and Weighted Timed Automata*. PhD thesis, Université de Mons-Hainaut, Belgium, 2006.

[BZ83] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2) :323–342, 1983.

[Cau95] Didier Caucal. *Bisimulation of Context-Free Grammars and of Pushdown Automata*, volume 53 of *CSLI Lecture Notes*, pages 85–106. Stanford University, 1995.

[CDP06] Fabrice Chevalier, Deepak D'Souza, and Pavithra Prabakhar. On continuous timed automata with input-determined guards. In *Proc. 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS'06)*, volume 4337 of *Lecture Notes in Computer Science*, pages 369–380. Springer, 2006.

[Cor06] Uppaal Cora, 2006. http ://www.cs.aau.dk/~behrmann/cora/.

[dAHM01] Luca de Alfaro, Thomas A. Henzinger, and Rupak Majumdar. Symbolic algorithms for infinite-state games. In *Proc. 12th International Conference on Concurrency Theory (CONCUR'01)*, volume 2154 of *Lecture Notes in Computer Science*, pages 536–550. Springer, 2001.

[Dav99] Jennifer M. Davoren. Topologies, continuity and bisimulations. *Informatique Théorique et Applications*, 33(4-5) :357–382, 1999.

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

[DM05] Deepak D'Souza and M. Raj Mohan. Eventual Timed Automata. In *Proc. 25th Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS'05)*, volume 3821 of *Lecture Notes in Computer Science*, pages 322–334. Springer, 2005.

[DP07] Deepak D'Souza and Pavithra Prabhakar. On the expressiveness of MTL in the pointwise and continuous semantics. *Formal Methods Letters*, 9(1) :1–4, 2007.

[D'S03] Deepak D'Souza. A logical characterisation of event clock automata. *International Journal of Foundations of Computer Science*, 14(4) :625–640, 2003.

[DT04] Deepak D'Souza and Nicolas Tabareau. On timed automata with input-determined guards. In *Proc. Joint Conference on Formal Modelling and Analysis of Timed Systems and Formal Techniques in Real-Time and Fault Tolerant System (FORMATS+FTRTFT'04)*, volume 3253 of *Lecture Notes in Computer Science*, pages 68–83. Springer, 2004.

[EH86] E. Allen Emerson and Joseph Y. Halpern. "Sometimes" and "not never" revisited : On branching versus linear time temporal logic. *Journal of the ACM*, 33(1) :151–178, 1986.

[Eme91] E. Allen Emerson. *Temporal and Modal Logic*, volume B (Formal Models and Semantics) of *Handbook of Theoretical Computer Science*, pages 995–1072. MIT Press Cambridge, 1991.

[FLM02a] Marco Faella, Salvatore La Torre, and Aniello Murano. Automata-theoretic decision of timed games. In *Proc. 3rd International Workshop on Verification, Model Checking, and Abstract Interpretation (VMCAI'02)*, volume 2294 of *Lecture Notes in Computer Science*, pages 94–108. Springer, 2002.

[FLM02b] Marco Faella, Salvatore La Torre, and Aniello Murano. Dense real-time games. In *Proc. 17th Annual Symposium on Logic in Computer Science (LICS'02)*, pages 167–176. IEEE Computer Society Press, 2002.

[FS01] Alain Finkel and Philippe Schnoebelen. Well structured transition systems everywhere ! *Theoretical Computer Science*, 256(1–2) :63–92, 2001.

[GPSS80] Dov M. Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. On the temporal analysis of fairness. In *Conference Record 7th ACM Symposium on Principles of Programming Languages (POPL'80)*, pages 163–173. ACM, 1980.

[GThW02] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games : A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[Hen95] Thomas A. Henzinger. Hybrid automata with finite bisimulations. In *Proc. 22nd International Colloquium on Automata, Languages and Programming (ICALP'95)*, volume 944 of *Lecture Notes in Computer Science*, pages 324–335. Springer, 1995.

[Hen96] Thomas A. Henzinger. The theory of hybrid automata. In *Proc. 11th Annual Symposium on Logic in Computer Science (LICS'96)*, pages 278–292. IEEE Computer Society Press, 1996.

[Hen98] Thomas A. Henzinger. It's about time : Real-time logics reviewed. In *Proc. 9th International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 of *Lecture Notes in Computer Science*, pages 439–454. Springer, 1998.

[HHM99] Thomas A. Henzinger, Benjamin Horowitz, and Rupak Majumdar. Rectangular hybrid games. In *Proc. 10th International Conference on Concurrency Theory (CONCUR'99)*, volume 1664 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 1999.

[HK99] Thomas A. Henzinger and Peter W. Kopke. Discrete-time control for rectangular hybrid automata. *Theoretical Computer Science*, 221 :369–392, 1999.

[HKPV98] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata ? *Journal of Computer and System Sciences*, 57(1) :94–124, 1998.

[HMR05] Thomas A. Henzinger, Rupak Majumdar, and Jean-François Raskin. A classification of symbolic transition systems. *ACM Transactions on Computational Logic*, 6(1) :1–32, 2005.

[Hod97] Wilfrid Hodges. *A Shorter Model Theory*. Cambridge University Press, 1997.

[HRS98] Thomas A. Henzinger, Jean-François Raskin, and Pierre-Yves Schobbens. The regular real-time languages. In *Proc. 25th International Colloquium on Automata, Languages and Programming (ICALP'98)*, volume 1443 of *Lecture Notes in Computer Science*, pages 580–591. Springer, 1998.

[HRS99] Thomas A. Henzinger, Jean-Francois Raskin, and Pierre-Yves Schobbens. Axioms for real-time logics. Research Report MPI-I-1999-3-005, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, 1999.

[Kam68] Johan A.W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, UCLA, Los Angeles, CA, USA, 1968.

[Koy90] Ron Koymans. Specifying real-time properties with Metric Temporal Logic. *Real-Time Systems*, 2(4) :255–299, 1990.

[KPS86] Julia F. Knight, Anand Pillay, and Charles Steinhorn. Definable sets in ordered structures II. *Transactions of the American Mathematical Society*, 295(2) :593–605, 1986.

[KV04] Margarita V. Korovina and Nicolai Vorobjov. Pfaffian hybrid systems. In *Proc. 18th International Workshop on Computer Science Logic (CSL'04)*, volume 3210 of *Lecture Notes in Computer Science*, pages 430–441. Springer, 2004.

[KV06] Margarita V. Korovina and Nicolai Vorobjov. Upper and lower bounds on sizes of finite bisimulations of pfaffian hybrid systems. In *Proc. 2nd Conference on Computability in Europe (CiE'06)*, volume 3988 of *Lecture Notes in Computer Science*, pages 267–276. Springer, 2006.

[Lar95] François Laroussinie. About the expressive power of CTL combinators. *Information Processing Letters*, 54(6) :343–345, 1995.

[LMM02] Salvatore La Torre, Supratik Mukhopadhyay, and Aniello Murano. Optimal-reachability and control for acyclic weighted timed automata. In *Proc. 2nd IFIP International Conference on Theoretical Computer Science (TCS 2002)*, volume 223 of *IFIP Conference Proceedings*, pages 485–497. Kluwer, 2002.

[LMS02] François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. Temporal logic with forgettable past. In *Proc. 17th Annual Symposium on Logic in Computer Science (LICS'02)*, pages 383–392. IEEE Computer Society Press, 2002.

[LPS00] Gerardo Lafferriere, George J. Pappas, and Shankar Sastry. O-minimal hybrid systems. *Mathematics of Control, Signals, and Systems*, 13(1) :1–21, 2000.

[LPZ85] Orna Lichtenstein, Amir Pnueli, and Lenore D. Zuck. The glory of the past. In *Proc. Conference on Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 413–424. Springer-Verlag, 1985.

[LW05] Slawomir Lasota and Igor Walukiewicz. Alternating timed automata. In *Proc. 8th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'05)*, volume 3441 of *Lecture Notes in Computer Science*, pages 250–265. Springer, 2005.

[Mar03a] Nicolas Markey. *Logiques temporelles pour la vérification : expressivité, complexité, algorithmes.* Thèse de doctorat, Laboratoire d'Informatique Fondamentale d'Orléans, France, 2003.

[Mar03b] Nicolas Markey. Temporal logic with past is exponentially more succinct. *EATCS Bulletin*, 79 :122–128, 2003.

[MP71] Robert McNaughton and Seymour Pappert. *Counter-Free Automata.* MIT Press, 1971.

[MPS95] Oded Maler, Amir Pnueli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems. In *Proc. 12th Annual Symposium on Theoretical Aspects of Computer Science (STACS'95)*, volume 900 of *Lecture Notes in Computer Science*, pages 229–242. Springer, 1995.

[OW04] Joël Ouaknine and James Worrell. On the language inclusion problem for timed automata : Closing a decidability gap. In *Proc. 19th Annual Symposium on Logic in Computer Science (LICS'04)*, pages 54–63. IEEE Computer Society Press, 2004.

[OW05] Joël Ouaknine and James Worrell. On the decidability of Metric Temporal Logic. In *Proc. 19th Annual Symposium on Logic in Computer Science (LICS'05)*, pages 188–197. IEEE Computer Society Press, 2005.

[OW06a] Joël Ouaknine and James Worrell. On Metric Temporal Logic and faulty Turing machines. In *Proc. 9th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'06)*, volume 3921 of *Lecture Notes in Computer Science*, pages 217–230. Springer, 2006.

[OW06b] Joël Ouaknine and James Worrell. Safety Metric Temporal Logic is fully decidable. In *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2006.

[OW07] Joël Ouaknine and James Worrell. On the decidability and complexity of Metric Temporal Logic over finite words. *Logical Methods in Computer Science*, 2007.

[PD06] Pavithra Prabhakar and Deepak D'Souza. On the expressiveness of MTL with past operators. In *Proc. 4th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'06)*, volume 4202 of *Lecture Notes in Computer Science*, pages 322–336. Springer, 2006.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *Proc. 18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE Computer Society Press, 1977.

[PR89] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symposium on Principles of Programming Languages (POPL'89)*, pages 179–190. ACM, 1989.

[PS86] Anand Pillay and Charles Steinhorn. Definable sets in ordered structures. *Transactions of the American Mathematical Society*, 295(2) :565–592, 1986.

[PT87] Robert Paige and Robert E. Tarjan. Three partition refinement algorithms. *Siam Journal on Computing*, 16(6) :973–989, 1987.

[Pur98] Anuj Puri. Dynamical properties of timed automata. In *Proc. 5th International Symposium on Formal techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'98)*, volume 1486 of *Lecture Notes in Computer Science*, pages 210–227. Springer, 1998.

[Rab02] Alexander M. Rabinovich. Finite variability interpretation of monadic logic of order. *Theoretical Computer Science*, 275(1-2) :111–125, 2002.

[Ras99] Jean-François Raskin. *Logics, Automata and Classical Theories for Deciding Real-Time*. PhD thesis, University of Namur, Namur, Belgium, 1999.

[RS97] Jean-François Raskin and Pierre-Yves Schobbens. State clock logic : a decidable real-time logic. In *Proc. International Workshop on Hybrid and Real-Time Systems (HART'97)*, volume 1201 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 1997.

[RW89] P.J.G. Ramadge and W.M. Wonham. The control of discrete event systems. *Proc. of the IEEE*, 77(1) :81–98, 1989.

[Sak03] Jacques Sakarovitch. *Éléments de théorie des automates*. Vuibert, 2003.

[SC85] A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3) :733–749, 1985.

[TW96] Denis Thérien and Thomas Wilke. Temporal logic and semidirect products : an effective characterization of the until hierarchy. In *Proc. 37th Annual Symposium on Foundations of Computer Science (FOCS'96)*, pages 256–263. IEEE Computer Society Press, 1996.

[vdD96] Lou van den Dries. O-minimal structures. In *Proc. Logic, From Foundations to Applications*, Oxford Science Publications, pages 137–185. Oxford University Press, 1996.

[vdD98] Lou van den Dries. *Tame Topology and O-Minimal Structures*, volume 248 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 1998.

[Wil94] Thomas Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. In *Proc. 3rd International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'94)*, volume 863 of *Lecture Notes in Computer Science*, pages 694–715. Springer, 1994.

[WTH91] Howard Wong-Toi and Gérard Hoffmann. The control of dense real-time discrete event systems. In *Proc. 30th IEEE Conference on Decision and Control*, pages 1527–1528. IEEE Computer Society Press, 1991.