Spécification et vérification de systèmes hybrides

Riadh Robbana

THESE

présentée par

ROBBANA Riadh

pour obtenir le titre de DOCTEUR
de l'UNIVERSITÉ JOSEPH FOURIER - GRENOBLE 1
(arrêtés ministériels du 5 Juillet 1984 et du 30 Mars 1992)

(Spécialité : INFORMATIQUE)

Spécification et vérification de systèmes hybrides

Date de soutenance : 5 Octobre 1995

Composition du jury :	Président	J. Voiron
	Rapporteurs	A. Arnold
		P. Wolper
	Examinateurs	A. Bouajjani
		M. Nivat
		J. Sifakis

Thèse préparée au sein du Laboratoire VERIMAG

Remerciements

Je tiens à remercier :

Monsieur Jacques Voiron, Professeur à l'université Joseph Fourier de Grenoble , pour avoir présidé le jury de cette thèse ;

Messieurs André Arnold, Professeur à l'Université de Bordeaux I et Pierre Wolper, Professeur à l'institut Montefiore de l'université de Liège, pour avoir accepté de juger ce travail et pour l'intérêt qu'ils lui ont accordé ;

Monsieur Maurice Nivat, Professeur à l'Université Paris VII, qui m'a fait le grand honneur d'être membre du jury ;

Monsieur Joseph Sifakis directeur de recherches au CNRS, pour m'avoir acceuilli au sein du laboratoire qu'il dirige. Je tiens aussi à le remercier d'avoir accepté de faire partie du jury, ainsi que tout l'intérêt qu'il a porté à mes travaux.

Monsieur Ahmed Bouajjani, maître de conférences à l'université Joseph Fourier, qui est le directeur de cette thèse. C'est sous son impulsion que ce travail a commencé, et c'est grâce à nos nombreuses discussions, souvent passionnées mais toujours fructueuses, qu'il a pu aboutir. Je le remercie pour toute la patience dont il a fait preuve lors de ces discussions. Je lui suis profondément reconnaissant.

Monsieur Rachid Echahed, chargé de recherches au CNRS, pour tous les travaux que nous avons effectué ensemble durant mes deux premières années de thèse. Je lui doit beaucoup dans la formation que j'ai pu acquérir durant cette thèse.

Je suis très reconnaissant à monsieur Sergio Yovine, chargé de recherches au CNRS, pour les différentes lectures de ce document, pour son soutien moral et pour tous ses conseils scientifiques dont il m'a fait bénéficier.

J'ai eu le plaisir de travailler avec Yassine Lakhnech durant ma dernière année de thèse. Je tiens à le remercier pour ses différents commentaires qu'il a fait des différentes versions de ce document.

Je remercie très sincèrement Monsieur Laurent Mounier, maître de conférences à l'université Joseph Fourier de Grenoble et responsable des moyens informatiques du laboratoire VERIMAG, pour les multiples reprises où j'ai fait appel à ces services pour palier aux différents problèmes de machine que j'ai pu avoir.

J'ai beaucoup apprécié l'aide et le soutien moral de Mihaela Sighireanu, Ghislaine Thuau, Sadek Ben Salem, Oded Maler, Xavier Nicollin, Alfredo Olivero, Conrado Daws, Sébastien Bornot, Peter Habermehl, David Lesens et Raymond Pascal.

Je remercie Chantal Costes, Sabine Maury et Christine Servonnet qui se sont toujours occupées des problèmes administratifs me concernant.

Je tiens évidemment à remercier Madame Sihem Gmara Elfatmi, Messieurs Ahmed Amine Jerraya et Mohamed Moalla, sans eux je n'aurais pas effectué d'études à Grenoble. Je leur suis très reconnaissant.

Je remercie Sonia, Imed, Ezzedine, Brigitte, Chadlia, Farouk et Ines.

Je suis particulièrement reconnaisant à Narjes Berregeb, Boutheina Chetali, Claude Lemaire, Slim Ben Attalah, Elies Alouini et Mustapha Bouraoui.

Je remercie mes parents.

Table des Matières

Liste des Figures

Chapitre 1

Introduction

1.1 Systèmes hybrides

Les systèmes hybrides sont des systèmes qui combinent des composantes *discrètes* et des composantes *continues*. Les composantes continues peuvent représenter un environnement physique obéissant à des règles de changement continu, tandis que les composantes discrètes peuvent représenter des contrôleurs discrets qui sondent et manipulent les composantes continues en temps-réel. Les systèmes hybrides peuvent aussi être vus comme une composition de systèmes dynamiques dont l'évolution continue de leurs variables est régie par des équations différentielles et d'un contrôleur qui est un ordinateur manipulant des structures de données discrètes. On retrouve cette notion de systèmes hybrides dans de multiples activités telles que le contrôle de processus industriels, dans le domaine des transports (avionique, spatial, $\cdots$) et bien d'autres domaines (multimédia, robotique, $\cdots$).

Ces systèmes peuvent être hautement réactifs et peuvent avoir des conséquences immédiates et irréversibles sur leur environnement. En particulier, toute défaillance même intermittente peut entraîner des dégâts d'un coût prohibitif. Il est donc essentiel de vérifier leur fonctionnement avant leur mise en service. Pour que cette vérification fournisse des résultats fiables, il est indispensable de l'effectuer sur l'ensemble des comportements possibles du système.

1.2 Formalismes

Plusieurs formalismes ont été proposés permettant de raisonner sur les systèmes hybrides. Ces formalismes incluent les réseaux de Petri, les algèbres de processus, les logiques et les automates.

Ces méthodologies permettent d'avoir des informations concernant l'ordre temporel des événements, l'évolution des variables du système dans le temps et les événements observables.

Nous allons rappeler brièvement les deux approches auxquelles nous nous intéressons dans cette thèse. Nous verrons d'une part l'approche basée sur les logiques temporelles puis d'autre part l'approche basée sur les automates.

1.2.1 Approche basée sur les logiques temporelles

L'utilisation de logiques temporelles comme formalisme pour la spécification des comportements des systèmes dans le temps a été proposée par Pnueli en 1977 [Pnu77]. Dès lors le sujet a été largement étudié dans un cadre non temporisé avec des logiques telles que PTL [Pnu77], CTL [CES83], CTL* [EH83], le μ-calcul [Koz83], ECTL* [EGK87, Tho89]. La logique temporelle est une logique modale ayant des modalités telles que le $\Diamond$ signifiant "inévitable" et le $\Box$ signifiant "toujours" (voir [Eme90]), permettant de spécifier de façon naturelle les exigences temporelles. Ces logiques temporelles permettent de raisonner sur la notion de temps logique qui est un temps qualitatif (par opposition au temps quantitatif). Cette notion de temps logique correspond au séquencement des événements dans le temps.
L'extension des logiques temporelles pour prendre en compte la notion de temps continu a donné lieu aux logiques temporisées telles que TPTL, MTL, MITL [Alu91], TCTL ou T_μ [HNSY94]. Ces logiques permmettent de raisonner sur un temps quantitatif qui enregistre la distance entre deux événements. Nous pouvons par exemple grace à cette extension, spécifier la propriété de réponse bornée: "A chaque fois que l'on a une demande on doit alors la satisfaire avant 5 secondes". Une autre extension possible fut de rajouter à ces logiques la notion de durée pour obtenir des logiques telles que CoD(Calculus of Durations) [CHR91] ou DTL (Durations Temporal Logic) [BES93], cette notion de durées permet de raisonner sur l'accumulation de certains temps de séjour liés à certaines phases. Nous pouvons par exemple exprimer pour un nœud intermédiaire dans un réseau de transmission que le temps passé à récupérer les messages perdus ne depasse pas 4% du temps global. Pour pouvoir exprimer une telle propriété nous avons besoin d'une variable particulière qui évolue comme une horloge lors des phases de récupération et qui est arrétée durant les autres phases.
Ces logiques, qui incluent des variables continues, permettent de raisonner sur le temps continu et plus précisément sur l'évolution de certaines variables dans le temps.

1.2.2 Approche basée sur les automates

Cette approche consiste à vérifier des systèmes modélisés par des automates. Son utilisation a commencé dans un cadre non temporisé en utilisant des ω-automates [WVS83, Var87]. Une exécution du système considéré est vue comme un mot infini sur l'alphabet des événements (ou états). Ce qui a établi une connexion très forte entre la vérification et la théorie des langages formels. Un système est modélisé par un automate générant des séquences infinies qui correspondent aux exécutions possibles du système. L'étude des automates sur les mots infinis a commencé par Büchi en étudiant la théorie S1S, théorie monadique du second ordre sur les nombres naturels [Buc62]. Les automates de Büchi ainsi que d'autres variantes telles que les automates de Muller ou de Rabbin ont été largement étudiées dans [Mul63, Rab69]. Il y a eu aussi dans ce cadre l'étude des langages ω-hors-contextes [CG77], qui sont des langages générés par des ω-automates à pile.
Plusieurs extensions de ces automates ont été effectuées, notemment par l'ajout de la notion de temps continu. Cet ajout a donné lieu à des ensembles denses de séquences infinis, dû au fait que le temps est continu. Cette extension a été obtenue en introduisant un certain nombre de variables (horloges) qui évoluent de manière continue et avec la même pente, les automates obtenus sont appelés automates temporisés. Les variables introduites seront utilisées pour restreindre les comportements des systèmes que nous modélisons. Par exemple l'événement e ne peut se produire que si le temps écoulé depuis le début de l'exécution est compris entre 2 et 3 secondes, ou bien que le temps pris pour récupérer un message perdu est compris entre 12 et 18 secondes [Alu91, ACD93].

Une autre extension fut l'introduction de variables dont l'évolution au cours du temps est régie par des équations différentielles. Les automates obtenus sont appelés automates hybrides, ces automates ont été intensivement étudiés dans [MMP92, KPSY93, OSY94, PV94, ACH+95, PHKV95]. Les automates temporisés sont un cas particulier d'automates hybrides.

1.3 Méthodologies de vérification

Plusieurs méthodologies de vérifications existent selon que l'on utilise un même formalisme aussi bien pour spécifier le système que pour exprimer les propriétés à vérifier sur ce système ou bien que l'on utilise un formalisme différent pour chacun. Dans ce qui suit rappelons brièvement certaines de ces méthodologies.

1. **Les systèmes et leurs propriétés sont décrits à l'aide de logiques temporelles:**
   Les logiques temporelles permettent de décrire le système ainsi que ses propriétés

par une formule et la vérification consiste à prouver un théorème dans le système déductif correspondant.
Par exemple, Manna et Pnueli [MP89] ont développés un modèle de systèmes de transitions pour décrire des implémentations et donnent un système de preuve pour vérifier des spécifications exprimées dans une logique temporelle. Mais cette technique est généralement difficile à utiliser, elle nécessite des interventions manuelles afin de construire la preuve.

Il en est de même dans le cadre de la logique de durée CoD, cette logique de durées est une extension de la logique d'intervalles de Moszkowski [Mos85] par la notion de durées sur les propriétés. La durée d'une propriété dans un intervalle est définie comme l'accumulation des instants auxquels cette propriété est vraie. L'approche basée sur le CoD utilise un même formalisme pour représenter les différents comportements et les propriétés d'un système hybride. La vérification s'effectue à l'aide de systèmes de preuves [RRH93, SS94, ZZ94]. Récemment des formes particulières de formules de CoD ont été identifiées pour spécifier les systèmes de contrôle (systèmes hybrides) et leurs exigences (leurs propriétés) [HHF+94, Rav94]. Dans ce fragment de la logique CoD, le fait qu'un système de contrôle spécifique satisfait un ensemble d'exigences est exprimé à l'aide d'implications logiques. La vérification de telles relations d'implications peut être prouvée en utilisant un système déductif pour la validité des formules de CoD [CHR91, RRH93, Rav94]. Mais il n'existe pas de procédure de décision pour cette logique; en effet la plupart des résultats de décidabilité existant pour cette logique en temps dense sont négatifs, le seul résultat positif connu concerne le fragment propositionnel de cette logique (CoD) sans contraintes de temps ni de durées [CHS93].
Certains outils ont été développés autour de ce formalisme [SS94], basés sur le prouveur de théorèmes PVS (Prototype Verification Systems).

2. **Les systèmes et leurs propriétés sont décrits en utilisant l'approche basée sur les automates:**
Dans ce cas, vérifier qu'un système est correcte par rapport à des propriétés spécifiées revient à vérifier que tout comportement généré par l'automate représentant le système est un comportement généré par l'automate représentant les propriétés. Donc le problème de la vérification se ramène au problème d'inclusion de langages. Par conséquent les résultats connus relatifs à la construction de l'intersection d'automates, de leur complémentation et le test du langage vide peuvent être utilisés pour une vérification automatique. L'outil COSPAN développé aux laboratoires Bell est basé sur ce principe [SRK83].
Nous utilisons cette technique dans le chapitre 6 de cette thèse pour effectuer la vérification d'une certaine sous-classe de systèmes hybrides (Graphes à intégrateurs étendus).

Une autre méthode de vérification existe dans le cas où les systèmes et les propriétés modélisés par des automates. Elle consiste à déterminer si les deux automates (celui du système et celui des propriétés) sont équivalents modulo une relation d'équivalence bien particulière, qui est la bisimulation introduite par Milner [Mil89]. L'outil ALDEBARAN développé au laboratoire VERIMAG est basé sur ce principe [FGM+92].

3. **Les systèmes sont spécifiés en utilisant l'approche basée sur les automates tandis que les propriétés sont exprimées dans des logiques temporelles:**
   Il existe plusieurs méthodes qui permettent de vérifier ces propriétés sur l'automate représentant le système.
   La première consiste à transformer ces propriétés en un automate et se ramener alors au problème de la vérification entre deux automates.

   La seconde est basée sur la technique du model-checking, dans cette méthode on évalue des propriétés sur le modèle de sorte à obtenir pour chaque propriété, l'ensemble des états du systèmes qui la satisfont [CE81, QS82, CES83, EL86, CS91, HNSY94, DY95].

   Une troisième technique consiste à coder les propriétés à vérifier dans l'automate représentant le système et réduire alors le problème de la vérification au problème d'atteignabilité dans le nouvel automate obtenu qui contient aussi bien la description du système que celle de ses propriétés, donc au problème du langage vide. C'est la technique que nous utilisons dans le chapitre 5 de cette thèse pour vérifier certaines sous-classes de systèmes hybrides. Les propriétés auxquelles nous nous intéressons dans cette thèse sont des propriétés d'invariances ou d'atteignabilités contraintes. Ce sont des invariants (respectivement des atteignabilités) exprimés à l'aide de contraintes arithmétiques sur les variables du système. Nous montrons comment inclure ces propriétés dans les systèmes que nous considérons, pour réduire le problème à celui de l'atteignabilité sans contraintes (problème du langage vide).

Dans toutes ces méthodologies que nous avons présentés, lorsque les systèmes que considérons sont continus (hybrides), il y a une étape importante dans leur analyse. Cette étape consiste en la réduction d'un espace dense dû à la continuité du temps en un espace discret et fini. Deux approches existent et permettent d'effectuer cette réduction du dense vers le discret. La première est le partitionnement de l'espace modulo une relation d'équivalence qui préserve les propriétés à vérifier [Alu91, ACD93, HNSY94, BR95]. La seconde consiste à remplacer le temps continu par un temps à évolution discrète [HMP92, KPSY93, PV94, BER94b, BER94d]. La seconde étape importante dans l'analyse des systèmes hybrides et qui suit la partie précédente, est la résolution du problème considéré dans le discret. Il est évident que

seule la seconde étape est nécessaire à la vérification des sytèmes non temporisés (discrets).

1.4 Limites de la spécification et la vérification de systèmes hybrides

Pour vérifier des systèmes hybrides, il faut donc au préalable disposer de formalismes permettant d'une part de représenter l'ensemble des comportements temps réels des systèmes qui nous intéressent et d'autre part d'exprimer les propriétés que nous voulons vérifier sur ces systèmes.
Deux approches sont généralement utilisées, la première est basée sur les automates hybrides(automates étendus à l'aide de variables) [MMP92, ACHH93, BES93, ACD93, HNSY94, PV94, OSY94, ACH+95] tandis que la seconde est basée sur les logiques et plus particulièrement sur la logique de durées CoD. De nombreux travaux ont été effectués autour de cette logique [CHR91, CHR93, RRH93, BES93, CL94, Zho94, YPS94, ZZ94, YWZP94].

Le problème de la vérification pour l'approche basée sur le CoD consiste à prouver la validité d'une formule incluant la description du système et les exigences qui lui sont associées. Cette preuve ne peut pas être effectuée de manière automatique. Jusqu'à récemment l'unique résultat de décidabilité pour cette logique, ne concernait qu'un fragment très restreint du CoD, le fragment propositionnel pur. Sinon dans le cas général, il faut une intervention manuelle de la part d'une personne qui doit à la fois maîtriser le système hybride à vérifier et l'outil permettant d'effectuer la preuve.

Pour ce qui est de l'approche basée sur les automates hybrides, le problème de la vérification de propriétés d'atteignabilités est en général indécidable. Plus encore, cette indécidabilité est facilement atteinte. Il faut savoir que le problème de la vérification de propriétés d'atteignabilités sur des automates temporisés est décidable [Alu91, ACD93], alors que ce même problème devient indécidable si l'on ajoute à ces automates deux variables ayant des évolutions linéaires [KPSY93]. Puis récemment Peter Kopke [Kop95b] a montré que si l'on ajoute aux automates temporisés une variable particulière qui est un intégrateur (peut être vue comme une horloge que nous pouvons arréter durant certaines phases), alors le problème d'atteignabilité devient indécidable. Ce résultat a considérablement réduit l'espace séparant les systèmes décidables (systèmes pour lesquels le problème de la vérification de propriétés d'atteignabilité est décidable) de ceux qui sont indécidables.
D'autres sous-classes de systèmes hybrides pour lesquelles ce problème est décidables ont été mises en évidences [ACHH93, BES93, KPSY93, PV94, BER94d, BR95, ACH+95]. Toute extension de l'ensemble des variables de ces systèmes entraînait des

restrictions importantes afin que le problème d'atteignabilité reste décidable.
Par ailleurs, il faut noter que pour tous ces systèmes toutes les variables considérées sont continues. Or dans certains comportements nous avons besoin de variables discrètes; en effet par exemple si nous voulons spécifier les comportements d'un nœud intermédiaire dans un réseau de transmission qui a pour fonction de transmettre tous les messages qu'il reçoit, nous devons assurer qu'à tout moment le nombre de messages reçus est supérieur ou égal au nombre de messages transmis. Nous avons donc besoin d'un compteur qui enregistre la valeur de la différence entre le nombre de messages reçus et le nombre de messages transmis.
Il est vrai que ce compteur et bien d'autres variables discrètes peuvent être simulées à l'aide des variables continues d'un système hybride. Ce procédé n'étant pas naturel, il n'est donc pas pratique pour la spécification.
Il est vrai aussi que les systèmes hybrides permettent de simuler les machines de Turing donc les automates à pile [AMP95], mais cette simulation prend du temps. Donc si on ne s'intéresse qu'au comportement non temporisés (séquences d'états propositionnels) les systèmes hybrides sont expressivement maximaux. Par contre si on considére les comportements temporisés, il n'est pas certain que ce résultat reste valable. De plus la classe des systèmes hybrides qui correspondrait exactement aux automates à pile temporisés (par exemple), si elle existe, elle pourrait-être très compliquée à caractériser.

1.5 Objectifs et réalisations

Etant données ces limites, inférieure et supérieure pour la décidabilité des systèmes hybrides, nous avons eu pour objectifs d'explorer l'espace restant entre ces deux limites. Dans cette thèse nous essayerons d'exhiber de nouvelles sous-classes de systèmes hybrides qui soient plus expressives que les systèmes temporisés et qui soient décidables.
Les systèmes que nous considérons sont des systèmes assez généraux relativement aux modèles récemment proposés dans [MMP92, NSY92, ACHH93, NOSY93]. Globalement, ces travaux considèrent des systèmes ayant uniquement des variables continues et particulièrement, ils considèrent des systèmes hybrides linéaires. Sachant que d'une part les systèmes temporisés sont décidables [ACD93, Alu91] et que d'autre part si nous rajoutons un intégrateur à ces modèles alors les systèmes que nous obtenons sont indécidables [HKPV95, Kop95b]. Nous essayons alors pour notre part d'enrichir les systèmes temporisés par des structures de données discrètes afin d'avoir une meilleure expressivité tout en préservant la décidabilité. Cela nous permettra d'exprimer de manière naturelle des comportements discrets pouvant même être non régulier. En effet, nous pourrons ainsi raisonner sur les occurrences des événements du système. Les comportements obtenus peuvent définir un langage hors-contexte grace à l'introduction d'une pile, ils peuvent même définir un langage sous-contexte grace aux raisonnements

sur les occurrences de certains événements.

De plus dans les travaux existant, après la réduction du dense vers le discret, le problème de la vérification était réduit à l'analyse d'un langage régulier. Par contre, en introduisant des structures de données discrètes non bornées, le problème de la vérification se réduit à l'analyse d'un langage qui n'est en général pas régulier, il est hors-contexte et peut même être sous-contexte. Ce qui établit des liens étroits entre le problème de la vérification de systèmes hybrides et la théorie des les langages, non restreinte dans ce cas aux langages réguliers.
Néanmoins les structures de données discrètes non bornées engendre un nouveau problème: nous n'avons plus une unique source induisant une infinité d'états qui est la continuité du temps, mais une seconde source vient s'y rajouter. Des liens seront donc établis avec l'études des systèmes à espace d'états infini [CS91].
Nous nous intéressons aussi aux liens (de simulations) entre les systèmes à variables discrètes et systèmes à variables continues. Lorsque ces liens peuvent être établis, l'analyse de systèmes hybrides peut être réduite à l'analyse de systèmes discrets. Ce qui entraînera le raisonnement sur les délais et les durées à un raisonnement sur les occurrences d'événements. Puisque l'indécidabilité du problème de la vérification pour les automates hybrides est très vite obtenue par le rajout d'une variable continue qui n'est pas une horloge, nous pouvant alors utiliser dans certains cas les structures de données discrètes que nous introduisons pour spécifier d'une façon approximative le comportement de certaines composantes continues.

De plus si on essaye de simuler un compteur non borné à l'aide de variables continues, il faut qu'une des variables continues prenne ses pentes dans l'ensemble $\{-1, 0, 1\}$ (cette variable est appelée intégrateur étendu). Nous verrons que pour pouvoir analyser de tels systèmes nous avons besoin d'imposer des restrictions très fortes. En effet nous verrons que ces systèmes doivent avoir au plus deux variables dont un seul intégrateur étendu et l'autre ne pouvant être qu'un intégrateur ou une horloge pour que le problème d'atteignabilité soit décidable.

Les méthodes automatiques ne s'appliquant pas au formalisme basé sur la logique de durées CoD, par conséquent notre objectif consiste aussi à rechercher des méthodes de vérification automatique pour cette logique.
Nous nous interessons particulièrement au fragment de CoD qui décrit dans [HHF+94, Rav94] comme servant à spécifier des systèmes de contrôle et leurs exigences. La plupart des systèmes récemment étudiés sont spécifié dans ce fragment.
Nous verrons que non seulement nous exhibons un fragment de CoD à temps dense qui soit décidable, mais que de plus, le fragment que nous exhibons inclu le fragment décrit dans [HHF+94, Rav94].

Nous avons eu pour perspectives d'établir des liens entre les deux formalismes, celui qui est basée sur la logique de durées CoD et celui qui est basée sur les auto-

mates hybrides alors que jusqu'à là aucune liaison ne fût établie dans un cadre dense. Etablir une liaison entre ces deux formalismes était considéré comme un problème complexe, cela était dû aux sémantiques adoptées pour chacun d'eux. Pour le CoD, le raisonnement est basé sur les intervalles de longueurs non nulles, tandis que pour les automates hybrides le raisonnement est basé sur les points (instants).
Nous verrons dans cette thèse que pour établir une liaison entre ces deux formalismes nous définirons une logique qui sert de forme intermédiaire entre eux.
Nous développerons une méthode automatique de traduction des formules vers les automates hybrides, pour un fragment important du CoD. Cette traduction nous permettra d'identifier un fragment important du CoD, pour lequel le problème de la vérification est décidable. Alors que seul le fragment propositionnel pur a pu être montré décidable, nous exhibons le premier fragment à temps dense pour lequel le problème de la vérification est décidable.
De plus les liens que nous établirons entre les deux approches de spécifications de systèmes hybrides nous permettra d'utiliser des méthodes d'analyse existantes dans les deux approches.
Une particularité très intéressante du fragment décidable que nous allons extraire est qu'il subsume le fragment proposé dans [HHF+94, Rav94] qui permet de spécifier des systèmes de contrôle et leurs exigences.

1.6 Organisation du document

Le **chapitre 2** a pour but d'introduire les différents modèles que nous considérons dans cette thèse. Nous évoquons les principales définitions et notations qui sont en liaison avec ces modèles et qui seront utilisées par la suite.

Dans le **chapitre 3** nous présentons les différentes propriétés que nous essayons de vérifier sur les systèmes hybrides que nous considérons.
Dans une première partie nous définissons les propriétés d'invariance contraintes. Nous montrons aussi que la vérification de ces propriétés est réductible à la vérification de propriétés d'atteignabilité purement propositionnelle.
Puis dans la seconde partie nous développons la description des propriétés ω-régulières non temporisées que nous voulons vérifier sur des systèmes très particuliers, ces systèmes se situent à la limite des sous-classes de systèmes pour lesquelles le problème de la vérification est décidable. Nous verrons que la décidabilité du problème de la vérification de ses systèmes est très sensible au nombre de variables que comportent ses systèmes; c'est la raison pour laquelle nous essayons de vérifier sur ces systèmes des propriétés non temporisées car elles n'engendrent pas d'augmentation du nombre des variables du systèmes contrairement aux propriétés temporisées.

Dans le **chapitre 4** nous décrivons les principes et techniques de réduction de la

vérification d'un domaine dense vers un domaine discret; nous commençons par le principe du partitionnement [ACD93, BR95] pour finir par le principe de la discrétisation [HMP92, BER94d].

Dans le **chapitre 5** nous adoptons l'approche de spécification basée sur les automates. Dans ce chapitre nous prenons des modèles permettant de spécifier des systèmes hybrides linéaires, nous considérons des graphes d'états de contrôle finis munis d'une part de structures de données discrètes (compteurs, piles, etc) et d'autre part de variables réelles changeant de manière continue (plus précisément de manière linéaire) [BER94a, BER95]. Dans ce cadre nous montrons que nous pouvons extraire des sous-classes de tels systèmes pour lesquels la vérification est décidable.
Puis dans le **chapitre 6** nous nous penchons sur le problème de la vérification de propriétés ω-régulières sur une sous-classe de système hybride [BR95]. Cette sous-classe est certes restreinte mais l'avantage de son étude est que d'une part sa réduction d'un domaine dense à un domaine discret nous plonge dans une structure hors-contexte comme c'est le cas des sous-classes présentés dans le chapitre 5. Puis d'autre part nous donnons une technique de partitionnement assez fin qui préserve l'équivalence des traces sans pour autant induire de bisimulation finie.

Dans le **chapitre 7** nous considérons l'approche basée sur les logiques et plus particulièrement celle basée sur la logique de durées (CoD) introduite dans [CHR91]. Nous établissons dans ce chapitre des liens entre cette approche et celle basée sur les automates; pour cela nous introduisons une nouvelle logique de durées "DIL" qui sert d'intermédiaire entre les deux approches [BLR95]. Nous montrons comment cette liaison entre les deux approches nous permet d'extraire un important fragment de la logique de durées CoD pour lequel le problème de vérification est décidable.

Chapitre 2

Les Modèles

Dans ce chapitre nous introduisons de manière formelle la définition et la sémantique des différents modèles relatifs à l'approche basée sur les automates que nous considérons dans cette thèse.
Les systèmes hybrides auxquels nous nous intéressons dans cette thèse sont d'une part linéaires, c'est-à-dire que toutes les variables sont continues et ont des pentes d'évolution qui sont des constantes entières; puis d'autre part munis de structures de données discrètes, telles qu'une pile et des compteurs.
Notre apport a été de munir les systèmes hybrides linéaires de structures de données discrètes. Nous commençons ce chapitre en donnant un exemple de systèmes hybrides linéaires sans structures de données discrètes; comme exemple nous donnons celui du brûleur à gaz introduit dans [CHR91].
Par la suite nous donnons la définition et la sémantique des systèmes hybrides linéaires munis de structures de données discrètes. Nous décrivons plusieurs sous-cas de tels systèmes ainsi qu'un exemple illustrant chacun d'eux. Ces sous-cas sont intéressants dans la mesure où d'une part c'est pour ces classes que nous avons établi des résultats de décidabilités et puis d'autre part ils sont assez proches des systèmes pour lesquels le problème de la vérification est indécidable.

2.1 Les systèmes hybrides linéaires munis de structures de données discrètes

Nous commençons dans cette partie par choisir des modèles convenant aux systèmes hybrides, c'est à dire des modèles résultants de la composition de modèles de systèmes dynamiques (systèmes d'équations différentielles) avec des modèles de machines discrètes (automates). Par conséquent les modèles adéquats aux systèmes hybrides

peuvent être obtenus en considérant des graphes d'états de contrôle finis muni de deux sortes de structures de données, d'une part des structures de données discrètes (compteurs, piles, etc) dont les variations coïncident avec l'évolution d'un état de contrôle vers un autre et d'autre part des variables réelles changeant de manière continue dans chaque état de contrôle ou de manière discrète lors du changement de l'état de contrôle. Les changements d'état de contrôle sont conditionnés par des contraintes sur les valeurs ou les configurations des variables aussi bien discrètes que continues et des structures de données du système (par exemple: l'état d'une pile).

Suivant cette vision mais ne tenant pas compte des structures de données discrètes, plusieurs modèles de systèmes hybrides ont été récemment proposés dans [MMP92, KPSY93, MP93, ACHH93, NOSY93, AD94, ACH+95]. Globalement, ces travaux considèrent uniquement des variables continues et plus particulièrement des systèmes hybrides dont l'évolution de chacune de leurs variables est définie par une dérivée égale à une constante entière. Ces systèmes sont généralement appelés systèmes hybrides linéaires.

Pour notre cas nous avons munis ces systèmes de structures de données discrètes, telles qu'une pile et des compteurs. Notre étude est restreinte aux systèmes hybrides linéaires car ce sont les seuls systèmes pour lesquels des résultats de décidabilité ont été établis. Dans ce qui suit nous définissons les systèmes hybrides contenant des variables continues linéaires, des variables discrètes (compteurs) et une pile. Notre intérêt se porte sur ces systèmes car ce sont des systèmes plus expressifs que les systèmes ne comportant pas de structures de données discrètes et pour lesquels certains résultats de décidabilités sont préservés. Les systèmes que nous considérons sont en plus à la frontière entre les systèmes pour lesquels le problème de la vérification est décidable et les systèmes pour lesquels ce même problème est indécidable.

Avant de présenter le modèle des systèmes que nous considérons, nous donnons un exemple d'introduction qui est un système hybride linéaire:

Exemple 2.1 Exemple de motivation : Le brûleur à gaz

L'exemple du brûleur à gaz a été introduit dans [CHR91]. C'est un système qui passe alternativement d'une phase dans laquelle il y a une fuite de gaz à une phase sans fuite de gaz.
Nous supposons que chaque période de fuite de gaz n'excède jamais une unité de temps.
Nous supposons aussi qu'entre deux périodes de fuite de gaz il s'écoule au moins 30 unités de temps.
Nous aimerions vérifier sous ces suppositions que pour tout intervalle de longueur supérieure à 150 l'accumulation des durées des périodes de fuite de gaz est au plus égale à 4% de la longueur de l'intervalle.

Le système est représenté dans la figure 2.1 et sa description est la suivante. Le système comporte trois nœuds de contrôles ℓ_0, ℓ_1 et ℓ_2. Il y a la proposition *fuite* qui

n'est vraie que dans le nœud ℓ_1 et qui représente la situation durant laquelle le système est dans la phase de fuite de gaz.

Dans la figure nous utilisons trois variables dont la description est comme suit:
La variable x mesure le temps passé à chaque visite dans un des nœuds ℓ_0, ℓ_1 ou ℓ_2. La variable x est donc remise à zéro à chaque entrée dans un de ces trois nœuds.
La variable y mesure le temps accumulé de fuite. La variable y croît de manière linéaire dans le nœud ℓ_1 et reste inchangée dans les autres nœuds.
La variable z mesure le temps total écoulé.

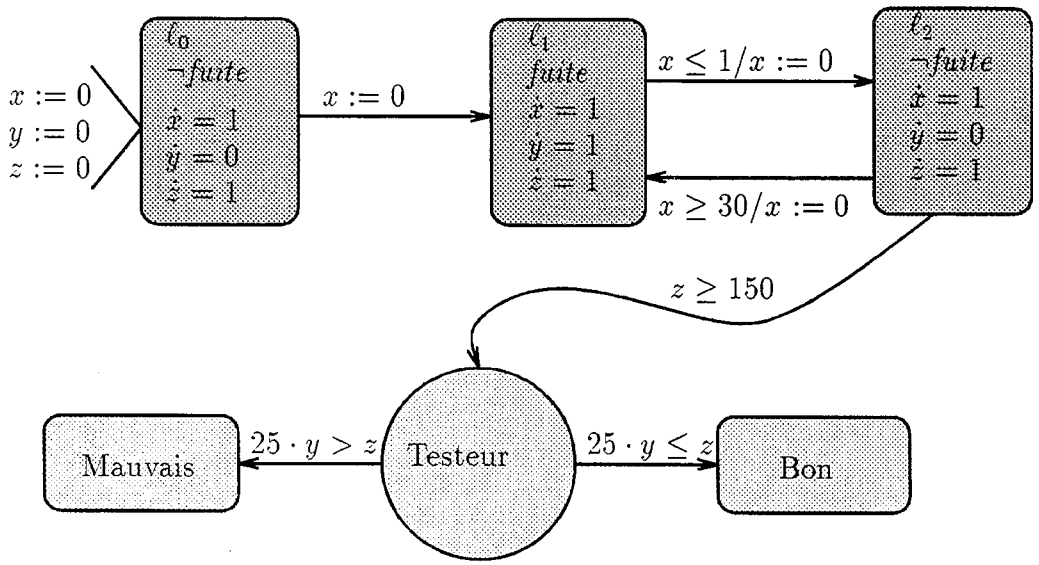


Figure 2.1: Le brûleur à gaz

Nous pouvons voir les trois nœuds de contrôle ℓ_0, ℓ_1 et ℓ_2 comme la partie opérative de la représentation et les autres nœuds servent à tester si la propriété que nous aimerions vérifier sur le système est satisfaite ou non. Le système peut quitter la partie opérative si le temps global écoulé dépasse 150 unités; à la sortie le système, effectue un test au niveau du nœud "Contrôleur". Si $25 \cdot y > z$ alors la durée accumulée des périodes de fuite de gaz est supérieure à 4% du temps global, le système passe alors dans le nœud "Mauvais" qui signifie que la propriété est violée. Sinon le système passe dans le nœud "Bon" signifiant que pour une telle exécution la propriété n'est pas violée. □

2.1.1 Définition et sémantique

Nous donnons la définition des systèmes hybrides que nous considérons en présentant l'introduction des structures de données discrètes dans les systèmes hybrides linéaires. A l'aide de cette introduction de structures de données discrètes nous pouvons d'une part raisonner sur les occurrences de certains événements puis d'autre part avoir des comportements non réguliers. Avant de donner la définition de ces systèmes hybrides nous introduisons la notion de contraintes linéaires qui interviennent dans l'expression des *invariants* associés à un nœud de contrôle du système ainsi que dans l'expression des conditions d'activation des transitions du système (*gardes*).

Définition 2.1 Contraintes linéaires

Soit $\mathcal{V}$ l'ensemble des variables continues ou discrètes de notre système hybride. Une contrainte linéaire sur $\mathcal{V}$ est la combinaison booléenne de contraintes élémentaires qui sont soit de la forme $x \sim c$ où $x \in \mathcal{V}$, c une constante entière ($c \in \mathbb{Z}$) et $\sim \in \{<, \leq\}$; soit de la forme $x \equiv_n c$ où $x \in \mathcal{V}$, $n \in \mathbb{N} - \{0\}$ et c est une constante naturelle ($c \in \mathbb{N}$). Les symboles $<$ et $\leq$ représentent les relations d'ordre usuelles (strictes et larges) sur les réels. Pour tout $n \in \mathbb{N} - \{0\}$ $\equiv_n$ est une relation entre les réels et les réels positifs défini comme suit: $\forall r_1 \in \mathbb{R}$, $\forall r_2 \in \mathbb{R}_{\geq 0}$, $r_1 \equiv_n r_2$ si et seulement si $0 \leq r_2 < n$ et $\exists q \in \mathbb{Z}$, $r_1 = q.n + r_2$. Pour toute variable x, on écrit $int(x)$ au lieu de $x \equiv_1 0$. Nous notons par *vraie* la contrainte définie par $\forall x \in \mathcal{V}\ x > 0 \vee x \leq 0$. Soit $\mathcal{C}_\mathcal{V}$ l'ensemble des contraintes linéaires sur $\mathcal{V}$. □

Soient $\mathcal{P}$ un ensemble fini de propositions atomiques et $\Sigma = 2^\mathcal{P}$, alors un système hybride $\mathcal{H}$ sur Σ est défini comme suit:

Définition 2.2 Un *système hybride* est un tuple $\mathcal{H} = \langle \mathcal{L}, \Pi, \mathcal{X}, \partial, \mathcal{Y}, \Gamma, I, \delta \rangle$ où :

- $\mathcal{L} = \{\ell_1, \cdots, \ell_n\}$ est un ensemble fini de nœuds de contrôle.
- Π est une fonction dans $[\mathcal{L} \mapsto \Sigma]$ associant un ensemble de propositions atomiques à chaque nœud de contrôle.
- $\mathcal{X}$ est un ensemble fini de variables réelles, qui sont des fonctions continues du temps.
- ∂ est une fonction dans $[\mathcal{L} \times \mathcal{X} \mapsto \mathbb{Z}]$ associant à chaque nœud de contrôle ℓ et chaque variable continue x une pente à laquelle cette variable change continuellement durant le séjour dans ce nœud de contrôle.

- $\mathcal{Y}$ est un ensemble fini de variables entières, qui évoluent de manière discrète lors du changement de nœud de contrôle (par l'addition ou la soustraction d'une constante entière). On appelle "compteurs" les éléments de $\mathcal{Y}$.

- Γ est un ensemble fini qui contient le vocabulaire de la pile.

- I est une fonction dans $[\ell \mapsto \mathcal{C}_{\mathcal{X}\cup\mathcal{Y}}]$ qui associe à chaque nœud de contrôle des contraintes sur les variables du système. C'est l'invariant $I(\ell)$ du nœud ℓ. Ces contraintes doivent être satisfaites tout au long de chaque séjour dans ce nœud ℓ.

- δ est l'ensemble d'arcs. Chaque arc étant un tuple $e = (\ell, g, A, \gamma, \kappa, R, \ell')$ où:

  - $\ell, \ell' \in \mathcal{L}$ sont le nœud source et le nœud cible.
  - $g \in \mathcal{C}_{\mathcal{X}\cup\mathcal{Y}}$ est une contrainte linéaire qui doit être satisfaite lorsque la transition e est effectuée.
  - $A \in \Gamma$ la tête de pile doit être A afin de pouvoir effectuer la transition e.
  - $\gamma \in \Gamma^*$ est une séquence de symboles remplaçant la tête de pile A. Ce remplacement est opéré une fois que la transition e est effectuée.
  - κ est une fonction dans $[\mathcal{Y} \mapsto \mathbb{Z}]$ associant à chaque compteur une valeur entière qui devra lui être rajoutée.
  - $R \subseteq \mathcal{X} \cup \mathcal{Y}$ est l'ensemble de variables qui devront être remises à zéro lorsque la transition e est effectuée.

□

Dans ce qui suit nous notons par **SHP** (pour dénoter les systèmes hybrides à pile) ces systèmes hybrides munis de structures de données discrètes.

Suite à cette définition des modèles que nous considérons, nous introduisons des notions très importantes relatives aux variables du système. Les variables d'un système hybride peuvent être classées selon leur utilité dans deux classes disjointes. La première classe contiendra les variables de contrôle qui ont pour fonction de contrôler le système en interdisant certains comportements, cela se fera en faisant intervenir ces variables dans les gardes du système ou dans les invariants associés aux nœuds de ce système. La seconde classe est celle qui contient les variables dites d'observation, lesquelles servent à exprimer des propriétés à vérifier (telles que des invariants). Ces variables dites d'observations, ont pour faculté d'enregistrer quantitativement les exécutions d'un système.

Définition 2.3 (Variables de Contrôle)
Toute variable qui intervient soit dans l'expression d'une garde d'un système hybride soit dans l'expression d'un invariant associé à un nœud de ce système et qui n'intervient pas dans les propriétés à vérifier est appelée variable de contrôle. □

Remarque 2.1 Dans les systèmes que nous considérons et qui sont munis d'une pile, la pile peut être vue comme une variable de contrôle. En effet l'état de la tête de pile restreint certains comportements. Le passage d'un nœud de contrôle vers un autre peut être conditionné par la valeur de la tête de la pile. □

La seconde classe est celle des variables d'observation, qui subissent l'influence du comportement du système et interviennent dans l'expression des invariants que nous voulons vérifier sur ce système. C'est à l'aide de ces variables que nous exprimons les contraintes des propriétés d'invariance.

Définition 2.4 (Variables d'observation)
Toute variable qui intervient dans l'expression d'une propriété à vérifier et qui n'est pas une variable de contrôle est une variable d'observation. □

Nous définissons dans ce qui suit les notions d'horloge, d'intégrateur et d'intégrateur étendu.

Définition 2.5 Horloge
Nous appelons *horloge* toute variable continue ayant une pente d'évolution égale à 1 dans tous les nœuds du système. C'est-à-dire:

$$x \in \mathcal{X} \text{ est une horloge si et seulement si } \forall \ell \in \mathcal{L} \;\; \partial(\ell, x) = 1$$

□

Nous définissons dans ce qui suit la notion d'intégrateur, qui est une variable continue qui peut être vue comme une horloge que nous arrêtons dans certains nœuds.

Définition 2.6 Intégrateur
Nous appelons *intégrateur* toute variable continue ayant une pente d'évolution dans l'ensemble $\{0, 1\}$ dans tous les nœuds du système. C'est-à-dire:

$x \in \mathcal{X}$ est un intégrateur si et seulement si $\forall \ell \in \mathcal{L}\ \partial(\ell, x) \in \{0, 1\}$

□

Définition 2.7 Intégrateur étendu
Nous appelons *intégrateur étendu* toute variable continue ayant une pente d'évolution dans l'ensemble $\{-1, 0, 1\}$ dans tous les nœuds du système. C'est-à-dire:

$x \in \mathcal{X}$ est un intégrateur étendu si et seulement si $\forall \ell \in \mathcal{L}\ \partial(\ell, x) \in \{-1, 0, 1\}$

□

Lors de la définition d'un système hybride il faut spécifier l'influence que le séjour dans un nœud de contrôle a sur les variables aussi bien de contrôle que d'observation (c'est à dire qu'il faut donner la loi d'évolution de ces variables dans ce nœud). Il faut aussi définir l'influence que le fait d'effectuer une transition a sur les variables de contrôle et d'observation, influence due aux remises à zéro et aux différentes additions d'entiers aux variables discrètes.

Avant de donner la sémantique des systèmes que nous considérons, nous donnons dans ce qui suit un exemple de système hybride linéaire muni de structures de données discrètes. C'est l'exemple du nœud intermédiaire dans un réseau de transmission. Cet exemple permet d'illustrer la nécessité de l'utilisation d'une pile et des compteurs. C'est aussi un exemple que nous reproduirons à plusieurs reprises dans ce chapitre afin d'illustrer les sous-classes que nous distinguons de systèmes hybrides linéaires munis de structures de données discrètes.

Exemple 2.2 L'exemple suivant permet de tenir compte de certaines contraintes de transmission lors de la modélisation du comportement d'un nœud intermédiaire dans un réseau. Nous supposons que la politique de transmission du noeud intermédiaire est la suivante:
Sachant que nous appelons session du nœud intermédiaire la phase de traitement d'un message par ce nœud et qu'un message est divisé en plusieurs paquets, le nœud intermédiaire a pour fonction de transmettre tous les paquets qu'il reçoit. Lors de la perte d'un paquet (par exemple lorsque le paquet reçu est différent de celui qui est attendu), le système exécute une procédure spéciale afin de récupérer ces paquets perdus. Nous supposons que chaque réception prend entre 1 et 2 unités de temps et que chaque transmission prend entre 2 et 3 unités de temps. Nous supposons aussi qu'entre deux pertes de paquets il s'écoule au moins un temps de 50 unités et que

la procédure de recouvrement prend entre 2 et 4 unités. Finalement nous supposons qu'une session prend au moins 240 unités de temps.

Nous aimerions vérifier que le système décrit vérifie les propriétés suivantes.

1. *Pour toute session, le temps passé à récupérer les paquets perdus est inférieur à 10 % du temps global de la session,*
2. *Pour toute session, il y a exactement le même nombre de réceptions et de transmissions,*
3. *durant toute session, le nombre de transmissions ne dépasse jamais le nombre de réceptions.*

Pour modéliser le système décrit ci-dessus, nous utilisons un système temporisé muni de structures de données discrètes (voir figure 2.2). C'est-à-dire un SHP ayant uniquement des horloges comme variables continues. Ce système est décrit comme suit:

Il possède deux nœuds de contrôle T et R en plus des nœuds de début D et de fin F. Le système utilise trois horloges x, y et t ainsi qu'un compteur c. L'horloge x est utilisée pour exprimer les hypothèses relatives aux réceptions et aux transmissions. L'horloge y est utilisée pour exprimer les contraintes supposées sur la fréquence des pertes de paquets et sur le délai de la procédure de récupération des paquets perdus. L'horloge t est utilisée pour enregistrer le temps global écoulé depuis le début de la session. Le compteur c donne le nombre de paquets présents (reçus mais non encore transmis). Il est à noter que y est initialisée à 50 pour modéliser le fait qu'il n'y a pas de contraintes sur le premier paquet perdu. Lorsque l'exécution est dans le nœud T, le système peut soit recevoir un paquet (transition REC), soit transmettre un paquet (transition TRANS) à condition que le système soit en possession de paquets reçus qu'il doit transmettre. Lorsque le système reçoit un paquet erroné le compteur c est décrémenté, ce qui a pour signification que le paquet n'est pas considéré et que le contrôle passe au nœud R qui correspond à la phase de récupération. Lorsque le nœud R est quitté (le paquet est récupéré) le compteur c est incrémenté.

Toutefois, nous avons besoin d'introduire de nouvelles variables pour pouvoir vérifier certaines propriétés du système considéré. En effet pour ce qui est des trois propriétés (1), (2) et (3) à vérifier , nous aurons besoin d'introduire deux variables d'observation: Un accumulateur u (variable discrète) qui contiendra la différence entre le nombre de réceptions (transitions REC) et le nombre de transmissions (transition TRANS). La seconde variable d'observation est une variable continue z qui évolue selon la pente $10 \cdot \Delta_R - t$, où Δ_R est la durée de R, c'est à dire le temps accumulé durant lequel le contrôle du système est dans le nœud de contrôle R. Ainsi lorsque le contrôle est dans le nœud T (respectivement R), la variable z décroît (respectivement

croît) avec une pente de -1 (resp. $+9$). Les évolutions discrètes ou continues de ces variables d'observations sont représentées dans la figure 2.2.

C'est à l'aide de ces variables d'observations que nous pouvons exprimer les différentes propriétés à vérifier, ces variables auront pour rôle d'enregistrer certaines informations nécessaires à cette vérification durant une exécution donnée.

Nous verrons dans le chapitre 3 comment les propriétés (1), (2) et (3) sont exprimées à l'aide des variables u et z.

Nous pouvons voir que le système décrit par la figure 2.2 peut être facilement défini comme un automate à pile en utilisant les transformations classiques d'une machine à un compteur vers un automate à pile, où la pile est utilisée pour représenter le compteur c.

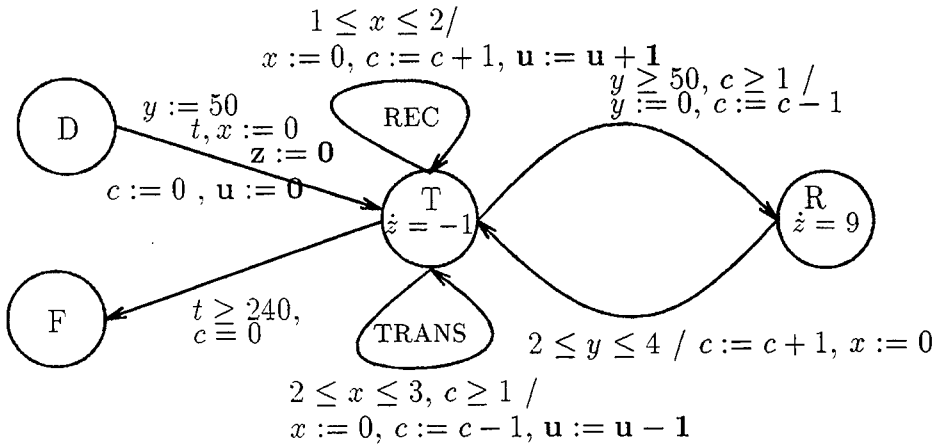


Figure 2.2: Nœud intermédiaire dans un réseau de transmission

□

Avant de donner la sémantique des modèles que nous considérons nous introduisons quelques définitions et notations. Nous commençons par définir la notion de valuation des variables du système.

Définition 2.8 Valuation
Une valuation des variables du système $\mathcal{H}$ est définie par le couple $\langle \nu, \mu \rangle$. Avec ν et μ deux fonctions définies comme suit:

- Le premier terme du couple est la valuation des variables continues du système. $\nu : \mathcal{X} \mapsto \mathbb{R}$.

- Le deuxième terme du couple étant la valuation des variables discrètes du système.
  $\mu : \mathcal{Y} \mapsto \mathbb{Z}$.

□

Définition 2.9 Valuation entière
On dit qu'une valuation est entière si $\forall x \in \mathcal{X}\ \nu(x) \in \mathbb{Z}$. □

Puis nous introduisons quelques notations relatives à cette notion de valuation.

Notation 2.1.1 *Etant donnée une valuation ν sur $\mathcal{X}$, un noeud de contrôle ℓ et $t \in \mathbb{R}_{\geq 0}$ nous notons par $[\nu + t]_\ell$ la valuation ν' telle que $\forall x \in \mathcal{X}.\ \nu'(x) = \nu(x) + \partial(l, x) \cdot t$, c'est à dire la valuation des variables obtenue à partir de ν en séjournant dans le noeud ℓ pendant une durée égale à t.*
Nous notons par $\nu + t$ la valuation ν' telle que $\forall x \in \mathcal{X}.\ \nu'(x) = \nu(x) + t$.

Notation 2.1.2 *Etant données une valuation μ sur $\mathcal{Y}$, et une fonction $\kappa \in [\mathcal{Y} \mapsto \mathbb{Z}]$, nous notons par $\mu + \kappa$ la valuation $\mu' \in [\mathcal{Y} \mapsto \mathbb{Z}]$ telle que $\forall y \in \mathcal{Y}.\ \mu'(y) = \mu(y) + \kappa(y)$.*

Notation 2.1.3 *Pour tout $X \subseteq \mathcal{X}$, nous notons par $\nu[X \mapsto 0]$ la valuation qui associe à chaque variable $x \in X$ la valeur 0 et coïncide avec ν pour toutes les autres variables.*

Notation 2.1.4 *Pour tout $Y \subseteq \mathcal{Y}$, nous notons par $\mu[Y \mapsto 0]$ la valuation qui associe à chaque variable $y \in Y$ la valeur 0 et coïncide avec μ pour toutes les autres variables.*

Nous pouvons maintenant donner la sémantique des systèmes hybrides. Plusieurs sémantiques ont été proposées dans [CHR93, NK93, MP93, MMP92, Lam93, Hoo93, ACH+95]. Nous allons commencer par donner la sémantique que nous considérons de manière informelle puis nous la donnerons formellement.
Soit $\mathcal{H} = \langle \mathcal{L}, \Pi, \mathcal{X}, \partial, \mathcal{Y}, \Gamma, I, \delta \rangle$ un système hybride.

- Une configuration du système hybride $\mathcal{H}$ est déterminée par un nœud $\ell \in \mathcal{L}$, la valuation (ν, μ) de ses variables et la séquence $\lambda \in \Gamma^*$ représentant la pile.
  Nous notons une configuration comme un quadruplet $\langle \ell, \lambda, \nu, \mu \rangle$.
  Nous appelons configuration temporisée du système hybride $\mathcal{H}$ le quintuplet $\langle \ell, \lambda, \nu, \mu, t \rangle$ où $\langle \ell, \lambda, \nu, \mu \rangle$ est une configuration de $\mathcal{H}$ et $t \in \mathbb{R}_{\geq 0}$ est l'estampille représentant le temps global écoulé depuis le début de la séquence d'exécution considérée de $\mathcal{H}$.

- L'arc $e = (\ell, g, A, \gamma, \kappa, R, \ell')$ peut être franchi à partir de la configuration $\langle \ell, \lambda, \nu, \mu \rangle$ si la tête de pile est A et la valuation (ν, μ) satisfait la garde g. La configuration résultante $\langle \ell', \lambda', \nu', \mu' \rangle$ est telle que l'affectation est satisfaite, c'est à dire que $\nu' = \nu[R \cap \mathcal{X} \mapsto 0]$, $\mu' = \mu[R \cap \mathcal{Y} \mapsto 0]$ et λ' est obtenue à partir de λ en remplaçant la tête de pile A par la séquence γ. La valuation (ν', μ') satisfait l'invariant $I(\ell')$
- Le franchissement d'un arc est instantané.
- Dans chaque sommet les valeurs des variables continues changent avec le passage du temps en respectant la relation d'évolution ∂.
- Le système peut séjourner dans un sommet ℓ tant que l'invariant $I(\ell)$ du sommet est satisfait par la valuation.

Nous définissons la sémantique des systèmes hybrides comme un ensemble de séquences d'exécution temporisées. Pour cela nous définissons le séquencement entre configurations temporisées.

Définition 2.10 Séquence d'exécution temporisée
Une séquence d'exécution temporisée de $\mathcal{H}$ commençant à une configuration donnée $\langle \ell, \lambda, \nu, \mu \rangle$ est une séquence infinie de configurations temporisées $\{\langle \ell_i, \lambda_i, \nu_i, \mu_i, \tau_i \rangle\}_{i \in \mathbb{N}}$ telle que:

- $\langle \ell, \lambda, \nu, \mu, 0 \rangle = \langle \ell_0, \lambda_0, \nu_0, \mu_0, \tau_0 \rangle$,
- $\forall i \in \mathbb{N}.\ \tau_{i+1} \geq \tau_i$,
- $lim_{i \mapsto \infty} \tau_i = \infty$, et
- $\forall i \in \mathbb{N}.$
  - Soit $\ell_i = \ell_{i+1}$, $\lambda_{i+1} = \lambda_i$, $\nu_{i+1} = [\nu_i + (\tau_{i+1} - \tau_i)]_{\ell_i}$, $\mu_{i+1} = \mu_i$ et $\forall t \in [0, \tau_{i+1} - \tau_i]\ [\nu_i + t]_{\ell_i} \models I(\ell_i)$
  - Soit $\tau_i = \tau_{i+1}$ et $\exists (\ell_i, g, A, \gamma, \kappa, R, \ell_{i+1}) \in \delta.\ (\nu_i, \mu_i) \models g$, $\lambda_i = A \cdot \lambda'$, $\lambda_{i+1} = \gamma \cdot \lambda'$, $\nu_{i+1} = \nu_i[(R \cap \mathcal{X}) \mapsto 0]$, $\mu_{i+1} = (\mu_i + \kappa)[(R \cap \mathcal{Y}) \mapsto 0]$ et $\nu_{i+1} \models I(\ell_{i+1})$.

□

De manière similaire nous définissons les séquences d'exécution qui consistent simplement à faire abstraction du temps global écoulé des séquences d'exécution temporisées depuis le début de ces séquences, c'est à dire qu'il faut faire abstraction des τ_i. Nous obtenons la définition suivante:

Définition 2.11 Séquence d'exécution
Une séquence d'exécution temporisée de $\mathcal{H}$ $\{\langle \ell_i, \lambda_i, \nu_i, \mu_i, \tau_i \rangle\}_{i \in I\!N}$ commençant à la configuration $\langle \ell, \lambda, \nu, \mu \rangle$ génère une séquence d'exécution de $\mathcal{H}$ commençant à la configuration $\langle \ell, \lambda, \nu, \mu \rangle$ et qui est définie comme la séquence infinie de configurations $\{\langle \ell_i, \lambda_i, \nu_i, \mu_i \rangle_{i \in I\!N}$ telle que: $\langle \ell, \lambda, \nu, \mu \rangle = \langle \ell_0, \lambda_0, \nu_0, \mu_0 \rangle$ □

Notation 2.1.5 *Etant donnée une configuration α, nous notons par $\mathcal{SET}(\alpha, \mathcal{H})$ l'ensemble des séquences d'exécution temporisées de $\mathcal{H}$ commençant à partir de α. De manière similaire nous notons $\mathcal{SE}(\alpha, \mathcal{H})$ l'ensemble des séquences d'exécution de $\mathcal{H}$ commençant à partir de α.*

Il est a noter que toute séquence d'exécution (temporisée) génère une trace (temporisée) qui est obtenue en faisant abstraction de la pile ainsi que des valuations des variables. Cette notion de trace est définie comme suit:

Définition 2.12 Trace temporisée
Nous définissons une trace temporisée comme une séquence infinie $\{\langle \ell_i, \tau_i \rangle\}$, telle que $\tau_0 = 0$, $\forall i \in I\!N.\ \tau_{i+1} \geq \tau_i$ et $lim_{i \to \infty} \tau_i = \infty$.
Nous obtenons la trace en faisant abstraction du temps de la trace temporisée.
La trace temporisée générée par la séquence $\sigma = \langle \ell_i, \lambda_i, \nu_i, \mu_i, \tau_i \rangle_{i \in I\!N}$ est la séquence $Trace_T(\sigma) = \langle \ell_i, \tau_i \rangle_{i \in I\!N}$.
Puis la trace associée à cette séquence σ est tout simplement: $Trace(\sigma) = \langle \ell_i \rangle_{i \in I\!N}$ □

Remarque 2.1.1 *Comme nous pouvons le remarquer, étant donnée une trace temporisée, le fait de connaître le séquencement des nœuds de contrôle visités et le temps de séjour pour chaque passage dans un nœud nous permet de reconstruire facilement une des séquences d'exécutions temporisées dont elle est issue.*
Les séquences d'exécutions temporisées dont est issue une trace temporisée peuvent être plusieurs car entre deux nœuds de contrôles nous pouvons avoir plusieurs arcs.

Puis nous appelons séquence d'exécution temporisée entière (respectivement trace temporisée entière) toute séquence $\langle \ell_i, \lambda_i, \nu_i, \mu_i, \tau_i \rangle_{i \in I\!N}$ (respectivement toute séquence $\langle \ell_i, \tau_i \rangle_{i \in I\!N}$) où tous les instants τ_i sont des nombres naturels. Nous appelons configuration entière toute configuration $\langle \ell, \lambda, \mu, \nu \rangle$ où ν est une valuation entière sur $\mathcal{X}$.

Notation 2.1.6 *Etant donnée une configuration entière α nous notons par $\mathcal{SET}_Z(\alpha, \mathcal{H})$ l'ensemble des séquences d'exécution temporisées entières de $\mathcal{H}$ commençant à partir de α.*

Puis nous définissons la notion d'état et de séquence d'états que nous utilisons tout au long de ce document.

Définition 2.13 Etat
Soit $\mathcal{P}$ un ensemble fini de propositions atomiques et $\Sigma = 2^{\mathcal{P}}$. Nous appelons *état* tout élément de Σ. □

Définition 2.14 Séquence d'états
Une *séquence d'états* est définie comme étant toute séquence infinie dans Σ^{ω}. □

Définition 2.15 Systèmes simples
Un système est dit simple si toutes ses horloges sont en phase, c'est-à-dire que toutes ses horloges ont la même partie fractionnelle. □

Définition 2.16 Systèmes larges
Un système est dit large si toutes les contraintes qui apparaissent dans le système sont des conjonctions d'inégalités larges. □

Tout au long de cette thèse, pour des raisons de convenance nous dirons tout simplement qu'un système est décidable ou indécidable, pour signifier que la vérification de propriétés d'atteignabilité propositionnelle (si un noeud de contrôle particulier est atteignable) sur ce système est décidable ou non.

2.2 Les sous-classes

Les systèmes hybrides telles que nous les avons définis dans la partie précédente sont assez généraux. Nous distinguons plusieurs sous-classes de tels systèmes, la particularité de ces sous-classes est que des résultats de décidabilités ont été établis pour leur vérification. La classe la plus utilisée est celle des systèmes temporisés pur [ACD93, HNSY94, Yov93] ayant uniquement des horloges comme variables. Plusieurs résultats très intéressants ont été établis pour ces systèmes, en effet la vérification de toute la logique $TCTL$ [ACD93] sur ces systèmes est décidable.

Pour notre part nous commençons par introduire la notion de pile à ces systèmes temporisés, nous notons ces systèmes temporisés à pile obtenus par **STP**.

Par la suite nous introduisons dans une première partie des compteurs de contrôle monotones pour obtenir des systèmes temporisés à pile munis de compteurs de contrôle

monotones que nous dénotons par **STPCM**. Ces compteurs sont dits monotones de par leur évolution, car ils sont croissants.

D'autre part nous introduisons des compteurs d'observations pour obtenir des systèmes temporisés à pile munis de compteurs d'observations que nous dénotons par **STPCO**.

Nous considérons aussi les systèmes temporisés à pile munis de ces deux types de compteurs (monotones de contrôle et d'observation) que nous dénotons par **STPCMO**.

Nous avons aussi munis les STP par des variables continues qui sont soit d'observation soit de contrôle. Dans le cas où nous munissons les STP par une variable d'observation, nous dénotons ces systèmes par **STPVO**, dans le second cas la variable de contrôle que nous considérons est un intégrateur et nous dénotons ces systèmes par **1-STPI**.

Puis il y a les deux dernières sous-classes de systèmes hybrides linéaires munis de structures de données discrètes qui sont les **STPCMOV** et les **STPCMOI**, ce sont des STPCMO munis respectivement d'une variable continue d'observation et d'un intégrateur. Nous pouvons extraire une dernière sous-classe assez particulière, c'est celle qu'on appelle la classe des graphes à intégrateurs étendus, qui est un système ayant pour uniques variables des intégrateurs étendus. Ces systèmes ne sont pas munis de pile, nous dénotons cette classe par **GIE**.

Dans les prochaînes sous-parties nous décrivons informellement ces différentes sous-classes et nous donnons pour un certain nombre d'entre elles un exemple d'illustration.

2.2.1 Les systèmes temporisés à pile

Les systèmes temporisés à pile (STP) est une sous-classe des SHP, les seules variables des STP autres que la pile sont des horloges. Dans cette sous-partie nous considérons des modèles permettant de représenter les comportements de systèmes hybrides à l'aide d'un système ayant un ensemble de nœuds de contrôles fini, une pile et un ensemble de variables continues qui sont des horloges. Dans la figure qui suit nous donnons un exemple de STP; nous avons repris l'exemple du nœud intermédiaire dans un réseau de transmission dans un cadre temporisé pur. Dans cet exemple le compteur c peut être représenté par une pile, les autres variables x, y et t sont des horloges.

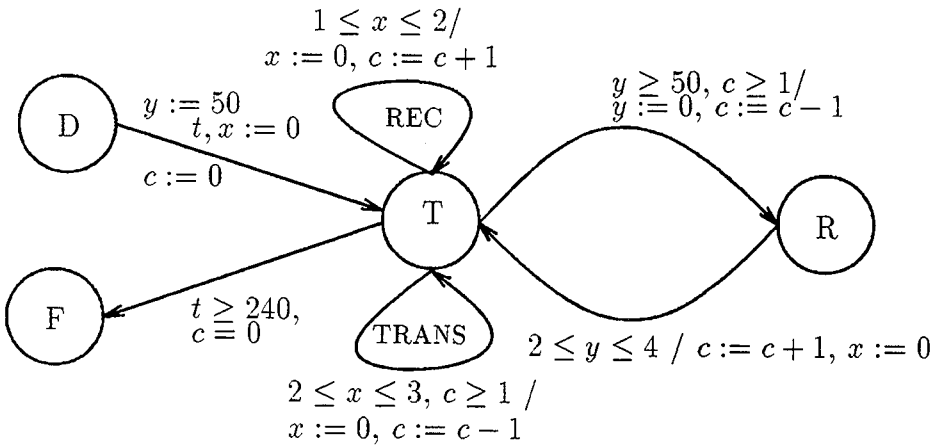


Figure 2.3: Exemple de système temporisé à pile

2.2.2 Systèmes temporisés à pile munis de compteurs de contrôles monotones

Dans cette partie nous portons notre intérêt sur les systèmes temporisés à pile enrichis à l'aide de variables de contrôles discrètes monotones que nous appelons compteurs de contrôle monotones, ces modèles sont notés STPCM. L'introduction de ces compteurs nous permet de décrire des systèmes ayant un comportement qui dépend des valeurs de certaines de ces variables discrètes. Ces variables discrètes sont les compteurs monotones que nous introduisons.

Reprenons l'exemple du nœud intermédiaire dans un réseau de transmission. Nous pouvons dans ce cas imposer dans le comportement de ce nœud, qu'il ne commence à effectuer les transmissions que si le nombre de paquets reçus dépasse une certaine valeur fixée. Le système passe donc par une phase d'initialisation durant laquelle il reçoit des paquets; une fois que le nombre de paquets reçus dépasse une certaine valeur la phase durant laquelle s'effectuent les transmissions et les réceptions sans restrictions est déclenchée. Nous retrouvons ce principe d'initialisation dans plusieurs applications.

Nous retrouvons par exemple ce principe dans une technique de visionnement en temps réel d'une bande vidéo à distance [RR93]. En effet, lorsque nous voulons visionner une bande vidéo se trouvant dans un site éloigné il est nécessaire de récupérer la bande et de la projeter sur notre écran. Or, vu le temps de transmission de la bande auquel s'ajoute le temps de projection, il y aura certainement des coupures dans la projection. Pour éviter ces coupures, il est nécessaire d'avoir une phase d'initialisation durant laquelle nous récupérons assez de données pour pouvoir passer à une phase de projection et de récupération de données simultanées sans qu'il n'y ait de coupure. La

quantité de données que nous récupérons dans la phase d'initialisation est généralement inférieure à la bande vidéo entière. L'idéal étant évidemment de minimiser la durée de cette phase d'initialisation.

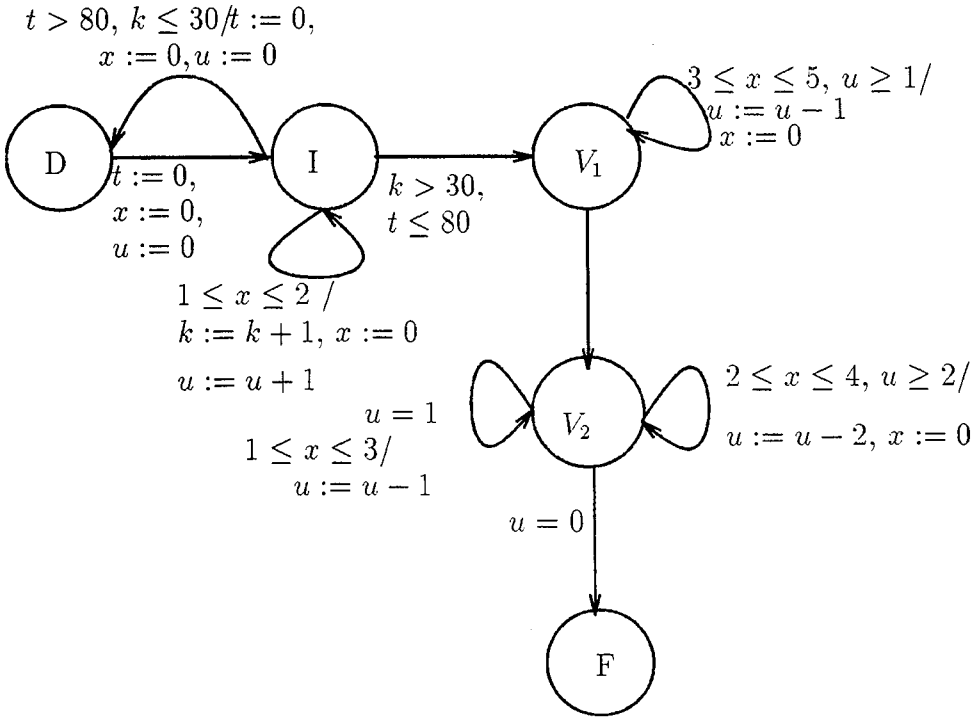


Figure 2.4: Visionnement d'une bande vidéo à distance

Pour illustrer les systèmes temporisés à pile munis de compteurs de contrôles monotones nous représentons par la figure 2.4 les comportements d'un processus de visionnement d'une bande vidéo se trouvant sur un site distant. La politique de visionnement est comme suit: le système pour commencer passe par une phase d'initialisation représentée dans la figure par le nœud de contrôle I, cette phase consiste à récupérer une partie de la bande comme réserve (afin d'éviter certains découpages pour manque de données). La réserve doit dans tous les cas être supérieure à 30 paquets. Aucun visionnement ne sera fait si la réserve est inférieure ou égale à 30 paquets. De plus si la réserve est encore inférieure ou égale à 30 paquets et que le temps écoulé depuis le début de la phase d'initialisation est supérieur à 80 on doit alors tout recommencer depuis le début.

Après avoir effectué cette réserve, le système passe dans une phase de croisière durant laquelle la récupération des paquets et leur consommation (visionnement) se font de manière simultanée. Cette phase est représentée par le nœud de contrôle V_1.

Il est à noter que durant cette phase, la consommation de deux paquets se fait pour chaque paquet récupéré.

Ensuite, le système passe par une phase de visionnement pur. Ce passage s'effectue lorsqu'il n'y a plus de paquets à récupérer. Le visionnement se fait comme dans la phase précédente, par paire de paquets. Mais durant cette phase le visionnement est un peu plus rapide que précédemment car il n'y a plus de récupération simultanée à assurer. Cette phase est représentée dans la figure par le nœud de contrôle V_2.

Puis si tous les paquets ont été consommés, le processus se termine en passant dans la phase représentée par le nœud de contrôle F. De plus, nous utilisons l'horloge x pour spécifier les contraintes relatives à chacune des tâches du système. Le compteur monotone k sert à spécifier les contraintes associées à la phase d'initialisation. Le compteur u sert à compter le nombre de paquets que le système a récupéré sans les avoir encore consommés.

2.2.3 Systèmes temporisés à pile munis de compteurs d'observation

Dans cette partie nous décrivons les systèmes temporisés à pile munis de compteurs d'observations. Ces systèmes sont différents des précédents, en effet les compteurs que nous introduisons sont différents. Ces compteurs sont différents dans le sens où d'une part dans la partie précédente les compteurs utilisés étaient monotones alors que dans la partie actuelle les compteurs ont une évolution quelconque, puis d'autre part, dans la partie précédente les compteurs que nous introduisons sont des compteurs de contrôle alors que les compteurs que nous introduisons dans la partie actuelle sont des compteurs d'observations. Nous rappelons que les compteurs de contrôle apparaissent dans les gardes ou dans les invariants associés à un noeud du système afin de restreindre certains comportements alors que les compteurs d'observations n'apparaissent pas dans les gardes mais servent plutôt à exprimer certaines propriétés d'atteignabilités que nous devons vérifier sur les systèmes considérés. Les compteurs que nous introduisons auront donc pour tâche d'observer et d'enregistrer certaines informations découlant des séquences d'exécution du système. Avoir par exemple un compteur qui enregistre la différence entre le nombre de réceptions et le nombre de transmissions d'une station dans un réseau. Cette variable sera donc initialisée à zéro lors de l'introduction de la station dans le réseau; elle sera incrémentée à chaque réception et décrémentée à chaque transmission. Cette variable pourra être utilisée pour exprimer un invariant (donc par complémentation une atteignabilité). Nous pouvons par exemple utiliser cette variable pour coder que le long de toute exécution du système, le nombre de réceptions est supérieur ou égal au nombre de transmissions. Ce qui peut être formulé par un invariant qui dit que le long de toute exécution du système la variable u est

toujours positive. Nous verrons dans le chapitre 3 que nous pouvons écrire cet invariant comme suit: $\forall\Box u \geq 0$ où u est la variable représentant le compteur d'observation.

2.2.4 Systèmes temporisés à pile munis de compteurs de contrôle monotones et des compteurs d'observation

Ces systèmes que nous notons par STPCMO, sont tout simplement des systèmes temporisés munis de compteurs de contrôle monotones et de compteurs d'observation, ils permettent d'avoir autant d'expressivité au niveau comportemental que les STPCM et autant d'expressivité au niveau des propriétés à vérifier sur ces modèles que les STPCO.

2.2.5 Systèmes temporisés à pile munis d'une variable d'observation

Dans cette partie nous considérons les systèmes temporisés à pile munis d'une variable continue d'observation; à l'aide d'une telle variable nous pouvons exprimer des invariants relatifs à la durée de certaines phases du système.

Pour illustrer cette partie nous reprenons l'exemple du nœud intermédiaire de transmission que nous avons introduit en début de ce chapitre dans la partie 2.2 en introduisant certaines modifications.

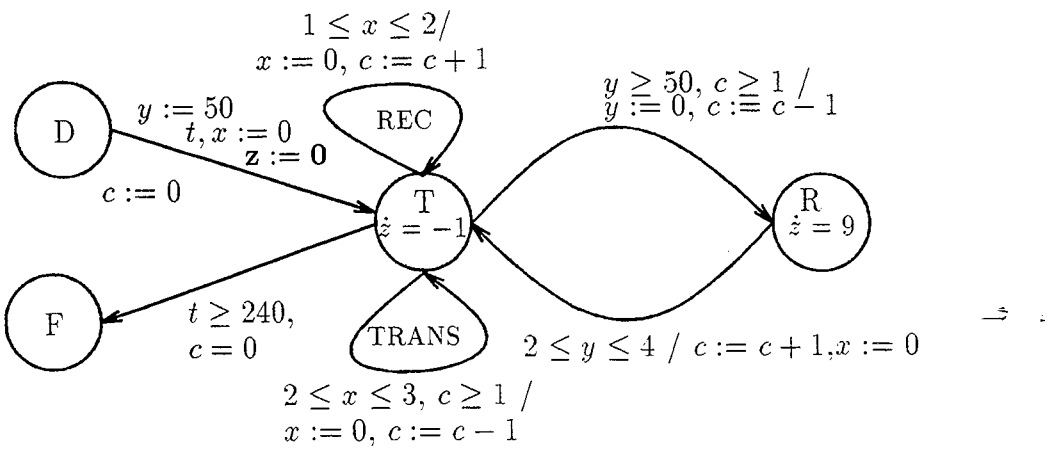


Figure 2.5: Exemple de systèmes temporisés à pile munis d'une variable d'observation continue

Dans le cadre de cette partie nous considérons l'exemple du nœud intermédiaire

dans un réseau de transmission que nous avons décrit dans l'exemple 2.2. Sauf que dans le cas actuel, nous gardons uniquement comme variables du système, les horloges, le compteur c que nous pouvons représenter par une pile et la variable continue d'observation z. L'introduction de la variable continue d'observation z va nous permettre d'exprimer la propriété que nous devrons vérifier sur le système. Le fait que *pour toute session le temps passé à recouvrir les paquets perdus est inférieur à 10 % du temps global de la session.* L'exemple est décrit par la figure 2.5. Nous rappelons que la variable d'observation est une variable continue z qui évolue selon la pente $10 \cdot \Delta_R - t$, où Δ_R est la durée de R, c'est à dire le temps accumulé durant lequel le contrôle du système est dans le nœud de contrôle R. Donc lorsque le contrôle est dans le nœud T (respectivement R), la variable z décroît (respectivement croît) avec une pente de -1 (resp. $+9$). La formule qui sera vérifiée est décrite informellement comme suit:
Pour toute exécution du système, si on est au nœud de fin de session F alors z doit être strictement négatif.

2.2.6 Systèmes temporisés à pile munis d'un intégrateur de contrôle

Les systèmes que nous décrivons dans cette partie, ont la particularité d'avoir leur contrôle qui porte non seulement sur les horloges mais aussi sur un intégrateur. Nous notons ces systèmes 1-STPI. Cet intégrateur de contrôle nous permet de spécifier certaines restrictions au niveau des comportements acceptés par les systèmes que nous étudions. Ce contrôle se fera en fonction de la durée de certaines phases. Donc dans cette partie nous munissons les systèmes temporisés à pile d'un intégrateur de contrôle. Nous rappelons qu'un intégrateur est une variable continue ayant ses pentes dans $\{0, 1\}$ et que cette variable peut donc être vue comme une horloge que nous pouvons arrêter durant les séjours dans certains nœuds. Par exemple, nous pouvons à l'aide de l'intégrateur, pour le cas du nœud intermédiaire de transmission, imposer le fait que nous ne permettrons la réception de nouveaux messages que si l'accumulation de la durée passée à effectuer le recouvrement de messages perdus est inférieure à une certaine constante. Nous pouvons voir ce que devient l'exemple en prenant cette constante égale à 20 dans la figure 2.6. Dans cet exemple, la variable y joue le rôle de l'intégrateur.

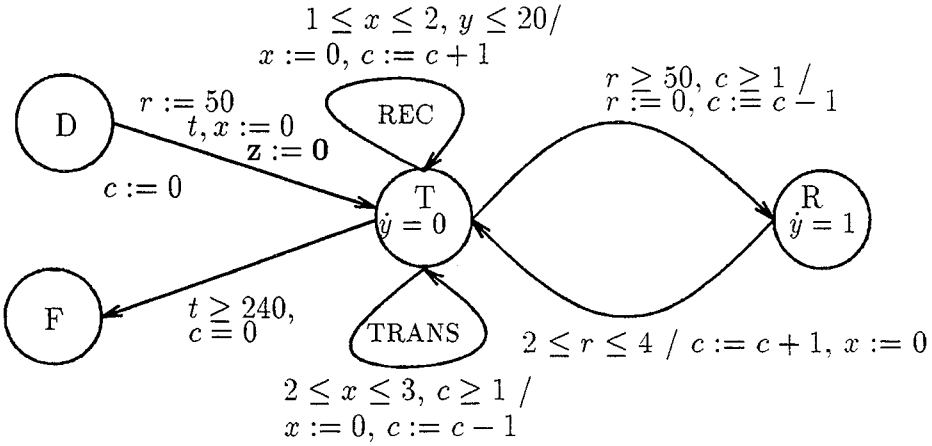


Figure 2.6: Exemple d'un STP + un intégrateur de contrôle

2.2.7 Systèmes temporisés munis d'intégrateurs étendus de contrôle

Nous décrivons les systèmes temporisés munis d'intégrateurs étendus de contrôle que nous notons par GIE (pour dire graphes à intégrateurs étendus) comme des systèmes qui ne sont pas munis de structures de données discrètes, ils possèdent comme variables uniquement des intégrateurs étendus. Le fait de considérer des pentes négatives permet par exemple de raisonner sur des différences entre des durées. Par exemple nous pouvons reprendre l'exemple du visionnement d'une bande vidéo à distance, en enlevant toutes les contraintes associées aux compteurs aussi bien celui qui est représenté par la pile que celui qui joue le rôle d'un compteur de contrôle monotone. Par contre nous imposons que d'une part la phase d'initialisation (représentée par le nœud I) prenne un temps compris entre 60 et 80 unités. De plus, pour toute exécution, la phase de visionnement pur (représentée par le nœud V_2) dure autant que la phase d'initialisation. Nous obtenons la figure 2.7.

2.3 Conclusion

Nous nous sommes intéressés dans ce chapitre à la présentation des systèmes hybrides linéaires munis de structures données discrètes. Nous avons décrit et illustré par des exemples les différentes sous-classes de ces systèmes que nous considérons dans la suite de cette thèse. En effet dans les prochains chapitres nous énonçons des résultats de décidabilités pour les sous-classes que nous avons décrites.

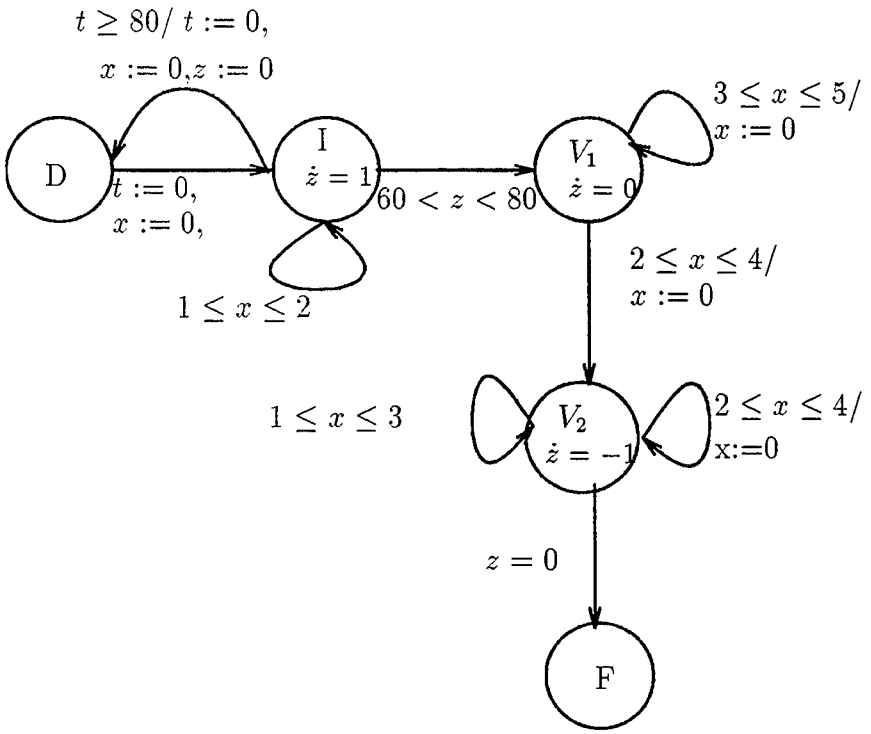


Figure 2.7: Exemple de systèmes a intégrateurs étendus

Dans le chapitre qui suit nous donnons les différentes propriétés que nous voulons vérifier sur ces sous-classes de systèmes hybrides. Dans une première partie nous présentons les différentes propriétés d'invariance auxquelles nous nous intéressons, qui sont des propriétés d'invariance contraintes. Les contraintes portent sur les différentes variables d'observation du système, qui sont aussi bien continues que discrètes.
Puis dans une seconde partie nous présentons des propriétés ω-régulières non temporisées qui sont décrites dans la logique ECTL* [EGK87, Tho89]. Ce sont les propriétés que nous voulons vérifier sur les graphes à intégrateurs étendus (GIE).

Chapitre 3

Les propriétés

Nous présentons dans ce chapitre les propriétés que nous voulons vérifier sur les systèmes que nous avons décrits dans le chapitre précédent.
Nous commençons par donner les formules d'invariance contraintes auxquelles nous nous intéressons, ce sont des propriétés d'invariance qui portent non seulement sur le temps mais aussi sur d'autres variables continues ou discrètes. Ces variables sont celles que nous avons définies comme étant des variables d'observation. Ce sont donc des variables qui n'agissent pas sur le contrôle du système mais qui enregistrent quantitativement les différentes évolutions lors d'une exécution du système considéré, c'est à l'aide de ces variables que nous pouvons exprimer certains invariants. Nous montrons par la suite comment réduire la vérification de ces propriétés à la vérification de propriétés d'atteignabilité propositionnelle. C'est-à-dire que le problème devient le suivant: "partant d'un nœud de contrôle particulier, peut-on atteindre ou non un certain nœud de contrôle spécifié ".

Nous décrivons ensuite les autres types de propriétés auxquelles nous nous intéressons, ces propriétés sont non-temporisées et sont décrites par la logique ECTL* [EGK87, Tho89]. A partir de cette logique, on extrait un fragment qui décrit toutes les propriétés ω-régulières non-temporisées (voir 3.2.1) que nous vérifierons sur les GIE.
Nous présentons dans ce cadre des résultats concernant les liens entre les équivalences comportementales basées sur la *bisimulation* (voir définition 3.1) avec les équivalences induites par la logique ECTL*.

3.1 Formules d'invariance et d'atteignabilité contraintes

Les propriétés d'invariance contraintes correspondent à des conditions de sureté sur le comportement (exécution) d'un système donné. Ces propriétés sont les duales des propriétés d'atteignabilité. En effet, vouloir vérifier un invariant sur un système, c'est à dire une propriété qui soit toujours vraie tout au long de toute exécution est équivalent à vérifier que jamais lors d'une exécution il est possible d'avoir la négation de la propriété qui nous intéresse. L'invariant est faux si on atteint sa négation sinon il est vrai. Dans cette section on définit des formules exprimant des propriétés d'invariance et d'atteignabilité des configurations des automates hybrides linéaires munis de structures de données discrètes.

Soient $\mathcal{P}$ l'ensemble des propositions, $\Sigma = 2^{\mathcal{P}}$ et $\mathcal{H} = \langle \mathcal{L}, \Pi, \mathcal{X}, \partial, \mathcal{Y}, \Gamma, I, \delta \rangle$ un système hybride sur $\Sigma = 2^{\mathcal{P}}$. On utilise les lettres $P, Q, \cdots$ pour les propositions atomiques (éléments de $\mathcal{P}$) et les lettres $f, g \cdots$ pour les contraintes linéaires sur $\mathcal{X} \cup \mathcal{Y}$ donc des éléments de $\mathcal{C}_{\mathcal{X} \cup \mathcal{Y}}$. Considérons l'ensemble de formules défini par la grammaire suivante:

$$\varphi ::= P \mid f \mid \neg\varphi \mid \varphi \vee \varphi \mid \forall\Box\varphi$$

La sémantique de ces formules est définie par la relation de satisfaction $\models$ sur les configurations du système $\mathcal{H}$. Pour toute configuration $\alpha = \langle \ell, \lambda, \nu, \mu \rangle$, la relation $\models$ est définie de manière inductive comme suit:

$$\begin{array}{lll}
\alpha \models P & \text{ssi} & P \in \Pi(\ell) \\
\alpha \models f & \text{ssi} & \langle \nu, \mu \rangle \models f \\
\alpha \models \neg\varphi & \text{ssi} & \alpha \not\models \varphi \\
\alpha \models \varphi_1 \vee \varphi_2 & \text{ssi} & \alpha \models \varphi_1 \text{ ou } \alpha \models \varphi_2 \\
\alpha \models \forall\Box\varphi & \text{ssi} & \forall \sigma = \{\langle \ell_i, \lambda_i, \nu_i, \mu_i, t_i \rangle\}_{i \in I\!N} \in \mathcal{SET}(\alpha, \mathcal{H}).\ \forall i \in I\!N.\ \langle \ell_i, \lambda_i, \nu_i, \mu_i \rangle \models \varphi
\end{array}$$

Nous notons par $\exists\Diamond$ pour signifier $\neg\forall\Box\neg$.

Exemple 3.1 Si nous reprenons l'exemple du nœud intermédiaire dans un réseau de transmission décrit dans le chapitre précédent. Nous rappelons que les propriétés que nous désirons vérifier sont les suivantes:

1. *Pour toute session, le temps passé à récupérer les paquets perdus est inférieur à 10 % du temps global de la session,*
2. *Pour toute session, il y a exactement le même nombre de réceptions que de transmissions,*

3. *durant toute session, le nombre de transmissions ne dépasse jamais le nombre de réceptions.*

Les propriétés (1), (2) et (3) sont exprimées à l'aide des variables u et z par les formules d'invariance suivantes:

1. $\forall\Box(at-F \Rightarrow z < 0)$
2. $\forall\Box(at-F \Rightarrow u = 0)$
3. $\forall\Box u \geq 0$

où "at-F" est une proposition atomique qui est vraie uniquement dans le nœud F. □

3.1.1 De l'invariant avec contraintes vers l'atteignabilité propositionnelle

Nous montrons dans cette partie comment réduire tous les problèmes de vérification d'invariants avec des contraintes sur des systèmes hybrides linéaires munis de structures de données discrètes à des problèmes de vérification de formules d'atteignabilité purement propositionnelle, c'est-à-dire à l'atteignabilité d'un nœud particulier.

En utilisant les règles standards pour les opérations booléennes avec le fait que toute formule $\exists\Diamond(\psi_1 \vee \psi_2)$ est équivalente à $\exists\Diamond\psi_1 \vee \exists\Diamond\psi_2$, nous pouvons alors facilement montrer que toute formule d'invariance avec contraintes est équivalente à la négation d'une formule de la forme:

$$\bigvee_{i=1}^{n} \exists\Diamond(\pi_i \wedge \bigwedge_{j=1}^{m_i} f_i^j)$$

Avec π_i une formule propositionnelle pour tout i tel que $1 \leq i \leq n$ La réduction se fait alors da la manière suivante:
Nous commençons par ajouter un nœud de contrôle final ℓ_{final} de sorte qu'à partir de tout nœud ℓ_k ayant π_i vraie (c'est-à-dire que $\pi_i \in \Pi(\ell_k)$) nous ajoutons un arc de ℓ_k vers ℓ_{final} ayant pour garde $\bigwedge_{j=1}^{m_i} f_i^j$. Nous étiquetons ce nouveau nœud par une proposition qui n'est vraie que dans ce nœud ($at_{\ell_{final}}$), c'est alors que nous aurons simplement à vérifier la formule d'atteignabilité propositionnelle pure $\exists\Diamond at_{\ell_{final}}$ sur le nouveau système obtenu par le rajout du nouveau nœud et des nouveaux arcs. Le nouveau système $\mathcal{H}'$ est obtenu à partir de $\mathcal{H}$ de la manière suivante:

$\mathcal{H}' = \langle \mathcal{L}', \Pi', \mathcal{X}', \partial', \mathcal{Y}', \Gamma', I', \delta' \rangle$ avec :

- $\mathcal{L}' = \mathcal{L} \cup \{\ell_{final}\}$.
- $\forall \ell \in \mathcal{L}$, $\Pi'(\ell) = \Pi(\ell)$ et $\Pi'(\ell_{final}) = at_{\ell_{final}}$.
- $\mathcal{X}' = \mathcal{X}$.
- $\forall x \in \mathcal{X}, \forall \ell \in \mathcal{L}, \partial'(\ell, x) = \partial(\ell, x)$
  $\forall x \in \mathcal{X}, \partial'(\ell_{final}, x) \in \mathbb{Z}$ c'est à dire que la pente des variables continues dans ce nœud final importe peu.
- $\mathcal{Y}' = \mathcal{Y}$.
- $\Gamma' = \Gamma$
- $\forall \ell \in \mathcal{L}$, $I'(\ell) = I(\ell)$ et $I'(\ell_{final}) = vraie$.
- $\delta' = \delta \cup \{\langle \ell_k, g_i, Z, Z, Id, \emptyset, \ell_{final}\rangle : g_i = \bigwedge_{j=1}^{m_i} f_i^j,\ \pi_i \in \Pi(\ell_k), Z \in \Gamma^*\}$

Il faut noter que $\mathcal{X}$ et $\mathcal{Y}$ restent inchangés car toutes les variables qui apparaissent dans les contraintes f_i^j doivent être prédéfinies dans le système comme des variables d'observations.

Remarque 3.1 Comme nous pouvons le remarquer toutes les variables qui apparaissent dans les contraintes de l'invariant apparaissent aussi dans le modèle, elles doivent en effet être prédéfinies dans le modèle. Donc la vérification d'une formule d'invariance contrainte a pour effet d'augmenter la dimension du modèle (nombre de variables). □

Comme nous pouvons le remarquer cette réduction de l'atteignabilité contrainte vers une atteignabilité propositionnelle a pour effet de transformer les les variables d'obervation en variables de contrôle. En effet à la suite de la réduction ces variables apparaitront dans certaines gardes du système considéré.

3.2 Propriétés ω-régulières

Dans cette partie nous introduisons d'autres types de propriétés, qui ne sont pas temporisées et qui sont représentées par la logique de branchement ECTL* [EGK87, Tho89]. Nous prenons des propriétés non-temporisées afin de ne pas augmenter la dimension des systèmes sur lesquels nous voulons vérifier ces propriétés (voir la remarque 3.1). Nous commençons par donner au préalable la définition des automates de Muller. Ces automates nous servent à exprimer, d'une part des propriétés ω-régulières que nous

voulons vérifier sur les GIE et d'autre part les conditions de divergence des séquences d'exécutions temporisées. Nous verrons que ces propriétés ω-régulières sont aussi exprimables dans un fragment de la logique ECTL*.
Nous présentons aussi des résultats d'adéquation concernant la logique ECTL*.

3.2.1 ω-automates de Muller

Nous donnons dans cette partie la définition des ω-automates de Muller [Mul63] et quelques résultats de base les concernant et qui sont très importants pour le chapitre 6. En effet, c'est à l'aide des conditions d'acceptance de Muller que nous pouvons exprimer le fait que nous effectuons la vérification uniquement sur les séquences d'executions (donc par définition divergentes). Nous verrons dans la partie 6.5 du chapitre 6 que pour exprimer les conditions de divergence des séquences, le choix des automates de Muller est tout à fait convenable.

Soit V un alphabet fini. Un ω-automate de Muller sur V est un tuple $\mathcal{A} = (\mathcal{Q}, q_I, \delta, \mathcal{F})$ où $\mathcal{Q}$ est un ensemble fini de nœuds de contrôle, $q_I \in \mathcal{Q}$ est le nœud de contrôle initial, $\delta \subseteq \mathcal{Q} \times V \times \mathcal{Q}$ est une relation de transition étiquetée et $\mathcal{F} \subseteq 2^{\mathcal{Q}}$ est une collection d'*ensembles de répétition.* Etant donnée une séquence infinie $\sigma = (s_i)_{i \in \mathbb{N}} \in V^\omega$, une exécution de $\mathcal{A}$ sur σ est une séquence infinie $\rho = (q_i)_{i \in \mathbb{N}}$ telle que $q_0 = q_I$, et $\forall i \in \mathbb{N}.\ (q_i, s_i, q_{i+1}) \in \delta$. Plus encore, l'exécution ρ sur σ est *acceptante* si $\{q \in \mathcal{Q} : \overset{\infty}{\exists}\, i \in \mathbb{N}.\ q_i = q\} \in \mathcal{F}$, $\overset{\infty}{\exists}\, i \in \mathbb{N}$ signifie: " il existe une infinité d'indices i". Donc l'exécution ρ sur σ est *acceptante* si l'ensemble des nœuds répétés infiniment sur la séquence ρ est un élément de $\mathcal{F}$.
On a alors la séquence $\sigma \in V^\omega$ qui est acceptée par l'automate $\mathcal{A}$ s'il a une exécution acceptante dans $\mathcal{A}$.
Nous notons $L(\mathcal{A})$ l'ensemble des séquences acceptées par $\mathcal{A}$. Les ensembles de langages définissables par des ω-automates de Muller sont appelés *langages ω-réguliers.* Il est bien connu que cette classe de *langages ω-réguliers* est fermée par les opérations booléennes et que le problème du langage vide des *langages ω-réguliers* est décidable [Tho90].

La définition que nous venons de présenter ci-dessus peut être étendue simplement (comme pour les automates de mots finis) aux ω-automates de Muller à un compteur et plus généralement aux ω-automates de Muller à pile.
Les ω-automates de Muller à pile définissent les *ω-langages hors-contextes.*
Il a été vu [CG77] que le problème d'inclusion, savoir si un langage ω-hors-contexte est inclu dans un langage ω-régulier est décidable. Ce résultat est un aboutissement dû fait que l'intersection d'un langage ω-hors-contexte avec un langage ω-régulier est un langage ω-hors-contexte et du fait que les ω-réguliers sont fermés par complémentation; puis finalement grâce au fait que le problème du langage vide des langages ω-hors-contextes est décidable [CG77]. En effet supposons que nous voulons savoir si

$L_1 \subseteq L_2$ ou non, avec L_1 un langage ω-hors-contexte et L_2 un langage ω-régulier. Il faut alors pouvoir décider si $L_1 \cap \bar{L}_2 = \emptyset$ ou non. L_2 étant un langage ω-régulier donc fermé par complémentation, nous obtenons alors que $\bar{L}_2$ est un langage ω-régulier. De plus nous savons que l'intersection d'un langage ω-hors-contexte avec un langage ω-régulier est un langage ω-hors-contexte, donc $L_1 \cap \bar{L}_2$ est un langage ω-hors-contexte. Puisque le problème du langage vide des langages ω-hors-contextes est décidable [CG77], nous avons alors que le problème de l'inclusion d'un langage ω-hors-contexte dans un langage ω-régulier est décidable.

3.2.2 La logique ECTL*

Cette logique est définie comme l'union de logiques stratifiées ECTL^*_n pour tout $n \in \mathbb{N}$. L'ensemble des formules de ECTL^*_0 est simplement l'ensemble des propositions atomiques $\mathcal{P}$, et pour tout $n \geq 1$, l'ensemble des formules de ECTL^*_n est le plus petit ensemble de formules telles que:

- $\mathrm{ECTL}^*_{n-1} \subseteq \mathrm{ECTL}^*_n$,
- si $\varphi \in \mathrm{ECTL}^*_n$, alors $\neg\varphi \in \mathrm{ECTL}^*_n$,
- si $\varphi_1, \varphi_2 \in \mathrm{ECTL}^*_n$, alors $\varphi_1 \vee \varphi_2 \in \mathrm{ECTL}^*_n$,
- si Φ est un ensemble fini de formules de ECTL^*_{n-1}, et $\Omega(\Phi)$ est un langage ω-régulier sur l'alphabet 2^Φ, alors $\exists\Omega(\Phi) \in \mathrm{ECTL}^*_n$.

Nous considérons comme abréviations les opérateurs booléens comme la conjonction ($\wedge$) et l'implication ($\Rightarrow$), aussi bien que $\forall\Omega(\Phi) = \neg\exists\overline{\Omega}(\Phi)$ où $\overline{\Omega}(\Phi) = 2^\Phi - \Omega(\Phi)$.

Soit $\mathcal{H}$ un GIE (graphe à intégrateurs étendus). Nous notons par $Conf(\mathcal{H})$ l'ensemble de toutes les configurations de $\mathcal{H}$. Alors, les formules de ECTL* sont interprétées comme des ensembles de configurations de $\mathcal{H}$. Intuitivement, la formule $\exists\Omega(\Phi)$ représente l'ensemble des configurations à partir desquelles il existe une séquence d'exécution $(\alpha_i)_{i\in\mathbb{N}}$ avec $\alpha_i = \langle \ell_i, \nu_i \rangle$ satisfaisant la *contrainte de chemin* $\Omega(\Phi)$, c'est à dire la séquence dans $(2^\Phi)^\omega$ correspondant aux sous ensembles de Φ qui sont satisfaits successivement par les α_i le long du langage $\Omega(\Phi)$. L'interprétation des formules de ECTL* est définie de manière inductive de la manière suivante:

- $[\![P]\!] = \{\langle \ell, \nu \rangle \in Conf(\mathcal{H}) \ : \ P \in \Pi(\ell)\}$,
- $[\![\neg\varphi]\!] = \{\alpha \in Conf(\mathcal{H}) \ : \ \alpha \notin [\![\varphi]\!]\}$,
- $[\![\varphi_1 \vee \varphi_2]\!] = [\![\varphi_1]\!] \cup [\![\varphi_2]\!]$,

- $[\![\exists\Omega(\Phi)]\!] = \{\alpha \in \mathcal{C}onf(\mathcal{H}) \ : \ \exists(\alpha_i)_{i\in\mathbb{N}} \in \mathcal{SE}(\alpha,\mathcal{H}).\ (T_\Phi(\alpha_i))_{i\in\mathbb{N}} \in \Omega(\Phi)\}$, où pour tout $i \in \mathbb{N}$, $T_\Phi(\alpha_i) = \{\varphi \in \Phi \ : \ \alpha_i \in [\![\varphi]\!]\}$.

Il est facile de voir que les logiques de branchement (branching-time temporal logics) CTL [CES83] et CTL* [EH83] sont moins expressives que ECTL*. Pour tout $n \in \mathbb{N}$, nous pouvons extraire CTL_n et CTL^*_n les fragments de ECTL^*_n correspondant respectivement aux formules de CTL et de CTL* ayant au plus n imbrications de quantificateurs de chemins ($\exists$).

Nous pouvons aussi observer que ECTL* peut exprimer toutes les propriétés ω-régulières linéaires, c'est-à-dire toutes les propriétés exprimables par un automate de Muller. En effet, ces propriétés correspondent aux formules de ECTL^*_1 de la forme $\forall\Omega(\mathcal{P})$. Par conséquent, ECTL^*_1 subsume les logiques temporelles linéaires telles que PTL [Pnu77] et ETL [Wol83], aussi bien que le μ-calcul linéaire [Var88].

3.3 Equivalences comportementales et équivalences logiques

Nous présentons dans cette section les résultats concernant les liens entre les équivalences comportementales basées sur la bisimulation avec les équivalences induites par la logique ECTL* et ses fragments ECTL^*_n, pour tout $n \in \mathbb{N}$.

Pour commencer nous introduisons la notion de *graphe de transition* associé à un GIE.
Soit $\mathcal{H}$ un GIE, nous considérons la relation de transition entre les configurations de $\mathcal{H}$ définies par:
$\langle \ell, \nu\rangle \ \triangleright \ \langle \ell', \nu'\rangle$ si et seulement si
soit que $\ell = \ell'$ et $\exists t \in \mathbb{R}_{\geq 0}.\ \nu' = [\nu + t]_\ell$, avec $\forall t' \in [0,t]\ [\nu + t']_\ell$ satisfait l'invariant $I(\ell)$.
ou bien que $\exists(\ell, g, X, \ell') \in \mathcal{E}.\ \nu \models g$ avec $\nu' = \nu[X \mapsto 0]$ et ν' satisfait l'invariant $I(\ell')$.
Alors, le graphe de transition associé à $\mathcal{H}$ est $\mathcal{G}_\mathcal{H} = (\mathcal{C}onf(\mathcal{H}),\ \triangleright)$.

Intuitivement, la relation $\triangleright$ représente soit les transitions associées à la progression du temps dans un même nœud de contrôle, soit des transitions discrètes entre différents nœuds de contrôles. Il est à noter que pour chaque séquence d'exécution $(\langle \ell_i, \nu_i\rangle)_{i\in\mathbb{N}}$, nous avons $\forall i \in \mathbb{N}.\ \langle \ell_i, \nu_i\rangle \ \triangleright \ \langle \ell_{i+1}, \nu_{i+1}\rangle$.

Dans ce qui suit nous définissons une relation d'équivalence particulière qui est la bisimulation. Nous pouvons ainsi définir des équivalences comportementales basées sur cette bisimulation. Nous introduisons donc cette notion de bisimulation sur le graphe $\mathcal{G}_\mathcal{H}$ dans la définition qui suit:

Définition 3.1 Une *bisimulation* sur $\mathcal{G}_{\mathcal{H}}$ est toute relation binaire symétrique R entre les configurations de $\mathcal{H}$ telle que $\langle \ell_1, \nu_1 \rangle R \langle \ell_2, \nu_2 \rangle$ si et seulement si

- $\Pi(\ell_1) = \Pi(\ell_2)$, et
- $\forall \alpha_1 \ \langle \ell_1, \nu_1 \rangle \rhd \alpha_1$ implique qu'il $\exists \alpha_2 . \ \langle \ell_2, \nu_2 \rangle \rhd \alpha_2$ et $\alpha_1 R \alpha_2$.

□

Nous notons par $\sim$ la plus grande bisimulation sur $\mathcal{G}_{\mathcal{H}}$. Nous pouvons facilement vérifier que $\sim$ est une relation d'équivalence.

La relation $\sim$ peut être étendue aux séquences d'exécutions. Deux séquences d'exécutions sont $\sim$-équivalentes si pour tout indice $i \in I\!N$, les configurations de l'indice i sont $\sim$-équivalentes.

Nous introduisons alors le lemme suivant qui concerne la bisimilarité entre les configurations et la bisimilarité entre séquences d'exécutions.

Lemme 3.1 $\forall \alpha, \alpha' \in Conf(\mathcal{H})$, $\alpha \sim \alpha'$ implique que $\forall (\alpha_i)_{i \in I\!N} \in \mathcal{SE}(\alpha, \mathcal{H})$, $\exists (\alpha'_i)_{i \in I\!N} \in \mathcal{SE}(\alpha', \mathcal{H})$ telle que $\forall i \in I\!N . \ \alpha_i \sim \alpha'_i$. ■

Preuve 3.1 La preuve de ce lemme s'effectue de manière triviale par induction et en utilisant la définition de la bisimulation.

- i=0 Nous avons $\alpha = \alpha_0$, $\alpha' = \alpha'_0$ et $\alpha \sim \alpha'$ donc $\alpha_0 \sim \alpha'_0$.
- Supposons que le lemme est vrai pour tout $i \leq n$ et montrons qu'il est alors vrai pour $i = n + 1$.
  Donc
  nous avons par hypothèse que $\alpha \sim \alpha'$ implique que $\forall (\alpha_i)_{i \leq n} \in Pref(\mathcal{SE}(\alpha, \mathcal{H}))$, $\exists (\alpha'_i)_{i \leq n} \in Pref(\mathcal{SE}(\alpha', \mathcal{H}))$ telle que $\forall i \leq n . \ \alpha_i \sim \alpha'_i$ avec $Pref$ d'un ensemble de séquences dénote l'ensemble des préfixes de cet ensemble de séquences.
  Donc pour tout α_{n+1} telle que $\alpha_n \rhd \alpha_{n+1}$ (d'après la définition de la bisimulation) $\exists \alpha'_{n+1}$ telle que $\alpha'_n \rhd \alpha'_{n+1}$ et $\alpha_{n+1} \sim \alpha'_{n+1}$.

■

En utilisant le lemme 3.1, le résultat suivant, de préservation des formules de ECTL* peut être prouvé par induction sur la structure des formules de ECTL*.

Proposition 3.1 Pour toutes configurations α et α', et toute formule de φ ECTL*, $\alpha \sim \alpha'$ implique que $\alpha \in [\![\varphi]\!]$ si et seulement si $\alpha' \in [\![\varphi]\!]$. ■

Preuve 3.2 Pour la preuve de cette proposition nous procédons par induction sur la structure des formules de ECTL*.

- Si $\varphi \in ECTL_0^*$ alors $\varphi \in \mathcal{P}$. Nous avons $\alpha = \langle \ell, \nu \rangle \in [\![\varphi]\!]$ donc $\varphi \in \Pi(\ell)$. De plus puisque $\alpha' = \langle \ell', \nu' \rangle \sim \alpha$ alors (d'après la définition de $\sim$) $\Pi(\ell) = \Pi(\ell')$ donc $\varphi \in \Pi(\ell')$. Nous obtenons alors que $\alpha' \in [\![\varphi]\!]$.

- Supposons que la proposition est vraie pour toute formule de ECTL$_n^*$ avec $n \in I\!N$ montrons que la proposition est alors vraie pour $n+1$.
  Soient $\varphi \in ECTL_{n+1}^*$, $\alpha \in Conf(\mathcal{H})$ et $\alpha' \in Conf(\mathcal{H})$ telles que $\alpha \sim \alpha'$ alors:

  1. Soit que $\varphi \in ECTL_n^*$ auquel cas il suffit d'utiliser l'hypothèse de récurrence.
  2. Soit que $\varphi = \exists\Omega(\Phi)$ avec $\Omega(\Phi)$ un langage ω-régulier sur l'alphabet 2^Φ.
     $\alpha \in [\![\varphi]\!]$ donc $\exists(\alpha_i)_{i \in I\!N} \in \mathcal{SE}(\alpha, \mathcal{H})$ telle que $(T_\Phi(\alpha_i))_{i \in I\!N} \in \Omega(\Phi)\}$, où pour tout $i \in I\!N$, $T_\Phi(\alpha_i) = \{\psi \in \Phi \ : \ \alpha_i \in [\![\psi]\!]\}$. D'après le lemme 3.1 $\exists(\alpha'_i)_{i \in I\!N} \in \mathcal{SE}(\alpha', \mathcal{H})$ et telle que $\forall i \in I\!N \ \alpha_i \sim \alpha'_i$ donc par hypothèse de récurrence (car toutes les formules de Φ sont dans ECTL$_n^*$) $\forall i \in I\!N \ T_\Phi(\alpha_i) = T_\Phi(\alpha'_i)$ donc $(T_\Phi(\alpha'_i)_{i \in I\!N} \in \Omega(\Phi))$ d'où nous obtenons $\alpha' \in [\![\varphi]\!]$.
  3. Soit que φ est une combinaison booléenne de formules de ECTL$_{n+1}^*$ de la forme du cas précédent, ce cas se ramene de manière triviale au cas précédent.

■

Nous introduisons maintenant la famille décroissante $(\sim_n)_{n \in I\!N}$ d'équivalences entre configurations, et nous montrons que pour tout $n \in I\!N$, $\sim_n$ est compatible avec (inclus dans) l'équivalence induite par ECTL$_n^*$. La famille $(\sim_n)_{n \in I\!N}$ est définie comme suit:

- $\langle \ell, \nu \rangle \sim_0 \langle \ell', \nu' \rangle$ si et seulement si $\Pi(\ell) = \Pi(\ell')$,
- $\forall n \in I\!N$, $\alpha \sim_{n+1} \alpha'$ si et seulement si
  - $\forall (\alpha_i)_{i \in I\!N} \in \mathcal{SE}(\alpha, \mathcal{H}). \exists (\alpha'_i)_{i \in I\!N} \in \mathcal{SE}(\alpha', \mathcal{H}). \forall i \in I\!N. \ \alpha_i \sim_n \alpha'_i$,
  - $\forall (\alpha'_i)_{i \in I\!N} \in \mathcal{SE}(\alpha', \mathcal{H}). \exists (\alpha_i)_{i \in I\!N} \in \mathcal{SE}(\alpha, \mathcal{H}). \forall i \in I\!N. \ \alpha_i \sim_n \alpha'_i$.

Nous pouvons vérifier que pour tout $n \in \mathbb{N}$, $\sim_n$ est en effet une équivalence. Il est à noter que pour toutes configurations α et α', $\alpha \sim_1 \alpha'$ si et seulement si $\mathcal{T}(\alpha, \mathcal{H}) = \mathcal{T}(\alpha', \mathcal{H})$, où $\mathcal{T}(\alpha, \mathcal{H})$ dénote l'ensemble des séquences d'états générées par les séquences d'exécutions dans $\mathcal{SE}(\alpha, \mathcal{H})$ (Nous appelons aussi $\mathcal{T}$ l'ensemble des traces interprétées). Nous obtenons donc que $\sim_1$ correspond à l'équivalence de trace.

Nous pouvons prouver similairement à la proposition 3.1, en utilisant l'induction sur les nombres naturels, que pour tout $n \in \mathbb{N}$, l'équivalence $\sim_n$ préserve les formules de ECTL^*_n.

Proposition 3.2 Pour tout $n \in \mathbb{N}$, toutes configurations α, α' et toute formule φ de ECTL^*_n, $\alpha \sim_n \alpha'$ implique que $\alpha \in [\![\varphi]\!]$ si et seulement si $\alpha' \in [\![\varphi]\!]$. ■

Preuve 3.3 Pour la preuve de cette proposition nous procédons par induction sur $n \in \mathbb{N}$. Cette preuve est similaire à celle de la proposition précédente. ■

3.4 Conclusion

Nous avons introduit dans ce chapitre, les différents types de propriétés que nous désirons vérifier sur les modèles que nous considérons.
Nous considérons d'une part des propriétés d'invariance contraintes; les contraintes qui apparaissent dans ces propriétés ne concernent pas uniquement des horloges, mais peuvent aussi porter sur des variables discrètes ce qui permet d'exprimer des propriétés relatifs aux occurrences de certains événements.
Les propriétés que nous exprimons peuvent être ni régulières et ni même hors-contextes [BER93, BER94c] on peut en effet décrire des propriétés sous-contextes. Telle que par exemple des propriétés décrivant que le nombre d'occurrences de l'événement e_1 est égal au nombre d'occurrences de l'événement e_2 et est égal au nombre d'occurrences de l'événement e_3 (le langage $e_1^n e_2^n e_3^n$ est sous-contexte). Ce qui nous permet d'obtenir une expressivité plus importante que les propriétés portant uniquement sur les variables continues.

Nous considérons par la suite des propriétés ω-régulières décrites dans la logique ECTL*. Nous allons vérifier ces propriétés sur des systèmes qui ont la particularité d'être sensibles au nombre de variables qu'ils comportent. Nous considérons donc uniquement des propriétés non temporisées, afin de ne pas augmenter la dimension des systèmes que nous considérons dans ce cas. Des systèmes tels que les GIE sont sensibles à l'augmentation de ses variables (sa dimension), dans le sens où pour une

certaine dimension le problème de la vérification est décidable alors que ce même problème devient indécidable si nous ajoutons une variable.
Nous donnons aussi pour cette logique ECTL* des résultats d'adéquation qui sont utilisés dans le chapitre 6 dans la vérification des propriétés exprimées dans des fragments de cette logique sur des GIE.

Chapitre 4

Partitionnement et discrétisation

Comme nous l'avons vu dans les définitions dans le chapitre 2, les modèles que nous utilisons pour représenter les systèmes hybrides sont des modèles denses ce qui est dû à la continuité de certaines variables, dont le temps. Une conséquence directe est que chaque nœud de contrôle définit un nombre infini de configurations sous l'unique condition que le temps puisse évoluer dans ce nœud. En effet deux cas se présentent lors de la visite d'un nœud de contrôle ℓ:

- Il est interdit de séjourner dans ce noeud (il faut le quitter au même moment où il est visité), auquel cas ce nœud définit une unique configuration temporisée.
- Il est permis de séjourner dans ce nœud un temps non nul. Donc du fait que le temps est dense, entre les valuations des variables continues à l'entrée dans le nœud et les valuations de ces mêmes variables à la sortie de ce nœud il existe une infinité de valuations de ces mêmes variables. Ce même nœud, peut donc définir une infinité de configurations temporisées, une pour chaque valuation.

Une phase importante pour la vérification de systèmes hybrides est donc la réduction du problème de vérification d'un domaine dense vers un domaine discret. Une seconde phase tout aussi importante que la première est de résoudre ce même problème dans le cadre discret.
Dans la partie qui suit nous présentons deux techniques de réductions d'un cadre dense à un cadre discret, nous commençons par la technique de partitionnement introduite dans [ACD93] pour introduire par la suite la technique de discrétisation qui est la numérisation uniforme [HMP92].

Pour la vérification des modèles que nous considérons dans cette thèse nous utiliserons soit les techniques de partitionnement introduite dans [ACD93], soit de numérisation uniforme introduite dans [HMP92], par contre nous montrons que ces techniques ne sont pas applicables pour d'autres modèles plus complexes que nous considérons.

Pour ces modèles nous utilisons soit notre technique de numérisation non uniforme que nous présentons dans le prochain chapitre, soit notre technique de partitionnement que nous présentons dans le chapitre 6.

4.1 Réduction du dense vers le discret

Une première phase pour effectuer la vérification est comme nous l'avons dit, de réduire le problème d'un domaine dense en un domaine discret; pour cela deux techniques existent.
La première introduite par [ACD93] est le partitionnement de l'espace infini des configurations de sorte à obtenir un ensemble discret de classes. Telles que toutes les configurations d'une même classe satisfont les mêmes propriétés auxquelles on s'intéresse. Dans [ACD93, HNSY94] les auteurs montrent que si les modèles considérés sont des automates temporisés et que les propriétés qui doivent être vérifiées sur ces modèles sont des formules de la logique temporisée arborescente TCTL (Timed Computational Tree Logic) alors un tel partitionnement est possible. En effet les auteurs montrent qu'il existe un partitionnement pour un graphe temporisé donné de sorte que deux configurations quelconques d'une même classe ne distinguent pas une même formule de TCTL.

La seconde technique que l'on retrouve entre autres dans [HMP92, Bur92, BER94b, AM95, BER94d] consiste à ne considérer qu'un sous-ensemble de l'ensemble infini des configurations de sorte que ce sous-ensemble soit discret. Cette technique est habituellement appelée discrétisation. Mais pour pouvoir se suffir à faire la vérification sur ce sous ensemble il est d'une part nécessaire d'avoir les deux propriétés qui vont suivre; puis il faut d'autre part avoir une méthode de vérification sur le sous-ensemble discret des configurations.

Propriété 4.1.1 *Il faut pouvoir associer à toute séquence réelle une séquence discrète qui vérifie la même formule à laquelle nous nous intéressons.*

Cette première propriété permet de réduire la vérification d'un cadre dense vers un cadre discret.

Propriété 4.1.2 *Il faut que pour toute séquence discrète σ vérifant la formule à laquelle nous nous intéressons et pour toute séquence ρ ayant pour discrétisation la séquence σ alors ρ doit satisfaire cette même formule.*

Cette seconde propriété impose que le modèle discret obtenu ne contient pas d'informations supplémentaires par rapport au modèle original pouvant engendrer une

erreur dans la vérification.
Si cela est respecté, nous pouvons alors transférer les résultats obtenus dans le discret vers le continu.

4.2 Réduction par partitionnement

Dans cette partie nous allons rappeler la technique de partitionnement introduite par [ACD93] et qui permet de passer d'un domaine dense à un domaine fini. Les auteurs dans [ACD93] montrent que pour un graphe temporisé il existe un partitionnement de sorte que les configurations appartenant à une même classe satisfont les même propriétés d'atteignabilité.

Soient $\mathcal{P}$ l'ensemble des propositions atomiques et $\Sigma = 2^{\mathcal{P}}$ alors un graphe temporisé $\mathcal{M}$ sur Σ est le tuple défini comme suit: $\mathcal{M} = \langle \mathcal{L}, \Pi, \mathcal{X}, \delta \rangle$ où :

- $\mathcal{L} = \{\ell_1, \cdots, \ell_n\}$ est un ensemble fini de nœuds de contrôles,
- Π est une fonction dans $[\mathcal{L} \mapsto \Sigma]$ associant un ensemble de propositions atomiques à chaque nœud de contrôle.
- $\mathcal{X}$ est un ensemble fini d'horloges.
- $I : \ell \mapsto \mathcal{C}_{\mathcal{X}}$ est une fonction qui associe à chaque nœud de contrôle des contraintes sur les horloges du système. Ces contraintes sont appelées invariant du nœud ℓ.
- δ est l'ensemble d'arètes. Chaque arète étant un tuple $e = (\ell, g, R, \ell')$ où:
  - $\ell, \ell' \in \mathcal{L}$ sont le nœud source et le nœud cible.
  - $g = \bigwedge_{i \in \{1,\cdots,n\}} a \sim x_i \sim b \wedge \bigwedge_{j,k \in \{1,\cdots,n\}} c \sim x_j - x_k \sim d$
    avec $a, b, c, d \in \mathbb{N}$, $\sim \in \{<, \leq\}$ et les x_i sont des horloges donc $\forall i \in \{1, \cdots, n\} : x_i \in \mathcal{X}$, sont des contraintes sur les horloges.
  - $R \subseteq \mathcal{X}$ est l'ensemble des horloges qui devront être remisent à zéro lorsque la transition e est effectuée.

Une configuration du graphe temporisé est la donnée d'un noeud de contrôle et de la valuation des horloges. On peut définir une relation d'équivalence sur l'ensemble infini des valuations des horloges, de sorte que si l'on effectue un partitionnement par rapport à cette relation les classes obtenues sont telles que: toutes les configurations d'une même classe satisfont les mêmes propriétés d'atteignabilité.

Définition 4.1 Une relation d'équivalence $\cong$ peut être définie entre les configurations du graphe temporisé. On dit que deux configurations $\langle \ell_1, \nu_1 \rangle$ et $\langle \ell_2, \nu_2 \rangle$ sont

équivalentes que l'on note par $\langle \ell_1, \nu_1 \rangle \cong \langle \ell_2, \nu_2 \rangle$ si et seulement si $\ell_1 = \ell_2$ et $\nu_1 \cong \nu_2$. □

Définition 4.2 Avant de définir la relation d'équivalence $\cong$ on note par c_x la constante la plus grande à laquelle l'horloge x est comparée dans les gardes du système. On dira que deux valuations ν_1 et ν_2 sont équivalentes que l'on note $\nu_1 \cong \nu_2$ si et seulement si les conditions suivantes sont vérifiées:

1. Pour toute horloge $x \in \mathcal{X}$ nous avons soit que $\nu_1(x) > c_x$ et $\nu_2(x) > c_x$ soit que $\lfloor \nu_1(x) \rfloor = \lfloor \nu_2(x) \rfloor$.
2. Pour toute horloge $x \in \mathcal{X}$ telle que $\nu_1(x) \leq c_x$ ou $\nu_2(x) \leq c_x$ nous avons $fract(\nu_1(x)) = 0$ si et seulement si $fract(\nu_2(x)) = 0$ où $fract$ dénote la partie fractionnelle.
3. Pour toutes horloges $x, y \in \mathcal{X}$ telles que $(\nu_1(x) - \nu_1(y)) \leq c_x$ et $(\nu_1(y) - \nu_1(x)) \leq c_y$, nous avons $fract(\nu_1(x)) \sim fract(\nu_1(y))$ si et seulement si $fract(\nu_2(x)) \sim fract(\nu_2(y))$ avec $\sim \in \{<, \leq, >, \geq\}$.

□

On constate facilement que l'ensemble des classes d'équivalence que nous noterons $\mathcal{R}$ est fini. Ces classes d'équivalence sont souvent appelées régions.
La figure 4.1 présente les différentes classes d'équivalence existantes pour un même nœud de contrôle, dans le cas où $\mathcal{X} = \{x, y\}$ et $c_x = c_y = 3$.
On distingue ici des classes à zéro dimension (les points), à une dimension (les segments ouverts et les demi-droites ouvertes) et à deux dimensions (les polygones ouverts bornés par des ségments et les polygones ouverts non bornés). On dénombre également quinze régions terminales, qui sont les demi-droites ouvertes et les polygones ouverts non bornés situés dans le quadrant du plan défini par $x > 3$ et $y > 3$.

Définition 4.3 Région bornée ou fermée
Nous appelons région *fermée*, toute classe obtenue à l'aide de la relation d'équivalence $\cong$ telle qu'il existe au moins une horloge x ayant $fract(\nu(x)) = 0$ pour toute valuation ν de cette région. □

Dans le cadre des systèmes temporisés (leurs seules variables sont des horloges), les régions bornées correspondent à des régions dont les valuations qu'elles contiennent sont instables par le moindre passage de temps. C'est à dire que pour toute valuation d'une telle région, le moindre passage du temps, engendre un changement de région.

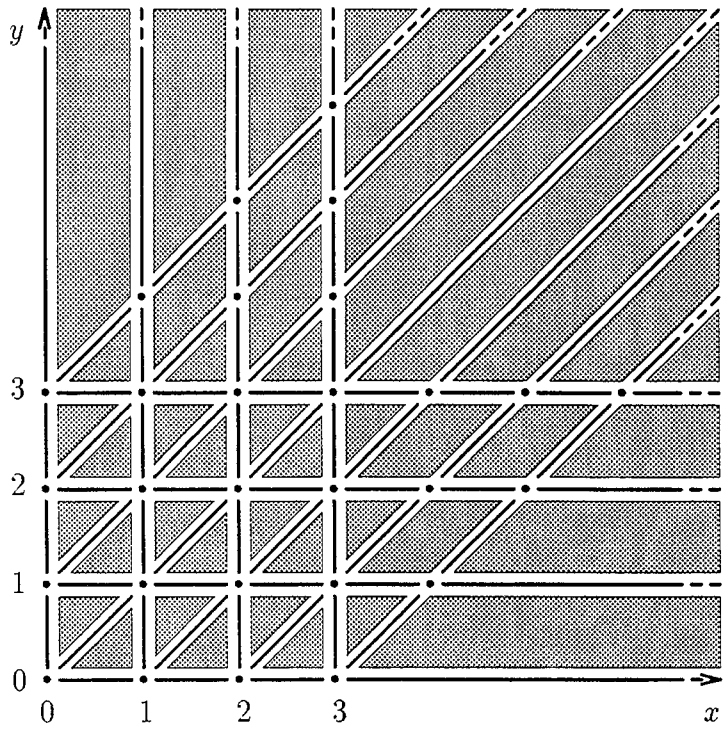


Figure 4.1: Partitionnement pour deux horloges

Définition 4.4 Région ouverte ou non bornée
Nous appelons *région ouverte*, toute région obtenue à l'aide de la relation d'équivalence $\cong$ et telle que pour toute horloge x, $fract(\nu(x)) \neq 0$ pour toute valuation ν de cette région. □

Toutes les régions qui ne sont pas ouvertes sont bornées et réciproquement.

Remarque 4.2.1 *La région qui succède à une région ouverte par le passage du temps est une région fermée. De même que la région qui succède à une région bornée par le passage du temps est une région ouverte.*

Lemme 4.1 Soient $\mathcal{H}$ un graphe temporisé et $\mathcal{R}$ l'ensemble des régions qui lui sont associées. Pour toute région $[\nu]$ de $\mathcal{R}$ telle qu'il existe au moins une horloge x qui a $\nu(x) \leq c_x$, il existe une unique région $[\nu']$ de $\mathcal{R}$ telle que $[\nu']$ est la région qui succède a $[\nu]$ par le passage du temps. ■

Dans le lemme 4.1, nous imposons une contrainte à $[\nu]$ qui est l'existence d'au moins une horloge x pour laquelle $\nu(x) \leq c_x$, de sorte à garantir qu'il existe une région qui lui est différente et qui lui succède. Car il est clair que si cette contrainte est fausse la région $[\nu]$ n'a pas de région qui lui soit différente et qui lui succède dans $\mathcal{R}$. La preuve de ce lemme est disponible dans la thèse de Rajeev Alur [Alu91].

L'intuition qui est derrière cette décomposition de l'espace infini des valuations des horloges est d'obtenir d'une part des classes dans lesquelles chaque garde du système prend une valeur unique vraie ou fausse dans tous les états d'une même classe et qui permet d'avoir le résultat énoncé dans 4.1.
Nous avons des décompositions au niveau des entiers car selon la position par rapport à un entier la valeur d'une garde pourrait être différente. Nous avons aussi des décompositions en diagonales car selon la position par rapport à des diagonales certaines gardes prennent des valeurs différentes, ce sont certaines gardes parmi celles qui sont des différences entre deux horloges du système, de plus c'est cette décomposition par rapport aux diagonales qui assure la notion de successeur unique (voir le lemme 4.1). De plus au-dela de la plus grande constante c_x à laquelle l'horloge x est comparée il est inutile de connaître la position de la valeur de x par rapport aux entiers car vu la monotonie des horloges (strictement croissantes) les gardes prennent chacune une valeur et ne changent plus jusqu'à la prochaine remise à zéro. C'est grace à ce dernier point que le nombre de classes est fini.

Dans la prochaine partie nous donnons le principe de la seconde technique de réduction; qui est la technique de la discrétisation.

4.3 Discrétisation

Les systèmes hybrides et temps réel opèrent sur le domaine dense des réels dû au temps continu. Mais plusieurs méthodes de vérification sont basées sur le fait que le temps est supposé discret (dans $\mathbb{Z}$). Ce que nous aimons savoir, c'est quand est-il est possible de reporter au continu le résultat de la vérification dans le discret. Ou plus encore quand est ce que le problème de savoir si les comportements réels (continus) du système considéré satisfont une certaine propriété peut être réduit au problème de savoir si les comportements observés (discrets) satisfont cette même propriété. La réduction par *discrétisation* consiste à ne considérer que des séquences d'exécution temporisées entières (toutes les configurations temporisées ont une valuation entière pour leur estampille t).

Définition 4.5 Discrétisation
Soit la séquence d'exécution temporisée $\rho = \{\langle \ell_i, \lambda_i, \nu_i, \mu_i, \tau_i \rangle\}_{i \in \mathbb{N}}$
On appelle discrétisation de ρ une séquence d'exécution temporisée ρ' définie comme

suit: $\rho' = \{\langle \ell_i, \lambda_i, \nu'_i, \mu_i, \tau'_i \rangle\}_{i \in \mathbb{N}}$

- $\forall i \in \mathbb{N},\ \tau'_i \in \mathbb{Z}$,
- $\tau'_{i+1} \geq \tau'_i$,
- $\tau'_0 = \tau_0$,
- $\forall i \in \mathbb{N}$, si $\tau_i \in \mathbb{Z}$ alors $\tau'_i = \tau_i$,
- $lim_{i \to \infty} \tau'_i = \infty$,
- $\nu'_0 = \nu_0$ et
- $\forall i > 0\ \nu'_i = \nu'_{i-1} + (\tau'_i - \tau'_{i-1})$.

□

Remarque 4.1 On peut avoir une séquence d'exécution temporisée ρ qui appartient à un système hybride $\mathcal{H}$ et qu'une discrétisation de ρ n'appartienne pas à $\mathcal{H}$. □

Plusieurs techniques de discrétisations existent. A chacune de ces techniques on peut associer certains avantages et certains inconvénients.
Il faut par discrétisation garder toute l'information nécessaire et rien que l'information nécessaire. Du moins lorsque certaines informations sont perdues ou rajoutées il ne faudrait pas que cela change le résultat du problème de vérification auquel on s'intéresse. Si par exemple nous nous intéressons au problème de l'atteignabilité propositionnelle, tel que dans le discret ce problème a pour réponse vraie et que les séquences discrètes permettant de répondre vraie n'appartiennent pas au système continu (c'est possible, voir la remarque 4.1) alors le transfert de cette réponse du discret vers le continu engendre une erreur.

Les problèmes qui peuvent être rencontrés dûs à la discrétisation sont:
il faut éviter le rajout d'informations (les discrétisations de séquences d'exécution temporisées du modèle sont des séquences d'exécution temporisées de ce même modèle), il faut aussi éviter d'en perdre (à toute séquence d'exécution temporisée du modèle nous pouvons associer une discrétisation qui appartient aussi a ce même modèle) .
Il faut garantir par exemple un fait très important, c'est que l'ordre du séquencement des événements dans le temps reste le même par discrétisation.

Exemple 4.1 Soit la séquence d'exécution temporisée:
$\rho = \langle \ell_0, 0\rangle\langle \ell_1, 0.4\rangle\langle \ell_2, 0.6\rangle\langle \ell_3, 1.5\rangle$

La séquence d'exécution temporisée $\rho_{d_1} = \langle \ell_0, 0\rangle\langle \ell_1, 1\rangle\langle \ell_2, 0\rangle\langle \ell_3, 1\rangle$ n'est pas une discrétisation de ρ.
Par contre $\rho_{d_2} = \langle \ell_0, 0\rangle\langle \ell_1, 1\rangle\langle \ell_2, 1\rangle\langle \ell_3, 2\rangle$ est une discrétisation de ρ. □

Plusieurs techniques de discrétisations existent, chacune d'elles a ses avantages et ses inconvénients. Certaines seront appliquées pour des modèles bien spécifiques d'autres pour des propriétés bien spécifiques. Donc le choix d'une technique de discrétisation est directement lié aux systèmes auxquels nous nous intéressons ainsi qu'aux propriétés que nous voulons vérifier sur ces systèmes. Parmi ces techniques de discrétisation nous pouvons en décrire brièvement quelques unes.

Il existe celle qui consiste à ramener tous les événements ayant lieu dans un intervalle de longueur 1 ($[u, u+1[$) au début de l'intervalle donc à l'instant entier de cet intervalle (u).

Exemple 4.2 Soit la séquence d'exécution temporisée ρ d'un système temps réel ayant une seule horloge. $\rho = \langle \ell_0, t_0\rangle\langle \ell_1, t_1\rangle \cdots \langle \ell_n, t_n\rangle$ Alors la séquence associée à ρ par la discrétisation énoncée ci-dessus est:
$\rho_{d_1} = \langle \ell_0, \lfloor t_0 \rfloor\rangle\langle \ell_1, \lfloor t_1 \rfloor\rangle \cdots \langle \ell_n, \lfloor t_n \rfloor\rangle$ □

Une autre technique consistera à ramener tout ce qui se passe dans un intervalle de longueur 1 ($]u, u+1]$) à la fin de l'intervalle donc à l'instant entier de cet intervalle ($u+1$).

Exemple 4.3 On considère toujours la même séquence $\rho = \langle \ell_0, t_0\rangle\langle \ell_1, t_1\rangle \cdots \langle \ell_n, t_n\rangle$, alors la séquence associée à ρ par la seconde discrétisation énoncée ci-dessus est:
$\rho_{d_2} = \langle \ell_0, \lceil t_0 \rceil\rangle\langle \ell_1, \lceil t_1 \rceil\rangle \cdots \langle \ell_n, \lceil t_n \rceil\rangle$ □

Remarque 4.2 Tous les événements ayant lieu à des instants entiers, doivent avoir lieu aux mêmes instants par discrétisation. C'est le cas des deux discrétisations énoncées ci-dessus.
Donc en particulier la discrétisation d'une séquence discrète devra être elle même. □

Une technique de dicrétisation différente des deux précédentes a été introduite dans [HMP92]. Nous appelons cette discrétisation, technique de numérisation uniforme.

Comme les deux précédentes, tous les événements ayant lieu à des instants entiers sont invariants par discrétisation. Par contre contrairement aux deux cas précédents les événements ayant lieu dans un même intervalle unitaire ouvert $(]u, u+1[)$ ne seront pas forcément tous projetés vers l'entier inférieur (u) ou vers l'entier supérieur $(u+1)$, mais certains seront projetés vers l'entier inférieur (u) et les autres vers l'entier supérieur $(u+1)$. Ce partage se fait comme suit:
Nous fixons un réel $\varepsilon \in [0,1[$ puis nous effectuons une sorte d'arrondissement par rapport à cet ε; en effet tout réel ayant une partie fractionnelle inférieure à ε sera projeté vers l'entier inférieur sinon (si sa partie fractionnelle est strictement supérieure à ε) il sera projeté vers l'entier supérieur.

Définition 4.6 Soit $x \in \mathbb{R}$ et soit $\varepsilon \in [0,1[$, nous dénotons par $[x]_\varepsilon$ la discrétisation du réel x moyennant ε. $[x]_\varepsilon$ est définie comme suit:

Si $x - \lfloor x \rfloor \leq \varepsilon$ alors $[x]_\varepsilon = \lfloor x \rfloor$ sinon $[x]_\varepsilon = \lceil x \rceil$.

Soit une séquence d'exécution temporisée
$\rho = \langle \ell_0, \nu_0, t_0 \rangle \langle \ell_1, \nu_1, t_1 \rangle \cdots \langle \ell_n, \nu_n, t_n \rangle$ alors sa discrétisation moyennant ε est:
$[\rho]_\varepsilon = \langle \ell_0, d(\nu_0), [t_0]_\varepsilon \rangle \langle \ell_1, d(\nu_1), [t_1]_\varepsilon \rangle \cdots \langle \ell_n, d(\nu_n), [t_n]_\varepsilon \rangle$
où : $\forall i > 0 \; d(\nu_i) = d(\nu_{i-1}) + ([t_i]_\varepsilon - [t_{i-1}]_\varepsilon)$ et $d(\nu_0) = 0$. □

Exemple 4.4 Soit la séquence d'exécution temporisée d'un système ayant une horloge:
$\rho = \langle \ell_0, 0 \rangle \langle \ell_1, 0.4 \rangle \langle \ell_2, 0.6 \rangle \langle \ell_3, 1.5 \rangle$
et soit $\varepsilon = 0.45$ alors la discrétisation de ρ moyennant le quantum ε est:
$[\rho]_\varepsilon = \langle \ell_0, 0 \rangle \langle \ell_1, 0 \rangle \langle \ell_2, 1 \rangle \langle \ell_3, 2 \rangle$ □

La technique de numérisation introduite par [HMP92], que nous venons de décrire peut être vue simplement comme suit :
C'est comme si l'on avait divisé la base du temps ($\mathbb{R}$) en fenêtres de même taille, cette taille est égale à 1. Puis on a décalé ce fenêtrage de ε par rapport à zéro. Nous obtenons un fenêtrage de la base du temps tel que chaque fenêtre contient un seul entier. La discrétisation moyennant ε consiste à projeter tous les événements ayant lieu dans une fenêtre au niveau de l'entier contenu dans cette même fenêtre, tout en gardant l'ordre de séquencement des événements.
Nous appelons cette technique de discrétisation, la numérisation uniforme par opposition à la technique que nous développerons dans le prochain chapitre et qui sera appelée numérisation non uniforme et qui est plus générale.

Le principe de la numérisation uniforme est aussi décrit par la figure 4.2 qui suit:

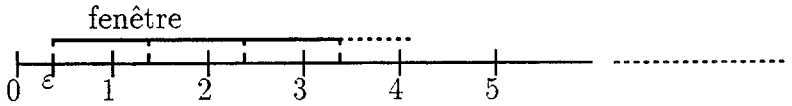


Figure 4.2: Numérisation uniforme

Définition 4.7 Pour toute séquence d'exécution ρ, on définit l'ensemble des numérisations qui lui sont associées que l'on note par $[\rho]$ comme suit:
$[\rho] = \{[\rho]_\epsilon | \epsilon \in [0,1[\}$. □

Définition 4.8 On définit l'ensemble de toutes les numérisations d'un ensemble de séquences temporisées Π que l'on note par $[\Pi]$ comme suit:

$$[\Pi] = \bigcup_{\rho \in \Pi} [\rho]$$

□

Définition 4.9 Fermeture par numérisation
Un ensemble de séquences d'exécution temporisées Π est dit *fermé par numérisation* si et seulement si pour toute séquence $\rho \in \Pi$ son ensemble de numérisations $[\rho]$ est inclus dans Π.
Une manière équivalente d'écrire cette même définition est:
Π est *fermé par numérisation* si et seulement si: $[\Pi] \subseteq \Pi$. □

Proposition 4.1 L'ensemble $\Pi_{\mathcal{H}}$ de toutes les séquences d'exécution temporisées d'un graphe temporisé dont toutes les gardes sont des inégalités larges ($\leq$), est fermé par numérisation. ■

Preuve 4.1 Soit $\rho = \langle \ell_0, \nu_0, t_0 \rangle \langle \ell_1, \nu_1, t_1 \rangle \cdots \langle \ell_n, \nu_n, t_n \rangle \cdots$ une séquence d'exécution temporisée du système considéré. Donc $\rho \in \Pi_{\mathcal{H}}$.

Il faut montrer que
$\forall \varepsilon \in [0,1[.\ [\rho]_\varepsilon \in \Pi_{\mathcal{H}}$.

Soit $\rho \in \Pi_{\mathcal{H}}$ alors $\forall i \in I\!N - \{0\}$ on a:
soit que $\ell_i \neq \ell_{i-1}$ alors $\exists e_i = (\ell_{i-1}, g_i, R_i, \ell_i)$ avec $\nu_{i-1} \models g_i$ et $\nu_i = \nu_{i-1}[R_i \mapsto 0] \models I(\ell_i)$
soit que $\ell_i = \ell_{i-1}$ alors $\forall t \in [0, t_i - t_{i-1}]$ $\nu_i = \nu_{i-1} + t \models I(\ell_i)$.

Il faut alors montrer que $\forall \varepsilon \in [0,1[$ si $\ell_i \neq \ell_{i-1}$ alors $d_\varepsilon(\nu_{i-1}) \models g_i$ et $d_\varepsilon(\nu_i) \models I(\ell_i)$ sinon il suffit de montrer que $d_\varepsilon(\nu_i) \models I(\ell_i)$.
Pour les deux cas il suffit de montrer que la différence pour chacun des indices entre la valeur d'origine dans la séquence ρ de chacune des horloges avec la valeur obtenue par numérisation est en valeur absolue inférieur à 1.

Pour toute horloge $x_k \in \mathcal{X}$ nous définissons un entier u_k tel que $0 \leq u_k < i$ et u_k est égal à l'indice de la dernière remise à zéro de l'horloge x_k dans la séquence ρ si elle existe sinon $u_k = 0$. On a alors pour toute horloge x_k, $d_\varepsilon(\nu_i(x_k)) = d_\varepsilon(\nu_{i-1}(x_k)) + ([t_i]_\varepsilon - [t_{i-1}]_\varepsilon)$ si R_i ne contient pas x_k ou que $\ell_i = \ell_{i-1}$ sinon $d_\varepsilon(\nu_i(x_k)) = 0$.
Nous avons donc: $d_\varepsilon(\nu_i(x_k)) = \sum_{j=u_k+1}^{i}([t_j]_\varepsilon - [t_{j-1}]_\varepsilon) = [t_i]_\varepsilon - [t_{u_k}]_\varepsilon$
Et nous avons $\nu_i(x_k) = t_i - t_{u_k}$.

Donc $d_\varepsilon(\nu_i(x_k)) - \nu_i(x_k) = ([t_i]_\varepsilon - t_i) - ([t_{u_k}]_\varepsilon - t_{u_k})$
Il est très facile de vérifier que nous avons:

$$|d_\varepsilon(\nu_i(x_k)) - \nu_i(x_k)| < 1 \tag{4.1}$$

Donc si $\nu_i(x_k) = v$ avec $v \in I\!N$ alors nous avons forcément $d_\varepsilon(\nu_i(x_k)) = \nu_i(x_k)$ car nous avons par définition $d_\varepsilon(\nu_i(x_k)) \in I\!N$ et 4.1. Donc dans ce premier cas si $\nu_i(x_k)$ ne falsifie pas de contrainte alors $d_\varepsilon(\nu_i(x_k))$ non plus.

Le deuxième cas est celui où il existe $v \in I\!N$ tel que $\nu_i(x_k) \in]v, v+1[$. Puisque $d_\varepsilon(\nu_i(x_k)) \in I\!N$ et que nous avons 4.1, il est clair que $d_\varepsilon(\nu_i(x_k))$ est égal soit à v soit à $v+1$. Donc dans ce second cas si $\nu_i(x_k)$ ne falsifie pas de contrainte et puisque les contraintes du système sont larges ($\leq$) alors $d_\varepsilon(\nu_i(x_k))$ ne falsifie pas de contraintes. En effet nous avons $v < \nu_i(x_k) < v+1$, $\nu_i(x_k)$ ne falsifie pas de contraintes et que les gardes du système sont larges donc le fait d'avoir $v \leq d_\varepsilon(\nu_i(x_k)) \leq v+1$ ne falsifiera pas de contraintes, puisque la contrainte en question imposera au moins que la valeur de l'horloge x_k soit dans $[v, v+1]$.

Donc finalement nous obtenons $\forall \varepsilon \in [0,1[$. $[\rho]_\varepsilon \in \Pi_{\mathcal{H}}$ donc $\Pi_{\mathcal{H}}$ est fermé par numérisation. ■

4.4 Conclusion

Nous avons décrit dans ce chapitre les deux principes de réduction de la vérification du continu vers le discret qui sont les techniques de partitionnement et de discrétisation. Nous avons décrit ces techniques telles qu'elles ont été introduites dans respectivement [ACD93] et [HMP92]. Nous verrons dans les chapitres qui suivent que nous pouvons utiliser ces techniques pour vérifier certains de nos systèmes. Par contre nous verrons que pour d'autres systèmes que nous considérons et qui sont assez complexes, il n'est pas possible d'utiliser ces deux techniques. Pour résoudre ce problème nous avons défini dans le chapitre 5 une nouvelle technique de discrétisation appelée numérisation non uniforme et dans le chapitre 6 une technique plus fine de partitionnement.

Chapitre 5

Systèmes hybrides munis de structures de données discrètes

Dans ce chapitre nous considérons des automates d'états finis munis de structures de données discrètes non bornées et de variables continues. Nous ciblons exactement le cas des systèmes manipulant d'une part des compteurs discrets et une pilè, puis d'autre part, des variables continues à pentes constantes. L'utilisation de structures de données discrètes non bornées permet de considérer des systèmes avec un contrôle plus puissant que celui des systèmes hybrides linéaires sans structures de données discrètes. Les systèmes que nous considérons permettent entre autres de raisonner sur des notions importantes telles que le nombre d'occurrences de certains événements sur certaines séquences d'exécutions.
De plus, l'utilisation de variables continues à pente constante permet de raisonner par exemple sur le temps séparant deux événements et sur les durées de certaines phases dans un certain intervalle d'exécution.

Nous nous intéressons aussi dans ce chapitre aux liens de simulations entre les systèmes à variables discrètes et systèmes à variables continues. Lorsque ces liens peuvent être établis, l'analyse de systèmes hybrides se raméne à l'analyse de systèmes discrets. Ce qui entraînera le raisonnement sur les délais et les durées à un raisonnement sur les occurrences d'événements.

Dans ce chapitre nous présentons des résultats de décidabilité pour plusieurs sous-classes de tels modèles de systèmes hybrides. En effet, nous commençons par donner une méthode de décision pour la vérification de systèmes temporisés à pile (STP).
Nous présentons ensuite une extension de cette méthode qui nous permet de vérifier les systèmes temporisés à pile munis de compteurs de contrôles monotones (STPCM). Puis nous considérons le cas des systèmes temporisés à pile munis de compteurs d'observations non-monotones (STPCO). Nous montrons que nous pouvons dans ce

cas réduire le problème de la vérification de formules d'atteignabilité sur de tels systèmes au problème du langage vide sur des langages hors-contexte avec des contraintes [BER93, BER94b].

Nous considérons par la suite les systèmes temporisés à pile que nous enrichissons d'une variable continue d'observation. Nous montrons que dans ce cas nous pouvons utiliser la technique de discrétisation introduite dans [HMP92] et que nous décrivons dans le chapitre 4 pour réduire le problème d'atteignabilité sur ces systèmes à un problème de langage vide sur des langages hors-contextes avec des contraintes [BER93, BER94b].

Puis finalement nous considérons des systèmes temporisés à pile munis d'un intégrateur de contrôle. Nous montrons que moyennant certaines restrictions sur ces systèmes, le problème d'atteignabilité est décidable. Pour cela nous utilisons une nouvelle technique de discrétisation (numérisation non-uniforme). Nous verrons que la restriction que nous imposons est assez contraignante mais l'intérêt que nous portons à ces systèmes est de montrer d'une part toute la difficulté à laquelle nous faisons face dès que nous essayons d'arrêter une horloge du système (intégrateur) puis d'autre part, d'illustrer notre nouvelle technique de discrétisation (numérisation non uniforme). C'est aussi un cas de systèmes hybrides se situant à la frontière entre les systèmes décidables et ceux qui sont indécidables.

5.1 Systèmes temporisés à pile

Dans cette partie nous considérons des modèles permettant de représenter le comportements de systèmes hybrides à l'aide d'un système ayant un ensemble de nœuds de contrôle fini, une pile et un ensemble de variables continues; plus précisément nous considérons uniquement les systèmes temporisés donc nous nous suffirons à ne considérer que des horloges comme variables continues du système. Donc dans cette partie notre intérêt se porte sur les systèmes que nous avons dénoté par STP dans le chapitre 2.
Nous allons montrer qu'étant donné un système temporisé à pile (STP), le problème d'atteignabilité est décidable. Pour cela nous réduisons ce problème au problème de la vérification de formules d'atteignabilité propositionnelle, lequel problème est équivalent à celui de la vérification de l'atteignabilité d'un nœud de contrôle particulier. Plus précisément, décider si une formule d'atteignabilité est satisfaite ou non, partant d'une configuration α d'un système temporisé à pile $\mathcal{H}$, est équivalent à décider si un nœud spécial est atteignable ou non à partir de α dans un nouveau système qui sera une extension appropriée du système initial. Pour résoudre ce problème d'atteignatibilité, nous donnons une extension de la technique de partitionnement (graphe de régions) introduite dans [ACD93] et que nous décrivons dans le chapitre 4. En effet, nous con-

sidérons une relation d'équivalence sur l'ensemble des valuations des horloges qui, d'une part préserve les propriétés d'atteignabilité propositionnelles et d'autre part induit un ensemble fini de classes d'équivalences. Nous montrons que la relation d'équivalence préserve les propriétés d'atteignabilité pour signifier que toutes les propriétés de ce type gardent leurs valeurs par partitionnement. En d'autres termes, c'est que les propriétés de ce type ont une valeur unique dans une même classe. Cette technique de partitionnement nous permettra donc de réduire le problème d'atteignabilité d'un noeud de contrôle sur une structure dense en un problème d'atteignabilité sur une structure discrète. Mais la structure discrète que nous obtiendrons sera infinie étant donné que le système que nous considérons est muni d'une pile non bornée. Mais nous montrons que le problème d'atteignabilité peut être réduit au problème d'atteignabilité dans un automate à pile, qui est un problème équivalent au problème du langage vide pour les langages hors-contextes.

Donc pour montrer la décidablité du problème d'atteignabilité sur des STP, nous utilisons une technique de partitionnement. Pour cela nous commençons par considérer un partitionnement de l'ensemble des valuations de $\mathcal{X}$ (donc des horloges, car ce sont les seules variables continues du système).
Dénotons par $\mathcal{E}$ cet ensemble, c'est-à-dire, $\mathcal{E} = [\mathcal{X} \to \mathbb{R}]$.
Dans [ACD93], une relation d'équivalence $\cong$ sur $\mathcal{E}$ est introduite pour les graphes temporisés de sorte que toute paire de configurations du système temporisé ayant le même nœud de contrôle et des valuations équivalentes selon $\cong$, les deux ensembles de nœuds de contrôles atteignables sont identiques (un pour chacune des deux configurations).

L'idée de base qui est derrière la définition de la relation d'équivalence $\cong$ est:
D'une part que les valuations d'une même classe satisfont les mêmes gardes du systèmes. Donc les seules valuations importantes pour chaque horloge x sont les valuations en dessous de la plus grande constante à laquelle cette horloge x est comparée dans le système, nous noterons cette constante c_x. Car au delà de cette constante c_x toutes les valeurs de l'horloge x satisfont les mêmes gardes, ceci est dû au fait que les horloges sont monotones(croissantes). Puis d'autre part que chaque classe ait une unique classe qui la succède par passage du temps.
Nous montrons dans ce qui suit que nous pouvons adapter partitionnement similaire dans le cas des systèmes temporisés à pile.

Nous considérons dans ce qui suit une relation d'équivalence $\cong$ qui est presque identique à celle qui est introduite dans [ACD93] (voir la définition 4.2 dans le chapitre 4). La seule différence est dûe au fait que dans les systèmes que nous considérons, les gardes ne contiennent pas de différences entre les horloges. Donc le partitionnement que nous obtiendrons est moins fin que celui obtenu par la relation d'équivalence définie dans [ACD93], nous retrouvons cette différence au niveau du troisème point de la définition.

Remarque 5.1 Il faut noter que dans tous les modèles étudiés dans ce chapitre, peuvent être étendus en permettant d'avoir des différences entre les horloges dans les gardes. Nous ne le faisons pas, juste pour des raisons de clartés dans la présentation. □

Définition 5.1 Nous notons par c_x la constante la plus grande à laquelle l'horloge x est comparée dans les gardes du système. On dira que deux valuations ν_1 et ν_2 sont équivalentes que l'on note $\nu_1 \cong \nu_2$ si et seulement si les conditions suivantes sont vérifiées:

1. Pour toute horloge $x \in \mathcal{X}$ nous avons soit que $\nu_1(x) > c_x$ et $\nu_2(x) > c_x$ soit que $\lfloor \nu_1(x) \rfloor = \lfloor \nu_2(x) \rfloor$
2. Pour toute horloge $x \in \mathcal{X}$ telle que $\nu_1(x) \leq c_x$ ou $\nu_2(x) \leq c_x$ nous avons $fract(\nu_1(x)) = 0$ si et seulement si $fract(\nu_2(x)) = 0$ où $fract$ dénote la partie fractionnelle.
3. Pour toutes horloges $x, y \in \mathcal{X}$ telles que $\nu_1(x) \leq c_x$ et $\nu_1(y) \leq c_y$, nous avons $fract(\nu_1(x)) \sim fract(\nu_1(y))$ si et seulement si $fract(\nu_2(x)) \sim fract(\nu_2(y))$ avec $\sim \in \{<, \leq, >, \geq\}$.

□

Maintenant, tenant compte du fait que $\mathcal{H}$ peut contenir des contraintes de la forme $x \equiv_n c$, nous considérons alors l'équivalence $\simeq$ qui est définie comme un raffinement de la relation d'équivalence $\cong$ par rapport aux ensembles finis décrits par les $\equiv_n$ qui apparaissent dans $\mathcal{H}$. En effet, si par exemple une horloge x est référée m fois dans $\mathcal{H}$ à l'aide de la relation $\equiv$
$x \equiv_{n_1} c_1$, $x \equiv_{n_2} c_2$, $\cdots$ et $x \equiv_{nm} c_m$
Soit n_k le plus petit commun multiple des n_i ($n_k = PPCM(n_1, \cdots, n_m)$), il suffit alors de faire le raffinement par rapport aux éléments de l'ensemble suivant:
$\{0,]0,1[, 1,]1,2[, 2, \cdots,]n_k - 1, n_k[\}$.
Si en plus de ces $x \equiv_{n_i} c_i$, la variable x apparaît dans des comparaisons et que c_x soit la plus grande constante à laquelle x est comparée il faut alors raffiner par rapport à $\{]c_x, c_x + 1[, c_x + 1, \cdots,]c_x + n_k - 1, c_x + n_k[, c_x + n_k\}$.

Donc pour chaque élément de cet ensemble nous considérons toutes les classes déjà établies à l'aide $\cong$.

Le lemme suivant énonce que si deux configurations sont équivalentes selon $\simeq$ alors si on leur applique l'opération de remise à zéro d'un sous-ensemble d'horloges nous obtenons deux nouvelles configurations équivalentes.

Lemme 5.1 Soient deux valuations ν, ν' équivalentes selon $\simeq$ ($\nu \simeq \nu'$) et soit $R \subseteq \mathcal{X}$ un sous-ensemble d'horloges, alors $\nu[R \mapsto 0] \simeq \nu'[R \mapsto 0]$. ■

La preuve du lemme 5.1 est triviale, il suffit d'appliquer directement la définition de l'équivalence $\simeq$.

Définition 5.2 Soit $[\mathcal{E}]$ l'ensemble quotient de $\mathcal{E}$ par rapport à la relation $\simeq$. L'ensemble $[\mathcal{E}]$ peut être muni par une fonction successeur *succ* entre deux classes d'équivalence qui code la progression du temps. La fonction *succ* est définie de la manière suivante:
Pour tout $\nu_1, \nu_2 \in \mathcal{E}$, $succ([\nu_1]) = [\nu_2]$ si et seulement si $\nu_1 \not\simeq \nu_2$, $\exists t \in \mathbb{R}^+$ tel que $\nu_2 \simeq \nu_1 + t$, et $\forall t' \in \mathbb{R}^+$, $0 \leq t' < t$, soit que $\nu_1 + t' \simeq \nu_1$ ou bien que $\nu_1 + t' \simeq \nu_2$. □

Lemme 5.2 $\forall \nu, \forall \nu'\ \nu \simeq \nu'$
$\forall t :\ [\nu + t] = succ(\nu)\ \exists t' :\ \nu' + t' \simeq \nu + t\ \wedge [\nu' + t'] = succ([\nu'])$. ■

Nous énonçons dans ce qui suit le lemme qui nous permet de ramener le problème d'atteignabilité auquel nous nous intéressons d'un cadre dense vers un cadre discret.

Lemme 5.3 Soient ν, $\nu' \in \mathcal{E}$. Alors, $\nu \simeq \nu'$ implique que pour toute formule d'atteignabilité propositionelle φ, pour tout $\ell \in \mathcal{L}$ et pour tout $\lambda \in \Gamma^*$, $\langle \ell, \lambda, \nu \rangle \models \varphi$ si et seulement si $\langle \ell, \lambda, \nu' \rangle \models \varphi$. ■

Preuve 5.1 Soient la configuration $\alpha_0 = \langle \ell_0, \lambda_0, \nu_0 \rangle$ et la formule d'atteignabilité propositionnelle φ, nous supposons que nous avons $\alpha_0 \models \varphi$. Nous pouvons écrire $\varphi = \exists \Diamond p$ que nous pouvons réécrire en $\varphi = \exists \Diamond \bigvee_{j=1}^{m} at - \ell_{n_j}$ qui peut aussi se réécrire en $\varphi = \bigvee_{j=1}^{m} \exists \Diamond at - \ell_{n_j}$, avec $at - \ell_{n_j}$ une proposition atomique qui est vraie uniquement dans ℓ_{n_j} et que la proposition p est vraie dans les états ℓ_{n_j} pour tout $1 \leq j \leq m$. Alors il existe une séquence d'exécution $\rho \in \mathcal{SE}(\alpha_0, \mathcal{H})$ telle que $\rho = \langle \ell_0, \lambda_0, \nu_0 \rangle \langle \ell_1, \lambda_1, \nu_1 \rangle \cdots \langle \ell_n, \lambda_n, \nu_n \rangle \cdots$ et $\ell_n \in \bigcup_{j=1}^{m} \{\ell_{n_j}\}$
Soit ν'_0 une valuation telle que $\nu_0 \simeq \nu'_0$ il faut alors montrer qu'il existe une séquence $\rho' \in \mathcal{SE}(\alpha'_0, \mathcal{H})$ telle que ρ' passe par les mêmes nœuds de contrôle que ρ. Il existe donc une séquence ρ' telle que $\rho' = \langle \ell'_0, \lambda'_0, \nu'_0 \rangle \langle \ell'_1, \lambda'_1, \nu'_1 \rangle \cdots \langle \ell'_n, \lambda'_n, \nu'_n \rangle \cdots$ avec pour tout $i \in \mathbb{N}$ $\ell_i = \ell'_i$ et $\lambda_i = \lambda'_i$. Nous montrerons même plus: en effet, nous montrerons aussi que pour tout $i \in \mathbb{N}$ $\nu_i \simeq \nu'_i$. Pour prouver ce résultat nous allons procéder par récurrence sur l'indice k des configurations apparentes dans ρ et ρ'. Pour $k = 0$ nous avons $\nu_0 \simeq \nu'_0$, $\ell_0 = \ell'_0$ et $\lambda_0 = \lambda'_0$, il n'y a plus rien à montrer dans ce cas. Nous

supposons que ce résultat est vrai à l'indice $k-1$, nous allons le montrer pour l'indice k.

Pour cela nous montrons que nous pouvons construire le k^{iemme} séquencement de ρ'. Nous montrons que la k^{iemme} transition s'opérant dans ρ, nous pouvons l'effectuer de manière similaire pour ρ' de sorte à garantir que la k^{iemme} configuration de ρ est équivalente à la k^{iemme} configuration dans ρ'. C'est à dire que la k^{iemme} configuration de ρ a le même nœud de contrôle et la même configuration de la pile que la k^{iemme} configuration de ρ', de plus que leurs valuations sont équivalentes ($\nu_k \simeq \nu'_k$). Dans cette preuve nous utilisons principalement le lemme 4.1 qui dit que pour toute région $[\nu] \in [\mathcal{E}]$ la région qui la succède par passage du temps est unique. Comme nous l'avons vu dans la sémantique des STP, la k^{iemme} transition apparent dans ρ peut être soit du type transition temporelle (correspondant à une progression du temps) sinon du type transition discrète (correspondant au changement du nœud de contrôle).

- Transition discrète:
  Dans ce cas nous avons un passage dans ρ de la configuration: $\langle \ell_{k-1}, \lambda_{k-1}, \nu_{k-1} \rangle$ à la configuration $\langle \ell_k, \lambda_k, \nu_k \rangle$ avec $\nu_k = \nu_{k-1}$. Il existe donc une arête $e = (\ell_{k-1}, g, A, \gamma, R, \ell_k) \in \delta$ telle qu'en utilisant cette arête nous avons un passage dans ρ de la configuration $\langle \ell_{k-1}, \lambda_{k-1}, \nu_{k-1} \rangle$ à la configuration $\langle \ell_k, \lambda_k, \nu_k \rangle$ avec $\nu_k = \nu_{k-1}[R \mapsto 0]$. Nous avons aussi $\nu_{k-1} \models g$, $\lambda_{k-1} = A.\ \lambda''$, $\lambda_k = \gamma.\ \lambda''$ et $\nu_k \models I(\ell_k)$. Or nous avons dans ρ' la configuration $\langle \ell'_{k-1}, \lambda'_{k-1}, \nu'_{k-1} \rangle$ avec $\ell'_{k-1} = \ell_{k-1}$, $\lambda'_{k-1} = \lambda_{k-1}$ et $\nu_{k-1} \simeq \nu'_{k-1}$ donc $\nu'_{k-1} \models g$. Cela implique qu'au niveau de cette configuration de ρ' l'arête e peut être utilisée pour obtenir la configuration $\langle \ell'_k, \lambda'_k, \nu'_k \rangle$ avec $\ell'_k = \ell_k$, $\lambda'_k = \lambda_k$ et $\nu'_k = \nu'_{k-1}[R \mapsto 0]$ donc $\nu'_k \simeq \nu_k$ d'après le fait que $\nu'_{k-1} \simeq \nu_{k-1}$ et en appliquant la proposition 5.1 nous avons donc aussi que $\nu_k \models I(\ell_k)$.

- Transition temporelle:
  Dans ce cas nous avons un passage dans ρ de la configuration $\langle \ell_{k-1}, \lambda_{k-1}, \nu_{k-1} \rangle$ à la configuration $\langle \ell_k, \lambda_k, \nu_k \rangle$ avec $\ell_k = \ell_{k-1}$, $\lambda_k = \lambda_{k-1}$, $\nu_k = [\nu_{k-1} + (t_k - t_{k-1})]$ (avec $t_k - t_{k-1}$ étant le temps écoulé entre les deux configurations) et $\forall t \in [0, t_k - t_{k-1}]\ \nu_{k-1} + t \models I(\ell_{k-1})$. Il est facile de voir que dans ρ' nous pouvons construire une transition temporelle de sorte à passer de la configuration $\langle \ell_{k-1}, \lambda_{k-1}, \nu'_{k-1} \rangle$ avec $\nu'_{k-1} \simeq \nu_{k-1}$ à la configuration $\langle \ell_k, \lambda_k, \nu'_k \rangle$ avec $\nu'_k \simeq \nu_k$. Cette construction est une application directe du lemme 4.1 et du lemme 5.2. En effet le lemme 4.1 indique que la région qui succède à une région donnée est unique. Le second lemme dit que si dans un nœud on peut laisser passer le temps pour passer d'une région $[\nu]$ à la région qui lui succéde $succ([\nu])$ alors pour toute valuation $\nu' \simeq \nu$ on peut laisser passer le temps pour passer de la région $[\nu']$ à la région qui lui succéde $succ([\nu'])$.

Nous avons montré par récurrence que nous pouvons construire ρ' de sorte qu'à

tout indice j la j^{iemme} configuration de ρ' est équivalente à la j^{iemme} configuration de ρ. ■

Le lemme 5.3 permet de réduire le problème d'atteignabilité d'un graphe dense vers un problème d'atteignabilité sur une structure discrète. Cette structure correspond au graphe de régions introduit dans [ACD93] dans le cas des systèmes temporisés. Le graphe de régions est obtenu en considérant le quotient du graphe de configuration de $\mathcal{H}$ par rapport à la relation d'équivalence $\simeq$.

Dans ce qui suit nous donnons la définition de la construction du graphe de régions. Nous définissons une *région* comme un triplet $\langle \ell, \lambda, [\nu] \rangle$ où $\ell \in \mathcal{L}$, $\lambda \in \Gamma^*$, et $[\nu] \in [\mathcal{E}]$. Soit *Reg* l'ensemble de ces régions. Alors, le graphe de région de $\mathcal{H}$, dénoté $\mathcal{R}(\mathcal{H})$, est défini comme le graphe $\langle Reg, Edg \rangle$ où l'ensemble des sommets est *Reg* défini ci-dessus et l'ensemble des arcs *Edg* est défini comme le plus petit ensemble tel que:

- Chaque sommet (région) $\langle \ell, \lambda, [\nu_1] \rangle$ a un arc vers $\langle \ell, \lambda, succ([\nu_1]) \rangle$,
- Chaque sommet $\langle \ell, A \cdot \lambda, [\nu_1] \rangle$ a un arc vers $\langle \ell', \gamma \cdot \lambda, [\nu_1[R \mapsto 0]] \rangle$ pour chaque arc $(\ell, g, A, \lambda, R, \ell') \in \delta$ telle que $\nu_1 \models g$ et $\nu_1[R \mapsto 0] \models I(\ell')$.

Clairement, le graphe de région $\mathcal{R}(\mathcal{H})$ est infini étant donné que nous utilisons une pile non bornée. Rappelons que le problème d'atteignabilité sur les graphes auxquels nous nous intéressons est de savoir si une configuration particulière avec un nœud de contrôle particulier ciblé (que nous notons *cible*) est atteignable ou non en partant d'une configuration initiale donnée $\alpha = \langle \ell, \lambda, \nu \rangle$.
En d'autres termes, la valeur de la pile n'est pas importante dans les configurations ciblées (celles que nous désirons atteindre). Le seul fait important est que ce sont des nœuds à atteindre. Nous pouvons alors réduire ce problème d'atteignabilité au problème d'atteignabilité d'un nœud de contrôle dans un automate à pile sans entrée (input-free) $\mathcal{A}$ ayant $\mathcal{R}(\mathcal{H})$ comme graphe de configurations.
L'automate $\mathcal{A}$ est tel que:

- Le vocabulaire est Γ,
- l'ensemble des nœuds de contrôles est $\mathcal{L}' = \mathcal{L} \times [\mathcal{E}]$,
- l'ensemble des nœuds finaux est l'ensemble des nœuds $\langle cible, \varepsilon \rangle \in \mathcal{L}'$, pour tout $\varepsilon \in [\mathcal{E}]$.
- la relation de transition est donnée par l'ensemble d'arcs $(\langle \ell_1, [\nu_1] \rangle, A, \gamma, \langle \ell_2, [\nu_2] \rangle)$ tel que:
  - Soit $\ell_1 = \ell_2$, $\gamma = A$, et $succ([\nu_1]) = [\nu_2]$,

- ou bien $\exists(\ell_1, g, A, \gamma, R, \ell_2) \in \delta$ telle que $\nu_1 \models g$, $[\nu_2] = [\nu_1[R \mapsto 0]]$ et $\nu_2 \models I(\ell_2)$.

Théorème 5.1 Le problème de la vérification de formules d'atteignabilités sur des STP est décidable. □

Preuve 5.2 Le lemme 5.3 permet de réduire le problème d'atteignabilité d'une structure dense vers une structure discrète. Puis la construction du graphe des régions (structure discrète) permet d'obtenir un graphe muni d'une pile. Puis nous donnons la réduction du problème d'atteignabilité au problème d'atteignabilité d'un nœud de contrôle dans un automate à pile (sans entrées) $\mathcal{A}$ ayant $\mathcal{R}(\mathcal{H})$ comme graphe des configurations.

En effet un nœud cible dans $\mathcal{H}$ est atteignable à partir d'une configuration $\alpha = \langle \ell, \lambda, \nu \rangle$ si et seulement si l'automate $\mathcal{A}$ accepte un mot commençant au nœud de contrôle $\langle \ell, [\nu] \rangle$ avec la pile égale à λ. Ce problème correspond au problème du langage vide de langages hors-contextes (plus exactement au langage non-vide). Ce problème est bien connu comme étant décidable [Har78]. ■

5.2 Systèmes temporisés à pile munis de compteurs

Dans cette section nous allons dans une première partie munir par des compteurs monotones de contrôle les systèmes temporisés à pile que nous avons considéré dans la partie précédente. Puis dans une seconde partie nous munissons ces systèmes temporisés à pile par des compteurs d'observations.
Nous montrons dans le premier cas que le problème d'atteignabilité est réductible au problème du langage vide d'un langage hors-contexte, il est donc décidable.
Puis dans le second cas, nous montrons que le problème d'atteignabilité est réductible au problème du langage vide d'un langage hors-contexte munis de contraintes sur les occurrences des éléments de son vocabulaire, nous verrons que dans ce cas ce problème est lui même réductible au problème de satisfaisabilité d'une formule linéaire sur les entiers, lequel problème est décidable.

5.2.1 Systèmes temporisés à pile munis de compteurs de contrôle monotones

Dans cette partie nous portons notre intérêt sur la vérification du problème de l'atteignabilité sur des automates à piles temporisés enrichis par des variables de contrôle discrètes monotones que nous appelons compteurs de contrôle monotones. Ce qui nous permet d'avoir un contrôle plus puissant et donc de décrire des systèmes ayant un comportement qui dépend des valeurs de certaines de ces variables discrètes. Ces variables discrètes sont les compteurs monotones que nous introduisons. Nous rappelons que nous notons ces systèmes STPCM.

Pour ces systèmes (STPCM), nous pouvons traiter ces compteurs de manière identique à celle des horloges. En effet nous pouvons les considérer comme des horloges discrètes. Ceci est dû à la monotonie de ces compteurs (caractéristique fondamentale des horloges), car sans cette monotonie l'ensemble des classes que nous obtenons par partitionnement ne serait plus fini. Vue la monotonie des compteurs (les compteurs ne décroissent jamais), les seules valeurs exactes importantes pour chaque compteur sont les valeurs qui se trouvent en dessous de la plus grande constante à laquelle ce compteur est comparé dans le système. Nous pouvons alors de manière identique à celle du cas temporisé pur, définir une relation d'équivalence $\simeq$ sur les valuations des horloges et des compteurs. Cette relation d'équivalence permet d'obtenir un nombre de classes fini, de telle sorte que les configurations ayant le même nœud de contrôle, le même contenu de la pile et des valuations équivalentes par rapport à la relation $\simeq$ satisfont les mêmes formules d'atteignabilité propositionnelle.
Nous donnons la relation d'équivalence $\simeq$ qui n'est autre qu'une extension directe de la relation d'équivalence définie pour le cas temporisé pur dans la partie précédente. En effet, dans le cas présent, nous tenons compte des valeurs des variables discrètes (compteurs monotones). Pour que deux valuations soient équivalentes, il faut que les valuations des variables continues soient équivalentes selon la relation d'équivalence définie pour les STP, mais il faut aussi que les valuations des variables discrètes soient égales.

Nous définissons donc la relation d'équivalence $\simeq$ de la manière suivante :
Pour toute variable $y \in \mathcal{Y}$ du système $\mathcal{H}$ nous associons la constante c_y qui est la plus grande constante comparée à y dans les gardes de $\mathcal{H}$. De même que nous associons à toute horloge $x \in \mathcal{X}$ une constante c_x qui est la plus grande constante comparée à x dans les gardes de $\mathcal{H}$. Nous considérons alors la relation d'équivalence $\cong$ telle que nous avons $\langle \nu, \mu \rangle \cong \langle \nu', \mu' \rangle$ si et seulement si les conditions suivantes sont vraies:

- $\nu \cong \nu'$
- $\forall z \in \mathcal{Y}$, si $\mu(z) \leq c_z$, alors $\mu(z) = \mu'(z)$ et si $\mu(z) > c_z$ alors $\mu'(z) > c_z$.

Comme dans la partie précédente 5.1, nous définissons la relation $\simeq$ comme le raffinement de la relation d'équivalence $\cong$ définie ci-dessus par rapport à tous les $\equiv_n$ qui apparaissent dans les gardes du système $\mathcal{H}$. Nous pouvons alors réduire le problème d'atteignabilité dans $\mathcal{H}$ au problème d'atteignabilité dans un automate à pile que nous pouvons décider comme le problème du langage vide d'un langage hors-contexte. Nous obtenons alors le résultat suivant.

Théorème 5.2 Le problème de la vérification de formules d'atteignabilités sur systèmes temporisés à pile munis de compteurs de contrôle est décidable. □

Preuve 5.3 La preuve de ce théorème découle directement de la preuve du théorème 5.1, le seul changement est dans la construction du graphe de région moyennant la nouvelle relation d'équivalence. Cette relation d'équivalence comme nous l'avons vu est une extension directe de la relation d'équivalence définie pour les STP. En effet puisque d'une part pour que deux valuations soient équivalentes il faut que les valuations des variables continues soient équivalentes selon la relation d'équivalence définie pour les STP, mais il faut aussi que les valuations des variables discrètes soient égales; puis d'autre part étant donnée la monotonie des compteurs il suffit de considérer pour chacun d'eux uniquement un nombre fini de valeurs. ■

5.2.2 Systèmes temporisés à pile munis de compteurs d'observations

Nous nous intéressons dans cette partie au problème de la vérification de formules d'atteignabilités pour des systèmes temporisés à pile munis de compteurs d'observations. Ces systèmes sont différents des précédents, en effet les compteurs que nous introduisons n'ont pas la même fonctionnalité que ceux qui ont été introduits dans la partie précédente. Ces compteurs sont différents dans le sens où d'une part dans la partie précédente les compteurs utilisés étaient monotones alors que dans la partie actuelle les compteurs ont une évolution quelconque, puis d'autre part, dans la partie précédente les compteurs que nous introduisons sont des compteurs de contrôle alors que les compteurs que nous introduisons dans la partie actuelle sont des compteurs d'observation. Nous rappelons que les compteurs de contrôle apparaissent dans les gardes ou dans les invariants associés à un nœud du système afin de restreindre certains comportements alors que les compteurs d'observations n'apparaissent pas dans les gardes mais servent plutôt à exprimer certaines propriétés d'invariance et d'atteignabilité contraintes que nous devons vérifier sur les systèmes que nous considérons. Les compteurs que nous introduisons auront donc pour tâche d'observer et

d'enregistrer certaines informations découlant des séquences d'exécution du système. Nous rappelons que nous dénotons ces systèmes par STPCO.

Nous considérons donc dans cette partie le problème de la vérification de formules d'atteignabilités sur des STPCO. Soient $\mathcal{H}$ un tel système, $\alpha = \langle \ell, \lambda, \nu, \mu \rangle$ une configuration de $\mathcal{H}$ et φ une formule d'atteignabilité donnée par:

$$\exists\Diamond(\pi \wedge f \wedge \bigwedge_{y \in \mathcal{Y}} ((a_y \sim_1^y y \sim_2^y b_y) \wedge \bigwedge_{i=1}^{m_y} y \approx_i^y c_i^y)) \tag{5.1}$$

où $f \in \mathcal{C}_{\mathcal{X}}$, π étant une formule d'état, les $\sim_1^y$ et $\sim_2^y$ sont dans $\{<, \leq\}$, les $\approx_i^y$ sont dans $\{\equiv_n, \not\equiv_n\}$, les a_y sont dans $\mathbb{Z} \cup \{-\infty\}$, les b_y sont dans $\mathbb{Z} \cup \{+\infty\}$ et les c_i^y sont dans $\mathbb{N}$.

Puis supposons que nous voulons décider si nous avons $\alpha \models \varphi$.

Nous avons montré dans la partie 5.1 que si nous considérons uniquement la formule d'atteignabilité $\exists\Diamond(\pi \wedge f)$, alors le problème de vérification qui nous intéresse se réduit à contrôler si le langage acceptée par un automate à pile $\mathcal{A}$ construit à partir de $\mathcal{H}$ est vide ou non. Dans le cas présent, ce problème de vérification est réduit au *problème du langage vide pour des langages contraints* [BER93, BER94c, BEH94, BEH95] ce qui consiste à contrôler si l'automate $\mathcal{A}$ accepte une séquence qui satisfait certaines contraintes sur les compteurs d'observations invoqués dans la formule d'atteignabilité φ. En d'autres mots, nous devons décider s'il existe une séquence acceptée par $\mathcal{A}$ telle que pour chaque compteur d'observation y, si nous accumulons à sa valeur initiale les effets d'emprunter certaines transitions dans la séquence (c'est à dire $\mu(y)$) alors la valeur obtenue que nous notons par v_y, doit satisfaire $(a_y \sim_1^y v_y \sim_2^y b_y) \wedge \bigwedge_{i=1}^{m_y} v_y \approx_i^y c_i^y$. L'effet de chaque transition de l'automate à pile est soit nul si la transition correspond à la progression du temps soit alors donné par la fonction κ du système $\mathcal{H}$ si cela correspond à l'exécution d'une transition gardée de $\mathcal{H}$.

Pour résoudre ce problème du langage vide pour des langages contraints, nous transformons l'automate $\mathcal{A}$ en une grammaire hors-contexte qui lui est équivalente.
Soit la grammaire hors-contexte $\mathcal{G} = (\Sigma, N, Prod, S)$ où $\Sigma = \emptyset$ est le vocabulaire, N est l'ensemble des non terminaux, $Prod$ est l'ensemble des productions (souvent appelé, ensemble des règles), et S est le symbole de départ (appelé souvent axiome). Il est à noter que toutes les productions sont de la forme $A \rightarrow \gamma$ où $\gamma \in N^*$. Par conséquent, si $L(\mathcal{G}) \neq \emptyset$, alors $L(\mathcal{G}) = \{\epsilon\}$ où ϵ est la séquence vide (mot vide). Nous rappelons que l'automate à pile de départ est sans entrée (input-free) et que donc la grammaire que nous obtenons a un vocabulaire vide, c'est la raison pour laquelle nous avons pu écrire que si le langage reconnu par la grammaire est non vide alors il est égal au mot vide.
Nous utilisons la transformation standard donnée dans [HU79] qui permet d'obtenir la grammaire hors-contexte équivalente à un automate à pile.

Chaque production (pouvant être multiple) dans $\mathcal{G}$ est obtenue à partir d'une transition de l'automate $\mathcal{A}$. Nous pouvons alors définir une fonction κ' qui décrit l'effet de l'application des productions dans $\mathcal{G}$ sur les variables d'observations. C'est à dire κ' associe à chaque production p de $\mathcal{G}$ et chaque compteur d'observation y un entier qui doit être rajouté à y à chaque application de p.

Dans ce qui suit nous dénotons par ($\Longrightarrow$) la relation de dérivation dans la grammaire $\mathcal{G}$ et par ($\overset{*}{\Longrightarrow}$) sa fermeture transitive-réflexive. Nous utilisons des indices afin de préciser l'ensemble des productions ou de séquences de productions que nous utilisons dans la dérivation.

Donc notre problème de langage vide avec contraintes consistera à décider s'il existe ou non une dérivation σ dans $\mathcal{G}$ donnée par

$$S \Longrightarrow_{p_1} \lambda_1 \cdots \Longrightarrow_{p_n} \lambda_n \Longrightarrow_{p_{n+1}} \epsilon \tag{5.2}$$

où les p_i sont dans $Prod$ et les λ_i sont des séquences non vides de non terminaux (dans N^+) et telles que $\bigwedge_{y\in\mathcal{Y}}((a_y \sim_1^y v_y \sim_2^y b_y) \wedge \bigwedge_{i=1}^{m_y} v_y \approx_i^y c_i^y)$ où $v_y = \mu(y) + \sum_{j=1}^{n+1} \kappa'(p_j, y)$.

Nous montrons dans ce qui suit que le problème de décision peut être résolu par la résolution d'un problème de programmation linéaire pour les entiers, c'est à dire que nous construisons une formule linéaire Ω qui est satisfaisable si et seulement si une dérivation dans $\mathcal{G}$ comme celle décrite ci-dessus (σ) existe.

Nous définissons les ensembles de variables qui sont invoquées dans la formule Ω. Nous associons à chaque compteur d'observation y une variable v_y qui aura la valeur de y à la fin de la séquence σ. Plus encore, à chaque production $p \in Prod$ nous associons une variable w_p qui enregistre le nombre d'applications de la production p dans σ.

Pour commencer nous devons exprimer le fait que les v_y et les w_p sont positives par la formule POS:

$$(\bigwedge_{y\in\mathcal{Y}} v_y \geq 0) \wedge (\bigwedge_{p\in Prod} w_p \geq 0)$$

Alors, pour chaque compteur d'observation y, la valeur de v_y est liée avec les valeurs des w_p par la formule ACC_y:

$$v_y = \mu(y) + \sum_{p\in Prod} \kappa'(p, y) \cdot w_p$$

Nous devons aussi considérer les contraintes sur les compteurs d'observation invoqués dans la formule φ (5.1). Par conséquent, nous définissons pour tout compteur y la contrainte linéaire COND_y:

$$(a_y \sim_1^y v_y \sim_2^y b_y) \wedge \bigwedge_{i=1}^{m_y} v_y \approx_i^y c_i^y$$

Nous devons aussi définir les contraintes sur les w_p. Ces contraintes devront exprimer le fait que toute occurrence d'un non terminal apparant le long de σ doit être réduit de sorte que nous obtenons la séquence ϵ à la fin de σ. Donc dans la dérivation σ, pour chaque non terminal A, le nombre des réductions de A, c'est à dire de l'application de production p ayant A comme partie gauche (que nous nommons A-productions), doit être égal au nombre de A-introductions, c'est à dire le nombre d'occurrences de A dans la partie droite des productions appliquées et plus un si A est égal à S. Ce plus un est dû au fait que S existe au départ, il faut donc comptabiliser cette existence en rajoutant la valeur un au nombre de créations du non terminal S. C'est le problème de flux dans un graphe.

Pour toute production $p : A \rightarrow \gamma$, nous commençons par noter respectivement $lhs(p)$ (left hand side) et $rhs(p)$ (right hand side) la partie gauche et la partie droite de p, c'est à dire $lhs(p) = A$ et $rhs(p) = \gamma$. Soient $\gamma \in N^*$ et $A \in N$, nous notons par $|\gamma|_A$ le nombre d'occurrences de A dans γ.

Alors pour tout non terminal A nous définissons la formule REDUCT_A:

$$\sum_{p \in Prod} |lhs(p)|_A \cdot w_p = |S|_A + \sum_{p \in Prod} |rhs(p)|_A \cdot w_p$$

Un autre fait que nous devons exprimer est qu'un non terminal est réduit si et seulement si il est atteignable en appliquant les productions participantes à la dérivation de σ. En effet considérons une production $p = B \rightarrow B$ et supposons que B n'apparaît jamais dans σ. Nous pouvons alors associer n'importe quelle valeur positive à w_p et satisfaire encore les contraintes REDUCT_A pour tout A. Ce qui n'est pas acceptable puisque les valeurs v_y calculées en utilisant les contraintes ACC_y ne correspondent pas nécessairement aux valeurs qui peuvent être obtenues à partir d'une dérivation existante dans $\mathcal{G}$. Nous devons donc exprimer le fait que pour tout non terminal A il existe une A-production p avec $w_p > 0$ si et seulement si A est S ou il apparaît dans les λ_i le long de σ.

Pour des raison de clarté nous avons besoin d'introduire une notation. Nous disons qu'une séquence de productions $\pi \in Prod^*$ est *élémentaire* si toutes ses productions s'appliquent à des non terminaux différents. Etant donné un non terminal A, nous définissons Π_A comme étant l'ensemble des séquences élémentaires π dans $Prod^*$ telles que $\exists \gamma, \gamma' \in N^*$, $S \stackrel{*}{\Longrightarrow}_\pi \gamma \cdot A \cdot \gamma'$.

Remarque 5.2 L'ensemble Π_A est fini et peut être calculé de manière automatique. □

Nous définissons alors pour tout non terminal A la formule REACH_A:

$$\sum_{p \in Prod} |lhs(p)|_A \cdot w_p > 0 \Leftrightarrow \bigvee_{\pi \in \Pi_A} \bigwedge_{p \in \pi} w_p > 0.$$

Finalement, la formule Ω est définie par:

$$\mathsf{POS} \wedge (\bigwedge_{A \in N} \mathsf{REDUCT}_A \wedge \mathsf{REACH}_A) \wedge (\bigwedge_{y \in \mathcal{Y}} \mathsf{ACC}_y \wedge \mathsf{COND}_y)$$

Puisque le problème de satisfaisabilité des formules linéaires sur les entiers est décidable [Dan48], nous obtenons le résultat suivant:

Théorème 5.3 Le problème de la vérification de formules d'atteignabilité pour de STPCO est décidable. □

Pour des raisons de clarté nous avons traité indépendamment les STPCM dans la partie précédente et les STPCO dans cette partie. Or nous pouvons facilement voir que le problème de la vérification de formules d'atteignabilité est aussi décidable dans le cas où nous considérons des modèles ayant ces deux types de compteurs. C'est à dire que le problème de la vérification de formules d'atteignabilités sur des systèmes temporisés à pile munis de compteurs monotones de contrôle et de compteurs d'observations est décidable. Pour voir cela, il suffit de construire la grammaire $\mathcal{G}$ à partir de l'automate à pile utilisé pour la justification du théorème 5.2 (vu dans la partie 5.2.1). Nous obtenons alors le résultat suivant:

Théorème 5.4 Le problème d'atteignabilité pour des systèmes temporisés à pile munis de compteurs monotones de contrôles et de compteurs d'observations (STPCMO) est décidable. □

5.3 Systèmes temporisés à pile munis de variables continues

Cette partie concerne l'extension des systèmes temporisés à pile à l'aide de variables continues. Nous considérons dans cette partie deux classes de systèmes hybrides, soit d'une part les systèmes temporisés à pile munis d'une variable d'observation soit d'autre part les systèmes temporisés à pile munis d'un intégrateur de contrôle. Nous

montrons que pour les systèmes temporisés à pile munis d'une variable d'observation, le problème de la vérification de formules d'atteignabilité est décidable. Puis nous montrons que pour les systèmes temporisés à pile munis d'un intégrateur de contrôle le problème de la vérification de formules d'atteignabilité larges est décidable. Nous rappelons que les formules d'atteignabilité larges sont les formules d'atteignabilité dont les contraintes sont larges (c'est à dire que toutes les inégalités que la formule contient sont dans $\{\leq, \geq\}$). Pour cela, nous utilisons la technique de discrétisation pour réduire respectivement ces problèmes de vérification au problème d'atteignabilité dans un automate à pile munis d'un compteur d'observation ou d'un compteur de contrôle monotone. Nous verrons que dans le cas où nous munissont les STP par un intégrateur de contrôle, la technique de numérisation uniforme n'est plus applicable; nous avons alors introduit une nouvelle technique de discrétisation qui généralise la numérisation uniforme et que nous appelons numérisation non uniforme. Cette technique nous permettra d'analyser ces STP munis d'un intégrateur de contrôle. Nous commençons par donner la description de cette technique dans ce qui suit.

5.3.1 Numérisation non uniforme

Nous introduisons dans cette partie la notion de numérisation non uniforme qui étend la notion de numérisation uniforme définie dans [HMP92] et que nous avons rappelé dans la partie 4.3. Nous appelons *quantum de numérisation* toute valeur $\epsilon \in [0, 1[$. Nous appelons *séquence de numérisation* toute séquence infinie de quanta de numérisation $\vec{\epsilon} = \{\epsilon_i\}_{i \in I\!N}$. En particulier, nous disons que la séquence de numérisation est *uniforme* si tous les ϵ_i sont égaux. Soit un temps $t \in I\!R_{\geq 0}$, nous définissons, pour tout quantum de numérisation $\epsilon \in [0, 1[$, le temps entier tel que:
$[t]_\epsilon =$ si $t \leq \lfloor t \rfloor + \epsilon$ alors $\lfloor t \rfloor$ sinon $\lceil t \rceil$.

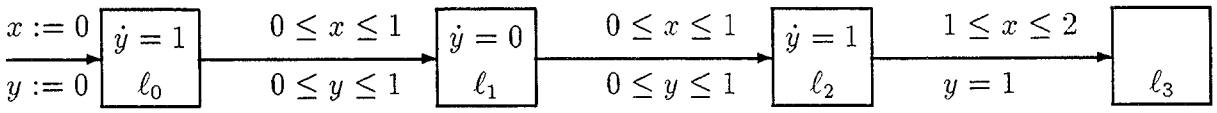
Maintenant, considérons une trace temporisée $\rho = \{\langle \ell_i, t_i \rangle\}_{i \in I\!N}$, et une séquence de numérisation $\vec{\epsilon} = \{\epsilon_i\}_{i \in I\!N}$. Alors, la *numérisation* de ρ par rapport à $\vec{\epsilon}$ est la trace entière $[\rho]_{\vec{\epsilon}} = \{\langle \ell_i, [t_i]_{\vec{\epsilon}(t_i)} \rangle\}_{i \in I\!N}$ où $\forall t \in I\!R_{\geq 0}$. $\vec{\epsilon}(t) = \epsilon_{\lfloor t \rfloor}$. Cette définition de la numérisation par rapport à un vecteur de numérisation peut s'étendre de manière naturelle aux séquences d'exécution temporisées. Soient la séquence d'exécution temporisée $\sigma = \{\langle \ell_i, \lambda_i, \nu_i, t_i \rangle\}_{i \in I\!N}$ et une séquence de numérisation $\vec{\epsilon} = \{\epsilon_i\}_{i \in I\!N}$ alors la *numérisation* de σ par rapport à $\vec{\epsilon}$ est la séquence d'exécution temporisée entière $[\rho]_{\vec{\epsilon}} = \{\langle \ell_i, \lambda_i, \nu_i^{\vec{\epsilon}}, [t_i]_{\vec{\epsilon}} \rangle\}_{i \in I\!N}$ avec : pour tout $i > 0$ et tel que la transition du nœud ℓ_{i-1} au nœud ℓ_i ne remet pas à zéro x $\nu_i^{\vec{\epsilon}}(x) = \nu_{i-1}^{\vec{\epsilon}} + \partial(\ell_{i-1}, x)([t_i]_{\vec{\epsilon}} - [t_{i-1}]_{\vec{\epsilon}})$ sinon si la transition du nœud ℓ_{i-1} au nœud ℓ_i remet à zéro x alors $\nu_i^{\vec{\epsilon}}(x) = 0$ sinon si $i = 0$ alors $\nu_i^{\vec{\epsilon}}(x) = 0$ aussi.

La notion de numérisation définie dans [HMP92] correspond à la notion de numérisation uniforme. Nous retrouvons l'utilisation de cette notion de numérisation uniforme lorsque nous raisonnons sur des systèmes temporisés larges (les contraintes qui

apparaissent dans les gardes sont larges), car pour ces systèmes, l'ensemble des traces générées par les séquences d'exécution sont fermées par numérisation (voir chapitre 4). Nous rappelons brièvement qu'un modèle est fermé par numérisation s'il contient toutes les numérisations de ses séquences.

Cette notion de numérisation uniforme est aussi utilisée dans le cas des systèmes temporisés larges avec une variable d'observation continue, nous retrouvons ce cas dans [KPSY93]. Nous allons montrer dans ce qui suit que nous pouvons étendre ces résultats aux systèmes temporisés à pile munis d'une variable d'observation (STPVO).

Par contre, cette notion ne permet pas de raisonner sur les systèmes à intégrateurs. Considérons par exemple le système 1-STPI (systèmes temporisés à pile munis d'un intégrateur que nous appelons $\mathcal{H}$, ce système possède une seule horloge x et un intégrateur y, il est représenté par la figure suivante:



et soit σ la séquence d'exécution
$\langle \ell_0, (x = 0, y = 0), 0\rangle\langle \ell_0, (x = 0.5, y = 0.5), 0.5\rangle\langle \ell_1, (x = 0.5, y = 0.5), 0.5\rangle$
$\langle \ell_1, (x = 1, y = 0.5), 1\rangle\langle \ell_2, (x = 1, y = 0.5), 1\rangle\langle \ell_2, (x = 1.5, y = 1), 1.5\rangle$
$\langle \ell_3, (x = 1.5, y = 1), 1.5\rangle$:

Nous pouvons alors voir que le système $\mathcal{H}$ décrit ci-dessus n'a pas d'exécution entière générée par une numérisation uniforme de $Trace_T(\sigma)$. En effet, si nous prenons n'importe quelle séquence de numérisation uniforme telle que tous ses éléments sont égaux à $\epsilon \in [0, 0.5[$ nous obtenons alors la trace entière:
$\rho_1 = \langle \ell_0, 0\rangle\langle \ell_0, 1\rangle\langle \ell_1, 1\rangle\langle \ell_1, 1\rangle\langle \ell_2, 1\rangle\langle \ell_2, 2\rangle\langle \ell_3, 2\rangle$, alors que si nous considérons que $\epsilon \in [0.5, 1[$, nous obtenons la trace
$\rho_2 = \langle \ell_0, 0\rangle\langle \ell_0, 0\rangle\langle \ell_1, 0\rangle\langle \ell_1, 1\rangle\langle \ell_2, 1\rangle\langle \ell_2, 1\rangle\langle \ell_3, 1\rangle$.
Nous pouvons alors vérifier que ces traces ne peuvent être générées par des séquences d'exécution, puisqu'à partir du temps global de ρ_1 (respectivement ρ_2) nous pouvons déduire que la transition partant de ℓ_2 allant vers ℓ_3 est exécutée lorsque la valeur de y est 2 (respectivement 0), ce qui falsifie la garde $y = 1$.

Par contre, le système $\mathcal{H}$ que nous considérons a deux séquences d'exécution entières générant des traces qui sont des numérisations *non uniformes* de $Trace_T(\sigma)$. Ces numérisations sont:
$\rho_3 = \langle \ell_0, 0\rangle\langle \ell_0, 0\rangle\langle \ell_1, 0\rangle\langle \ell_1, 1\rangle\langle \ell_2, 1\rangle\langle \ell_2, 2\rangle\langle \ell_3, 2\rangle$ et
$\rho_4 = \langle \ell_0, 0\rangle\langle \ell_0, 1\rangle\langle \ell_1, 1\rangle\langle \ell_1, 1\rangle\langle \ell_2, 1\rangle\langle \ell_2, 1\rangle\langle \ell_3, 1\rangle$;
la trace ρ_3 (respectivement ρ_4) est obtenue lorsque les valeurs du temps dans l'intervalle $[0, 1]$ sont numérisées par rapport à un $\epsilon_1 \in [0.5, 1[$ (respectivement $\epsilon_1 \in [0.\, 0.5[$), et les

valeurs du temps dans l'intervalle $[1,2]$ par rapport à un $\epsilon_2 \in [0, 0.5[$ (respectivement $\epsilon_2 \in [0.5, 1[$).

5.3.2 STP munis d'une variable continue d'observation

Soit $\mathcal{H}$ un STP large que nous munissons d'une variable d'observation continue nous rappelons que nous notons ces systèmes par STPVO; à l'aide d'une telle variable nous pouvons exprimer des invariants relatifs à la durée de certaines phases du système. Soit z cette variable. Nous montrons dans ce qui suit que le problème de vérification des formules d'atteignabilité sur de tels systèmes est décidable dans le cas où nous commençons l'exécution à une configuration entière.
Pour cela nous utilisons le résultat de numérisation prouvé dans [KPSY93], qui dit que si une contrainte $z \prec c$ (avec $\prec \in \{<, \leq\}$) est satisfaite le long d'une séquence σ à une position k et si nous supposons que la valeur initiale de chaque variable est 0, alors $z \prec c$ est aussi satisfaite le long d'une numérisation uniforme de σ à la même position k. Le fait d'initialiser les variables à la valeur 0 n'est qu'une formalité car les résultats que nous énonçons restent valables pour toute autre initialisation entière.

Nous énonçons ici un lemme qui servira à la preuve de la proposition 5.1. Le lemme est le suivant:

Lemme 5.4 Soient $0 < a_1 \leq a_2 \leq \cdots \leq a_m \leq 1$ et $0 < b_1 \leq b_2 \leq \cdots \leq b_n \leq 1$ deux suites finies croissantes. Si $\sum_{i=1}^{m} a_i \prec \sum_{j=1}^{n} b_j$, où $\prec \in \{\geq, >\}$ alors il existe $\epsilon \in [0,1[$ tel que $\sum_{i=1}^{m} [a_i]_\epsilon \prec \sum_{j=1}^{n} [b_j]_\epsilon$ ■

La preuve de ce lemme est donnée dans [KPSY93].

Proposition 5.1 Soit $\rho = \{\langle \ell_i, t_i \rangle\}_{i \in I\!N}$ une trace temporisée. Alors, s'il existe $m \in I\!N$ tel que $\sum_{i=0}^{m} \partial(\ell_i, z) \cdot (t_{i+1} - t_i) \prec c$ avec $\prec \in \{<, \leq\}$, alors il existe un quantum de numérisation $\epsilon \in [0,1[$ tel que $\sum_{i=0}^{m} \partial(\ell_i, z) \cdot ([t_{i+1}]_\epsilon - [t_i]_\epsilon) \prec c$. ■

Preuve 5.4 Nous avons $\sum_{i=0}^{m} \partial(\ell_i, z) \cdot (t_{i+1} - t_i) \prec c$ que nous pouvons réécrire en:

$$\sum_{i=0}^{m} \partial(\ell_i, z) \cdot t_{i+1} \prec \sum_{i=0}^{m} \partial(\ell_i, z) \cdot t_i + c$$

Or pour chaque expression $\partial(\ell_i, z) \cdot t_k$ avec $k = i$ ou $k = i+1$ nous pouvons écrire $t_k = \lfloor t_k \rfloor + \Upsilon_k$ avec $\Upsilon_k \in [0,1[$. Nous pouvons donc réécrire l'expression comme suit:

$$\overbrace{1 + \cdots + 1}^{\partial(\ell_i,z)\cdot\lfloor t_k \rfloor \text{ fois}} + \overbrace{\Upsilon_k + \cdots + \Upsilon_k}^{\partial(\ell_i,z)\cdot\lfloor t_k \rfloor \text{ fois}}$$

Nous utilisons alors le lemme 5.4 pour obtenir l'existence d'un $\epsilon \in [0,1[$ tel que: $\sum_{i=0}^{m} \partial(\ell_i, z) \cdot ([t_{i+1}]_\epsilon - [t_i]_\epsilon) \prec c$ ■

Il a été montré dans [HMP92] que pour tout système temporisé large, toute séquence d'exécution σ et toute séquence de numérisation uniforme $\vec{\epsilon}$, il existe une séquence d'exécution σ' telle que $[Trace(\sigma)]_{\vec{\epsilon}} = Trace(\sigma')$.
Donc par la proposition 5.1, nous pouvons déduire que pour les systèmes temporisés ayant une variable d'observation z, pour toute séquence d'exécution σ qui commence à partir d'une configuration entière et où la contrainte $z \prec c$ est satisfaite à une position k, il existe une exécution entière σ' et une séquence de numérisation uniforme $\vec{\epsilon}$ telle que $[Trace(\sigma)]_{\vec{\epsilon}} = Trace(\sigma')$ et $z \prec c$ est satisfaite dans σ' à la position k.
Ce fait peut être étendu aux systèmes munis de pile et de compteurs. Car la numérisation préserve les séquences d'opérations qui s'effectuent sur les structures de données discrètes. En effet, une numérisation entraîne que la séquence d'exécution que nous obtenons possède le même séquencement des opérations sur les structures de données discrètes que le séquencement des ces opérations dans la séquence d'exécution d'origine. Nous obtenons alors la proposition suivante:

Proposition 5.2 Soient $\mathcal{H}$ un système temporisé à pile large muni d'une variable continue d'observation, α une configuration entière de $\mathcal{H}$ et $\sigma \in \mathcal{SET}(\alpha, \mathcal{H})$. Alors, il existe une séquence de numérisation uniforme $\vec{\epsilon}$ et $\sigma' \in \mathcal{SET}_Z(\alpha, \mathcal{H})$, telle que :

$$[Trace(\sigma)]_{\vec{\epsilon}} = Trace(\sigma')$$

■

Preuve 5.5 D'après la proposition 5.1 à toute séquence continue satisfaisant à un indice donnée la contrainte relative à la variable continue d'observation, nous pouvons associer par numérisation une séquence discrète qui satisfait cette même contrainte à ce même indice. De plus si la séquence continue appartient au système $\mathcal{H}$ alors sa numérisation appartient au système $\mathcal{H}$, car $\mathcal{H}$ est un système temporisé large et donc d'après la proposition 4.1 il est fermé par numérisation. ■

Par la proposition 5.2, nous pouvons déduire que la vérification de formules d'atteignabilité pour les STP munis d'une variable continue d'observation, peut être résolue en ne considérant que les valuations entières des variables continues. (horloge + variable d'observation z). Toutes les horloges deviennent donc des compteurs monotones de contrôle alors que la variable z devient un compteur d'observation. Par

conséquent, en utilisant un raisonnement similaire à celui que nous utilisons dans la partie 5.2.2, la vérification de formules d'atteignabilité sur $\mathcal{H}$ est réduite au problème d'atteignabilité dans un automate à pile ayant des contraintes sur des compteurs d'observations, ce qui est réductible à un problème de programmation linéaire dans les entiers. Pour cela nous utilisons des arguments similaires à ceux utilisés pour le graphe des régions. Nous pouvons considérer une relation d'équivalence qui induit un nombre fini de classes. Cette relation d'équivalence concerne les horloges du système de sorte que les configurations ayant le même nœud de contrôle et des valuations entières équivalentes satisfont les mêmes formules d'atteignabilité. Par conséquent nous pouvons construire un automate à pile avec des compteurs d'observation (au moins un, correspondant à z) qui reconnaît l'ensemble des séquences d'exécution entières du système d'origine $\mathcal{H}$. Le problème de la vérification de formules d'atteignabilité sur $\mathcal{H}$ est alors réductible au problème d'atteignabilité dans un automate à pile avec des contraintes sur les compteurs d'observations. Ce problème est décidable comme nous l'avons vu dans la partie 5.2.1.

Théorème 5.5 Soient $\mathcal{H}$ un STP large muni d'une variable d'observation continue linéaire(STPVO), α une configuration entière de $\mathcal{H}$, et φ une formule d'atteignabilité. Alors, le problème de la vérification $\alpha \models \varphi$ est décidable. □

Preuve 5.6 D'après la proposition 5.2, à toute séquence continue de $\mathcal{H}$ qui satisfait la formule d'atteignabilité φ nous pouvons associer une séquence discrète qui satisfait cette même formule d'atteignabilité. Il suffit donc de faire la vérification sur le modèle extrait de $\mathcal{H}$ qui reconnaît uniquement toutes les séquences d'exécution discrètes de $\mathcal{H}$. Nous obtenons un automate à pile $\mathcal{A}$ où la variable d'observation continue est transformée en un compteur d'observation. Nous retombons alors dans le cas des systèmes temporisés à pile munis de compteurs d'observations. Comme nous l'avons vu dans la partie précédente (voir le théorème 5.3) le problème d'atteignabilité dans de tels systèmes est décidable. Il est réduit au problème de la satisfiabilité d'une formule linéaire sur les entiers. ■

5.3.3 STP + un intégrateur de contrôle

Dans cette partie nous essayons d'avoir d'une part un contrôle plus puissant au niveau des systèmes temporisés à pile et d'autre part de se rapprocher de la frontière entre les sous-classes décidables et les sous-classes indécidables. Ce contrôle nous permet de spécifier certaines restrictions des comportements acceptés des systèmes que nous étudions. Ce contrôle se fera en fonction de la durée de certaines phases. Donc dans cette partie nous munissons les systèmes temporisés à pile d'un intégrateur de contrôle.

Puis nous définissons la notion de séquence d'exécution temporisée complète qui va nous servir à simplifier la preuve du résultat principal de cette partie. Elle va simplifier la preuve énonçant qu'à toute séquence d'exécution nous pouvons associer au moins un vecteur de numérisation, de sorte que la numérisation de cette séquence par ce vecteur de numérisation soit aussi une séquence d'exécution temporisée du système.

Définition 5.3 Séquence d'exécution temporisée complète
Nous dirons qu'une séquence d'exécution temporisée $\sigma = \langle \ell_i, \lambda_i, \nu_i, t_i \rangle_{i \in I\!N}$ est complète si pour tout $k \in I\!N$ il existe un indice i_k tel que $t_{i_k} = k$. □

Nous appelons trace temporisée complète toute trace temporisée générée par une séquence d'exécution temporisée complète.

Proposition 5.3 Toute séquence d'exécution temporisée d'un système $\mathcal{H}$ peut être transformée en une séquence d'exécution temporisée complète de $\mathcal{H}$. ■

Preuve 5.7 Toute séquence d'exécution peut être complétée en rajoutant des configurations temporisées correspondant aux valeurs entières omises (manquantes).
En effet, soit $\sigma = \langle \ell_i, \lambda_i, \nu_i, t_i \rangle_{i \in I\!N}$ une séquence d'exécution temporisée alors pour tout indice $i \geq 0$ tel que $t_i \neq t_{i+1}$ (donc nécessairement $\ell_i = \ell_{i+1}$ et $\lambda_i = \lambda_{i+1}$), soient $\{u_1, u_2, \cdots, u_m\}$ l'ensemble des entiers entre t_i et t_{i+1}. Nous pouvons alors insérer entres les configurations temporisées $\langle \ell_i, \lambda_i, \nu_i, t_i \rangle$ et $\langle \ell_{i+1}, \lambda_{i+1}, \nu_{i+1}, t_{i+1} \rangle$ la séquence de configurations temporisées suivante:

$\langle \ell_i, \lambda_i, \nu_i, t_i \rangle \langle \ell_i, \lambda_i, \nu_i^1, u_1 \rangle \cdots \langle \ell_i, \lambda_i, \nu_i^m, u_m \rangle \langle \ell_{i+1}, \lambda_{i+1}, \nu_{i+1}, t_{i+1} \rangle$

avec $\nu_i^j = [\nu_i + (j + u_1 - t_i)]$

Il est clair que la séquence d'exécution temporisée que nous obtenons est une séquence du système $\mathcal{H}$. ■

Avant d'énoncer les résultats, nous allons présenter dans la figure 5.1 un système temporisé muni d'un intégrateur tel que ce système possède une séquence continue qui satisfait une certaine propriété d'atteignabilité alors qu'il ne possède aucune séquence discrète. Ce système est décrit dans l'exemple 5.1.

Exemple 5.1

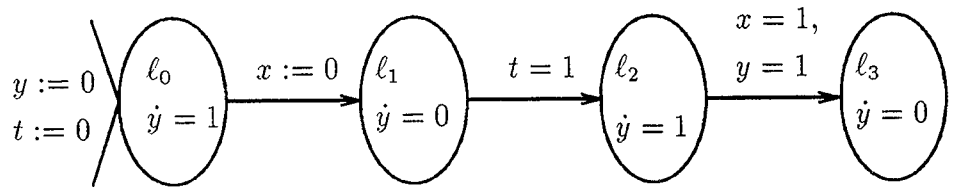


Figure 5.1: Système sans séquences discrètes

Dans la figure 5.1 nous décrivons un système temporisé muni d'un intégrateur de contrôle.

Le système possède deux horloges x et t et un intégrateur y. Le système se compose de quatre nœuds de contrôles. La propriété d'atteignabilité à laquelle nous nous intéressons est:

$\exists\Diamond at-\ell_3$ où $at-\ell_3$ est une proposition atomique qui est vraie uniquement dans le nœud ℓ_3.

Nous pouvons facilement vérifier que la séquence d'exécution temporisée continue suivante qui satisfait cette propriété est une vraie séquence d'exécution temporisée du système.

$\langle \ell_0, (x = 0, t = 0, y = 0), 0\rangle\langle \ell_0, (x = 0.5, t = 0.5, y = 0.5), 0.5\rangle$
$\langle \ell_1, (x = 0, t = 0.5, y = 0.5), 0.5\rangle\langle \ell_1, (x = 0.5, t = 1, y = 0.5), 1\rangle$
$\langle \ell_2, (x = 0.5, t = 1, y = 0.5), 1\rangle\langle \ell_2, (x = 1, t = 1.5, y = 1), 1.5\rangle$
$\langle \ell_3, (x = 1, t = 1.5, y = 1), 1.5\rangle$

Alors qu'il est clair que le système décrit par la figure 5.1 ne possède pas de séquence discrète (C'est donc aussi un exemple de système non fermé par numérisation). En effet si nous notons $\Upsilon(\sigma, \ell_i)$ la fonction qui associe au nœud ℓ_i le temps de séjour dans ce nœud dans la séquence d'exécution temporisée σ nous avons alors :

- La garde $t = 1$ entraîne que $\Upsilon(\sigma, \ell_0) + \Upsilon(\sigma, \ell_1) = 1$.
- La garde $x = 1$ entraîne que $\Upsilon(\sigma, \ell_1) + \Upsilon(\sigma, \ell_2) = 1$.
- La garde $y = 1$ entraîne que $\Upsilon(\sigma, \ell_0) + \Upsilon(\sigma, \ell_2) = 1$.

Or la résolution de ces trois équations donne l'unique solution:

$$\Upsilon(\sigma, \ell_0) = \Upsilon(\sigma, \ell_1) = \Upsilon(\sigma, \ell_2) = 0.5$$

C'est la solution que nous adoptons pour obtenir la séquence continue. Il n'y a pas de solution ayant $\forall j : 0 \leq j \leq 2\ \Upsilon(\sigma, \ell_j) \in I\!N$ le système n'a donc pas de séquence discrète. □

Soit $\mathcal{H}$ un système à pile muni d'intégrateurs de contrôles, nous supposons que ce système est large et simple, nous supposons aussi que ce système possède un seul intégrateur; nous notons alors ce système 1-SIP. Nous rappelons qu'un système est dit large si toutes ses gardes sont des inégalités larges. Nous rappelons aussi qu'un système est simple si toutes ses horloges sont en phases.
Nous prouvons dans ce qui suit qu'étant donnée n'importe quelle configuration temporisée entière α de $\mathcal{H}$, pour toute séquence d'exécution $\sigma \in \mathcal{SET}(\alpha, \mathcal{H})$, il existe une numérisation de σ qui appartient à $\mathcal{H}$.

Proposition 5.4 Soient un 1-SIP $\mathcal{H}$, une configuration temporisée entière α et une séquence d'exécution temporisée $\sigma \in \mathcal{SET}(\alpha, \mathcal{H})$ alors il existe une séquence d'exécution entière $\sigma' \in \mathcal{SET}_{\mathbb{Z}}(\alpha, \mathcal{H})$ et une séquence de numérisation $\vec{\epsilon}$ telle que $[\sigma]_{\vec{\epsilon}} = \sigma'$. ■

Nous commençons par donner l'idée informelle de la preuve.

Nous allons supposer sans pour autant que ce soit une restriction (voir la remarque 5.3), que la séquence d'exécution temporisée $\sigma = \langle \ell_i, \lambda_i, \nu_i, t_i \rangle_{0 \leq i \leq n}$ est complète. Soit $\rho = Trace_T(\sigma)$.

Alors, puisque $\mathcal{H}$ est un système simple, toutes ses horloges ont la même partie fractionnelle le long de la séquence σ, notons aussi que cette partie fractionnelle dans chaque configuration temporisée de σ est égale à la partie fractionnelle du temps global t_i. L'idée de base est de montrer qu'il existe une séquence de numérisation $\vec{\epsilon}$ (c'est à dire un quantum de numérisation ϵ_u pour chaque intervalle $[u, u+1]$) telle qu'à toute position k où une transition gardée est empruntée par σ, la distance entre la valeur (d'origine) de y, et sa valeur obtenue dans la numérisation $[\rho]_{\vec{\epsilon}}$, est strictement plus petite que 1. Nous appelons cette distance *l'erreur* sur y à la position k. En effet, si une telle séquence de numérisation existe et puisque $\mathcal{H}$ est un système large alors la séquence entière (numérisée) $[\rho]_{\vec{\epsilon}}$ respecte toutes les gardes contenant y des transitions empruntées par σ. Puis d'autre part nous montrons que tous les vecteurs de numérisation engendrent pour toute horloge une distance entre la valeur numérisée et la valeur d'origine inférieure à 1 et donc en particulier pour le vecteur construit pour l'intégrateur y, donc la séquence numérisée respecte les gardes contenant les horloges. Par conséquent, elle correspond à une séquence d'exécution de $\mathcal{H}$.
Nous prouvons l'existence de $\vec{\epsilon}$ en montrant que pour tout intervalle de temps $\mathcal{I}_u =$

$[u, u+1]$, s'il est possible de numériser ρ dans tous les intervalles de temps précédents $\mathcal{I}_u$ de sorte que l'erreur sur y au début de l'intervalle $\mathcal{I}_u$ est < 1, il est alors toujours possible de choisir un quantum de numérisation pour l'intervalle courant $\mathcal{I}_u$ de sorte que l'erreur sur y reste < 1 à chaque position dans $\mathcal{I}_u$.

Avant de donner la preuve de la proposition 5.4 nous introduisons quelques notions techniques.

Définition 5.4 Soient une séquence d'exécution temporisée σ et $\vec{\epsilon}$ un vecteur de numérisation, alors pour tout $k \geq 0$ nous définissons

$$\Delta^{\sigma}(k) = t_{k+1} - t_k$$

et

$$\Delta^{\sigma}_{\vec{\epsilon}}(k) = [t_{k+1}]_{\vec{\epsilon}(\lfloor t_{k+1} \rfloor)} - [t_k]_{\vec{\epsilon}(\lfloor t_k \rfloor)}$$

Intuitivement $\Delta^{\sigma}(k)$ est le temps pris par la k^{iemme} transition dans la séquence d'exécution temporisée σ alors que $\Delta^{\sigma}_{\vec{\epsilon}}(k)$ correspond au temps pris par la même transition dans la séquence d'exécution numérisée $[\sigma]_{\vec{\epsilon}}$. □

Nous allons définir la notion d'erreur par numérisation pour toutes les variables de $\mathcal{H}$.

Définition 5.5 Erreur par numérisation
Soient un vecteur de numérisation $\vec{\epsilon}$, une séquence d'exécution continue σ et $k \in \mathbb{N}$, nous définissons pour tout $x \in \mathcal{X}$ l'erreur par numérisation à l'indice k de la variable x que nous notons $E^{\vec{\epsilon}}_k(\sigma, x)$ comme la différence entre sa valeur obtenue suite à la numérisation avec sa valeur d'origine. Donc formellement:

$$E^{\vec{\epsilon}}_k(\sigma, x) = \nu^{\vec{\epsilon}}_k(\sigma, x) - \nu_k(\sigma, x)$$

Avec $\nu_k(\sigma, x)$ la valuation de la variable x dans $(k+1)^{iemme}$ configuration de σ et $\nu^{\vec{\epsilon}}_k(\sigma, x)$ a valuation de la variable x dans $(k+1)^{iemme}$ configuration de la numérisation de σ par $\vec{\epsilon}$. □

Lemme 5.5 Soient $k < n$ et $x \in \mathcal{X}$ alors $|E^{\vec{\epsilon}}_k(\sigma, x)| < 1$ implique que pour tout $u \in \mathbb{N}$,

- $\nu_k(\sigma, x) = u \Rightarrow \nu_k^{\vec{\epsilon}}(\sigma, x) = u$,
- $u < \nu_k(\sigma, x) < u + 1 \Rightarrow \nu_k^{\vec{\epsilon}}(\sigma, x) \in \{u, u + 1\}$.

■

Preuve 5.8 Premièrement considérons le cas $\nu_k(\sigma, x) = u$. Nous avons alors par hypothèse

$$|E_k^{\vec{\epsilon}}(\sigma, x)| = |\nu_k^{\vec{\epsilon}}(\sigma, x) - \nu_k(\sigma, x)| < 1 \tag{5.3}$$

et puisque $\nu_k^{\vec{\epsilon}}(\sigma, x)$ est un entier, alors nous obtenons nécessairement, $\nu_k^{\vec{\epsilon}}(\sigma, x) = u$.

Puis considérons le second cas, $u < \nu^k(\sigma, x) < u+1$. Dans ce cas aussi puisque (5.3) est vraie par hypothèse et puisque $\nu_k^{\vec{\epsilon}}(\sigma, x)$ est un entier, nous avons nécessairement $E_k^{\vec{\epsilon}}(\sigma, x) \neq 0$. Donc si $0 < E_k^{\vec{\epsilon}}(\sigma, x) < 1$, nous avons nécessairement $\nu_k^{\vec{\epsilon}}(\sigma, x) = u + 1$ et si $-1 < E_k^{\vec{\epsilon}}(\sigma, x) < 0$ nous avons $\nu_{\vec{\epsilon}}^k(\sigma, x) = u$. ■

Le lemme 5.5 assure que si σ est une séquence d'exécution temporisée de $\mathcal{H}$ ce qui veut dire en particulier que la garde associée à sa k^{iemme} transition est satisfaite dans ρ. Si $|E_k^{\vec{\epsilon}}(\sigma, x)| < 1$ alors cette même garde est aussi satisfaite dans $[\sigma]_{\vec{\epsilon}}$. Cela est dû au fait que $\mathcal{H}$ est un système large.

Maintenant il faut alors montrer que pour toute séquence d'exécution temporisée σ de $\mathcal{H}$ nous pouvons toujours trouver un vecteur de numérisation $\vec{\epsilon}$ tel que $|E_k^{\vec{\epsilon}}(\sigma, x)| < 1$ pour toute position k dans σ et pour tout $x \in \mathcal{X}$. Nous allons diviser la preuve en deux parties, la première concerne les horloges, la seconde concerne l'intégrateur y. En réalité nous montrons que nous pouvons trouver ce vecteur de numérisation de sorte à ce que l'erreur pour l'intégrateur soit inférieure à 1. Nous montrons au préalable que pour tout vecteur de numérisation et pour toute horloge du système l'erreur pour cette horloge moyennant cette numérisation est inférieure à 1 et plus particulièrement pour le vecteur de numérisation convenant à l'intégrateur.

Lemme 5.6 Pour toute séquence d'exécution temporisée σ, toute horloge $x \in \mathcal{X}$ et pour tout vecteur de numérisation $\vec{\epsilon}$. Nous avons $\forall k \in I\!N \ |E_k^{\vec{\epsilon}}(\sigma, x)| < 1$. ■

Preuve 5.9 Nous avons $E_k^{\vec{\epsilon}}(\sigma, x) = \nu_k^{\vec{\epsilon}}(x) - \nu_k(x)$
donc $E_k^{\vec{\epsilon}}(\sigma, x) = \sum_{i=j_{x_k}}^{k-1} \partial(\ell_i, x).\ \Delta_{\vec{\epsilon}}^{\sigma}(i) - \sum_{i=j_{x_k}}^{k-1} \partial(\ell_i, x).\ \Delta^{\sigma}(i)$ avec j_{x_k} l'indice le plus élevé de la remise à zéro de la variable x (la remise à zéro la plus récente), nous

prendrons $j_{x_k} = 0$ s'il n'y a pas eu de remise à zéro. Puisque x est une horloge donc pour tout $i \in I\!N$ nous avons $\partial(\ell_i, x) = 1$ donc nous obtenons:
$E^{\vec{\epsilon}}_k(\sigma, x) = \sum_{i=j_{x_k}}^{k-1} \Delta^{\sigma}_{\vec{\epsilon}}(i) - \sum_{i=j_{x_k}}^{k-1} \Delta^{\sigma}(i)$
$\Rightarrow E^{\vec{\epsilon}}_k(\sigma, x) = ([t_k]_{\vec{\epsilon}} - [t_{j_{x_k}}]_{\vec{\epsilon}}) - (t_k - t_{j_{x_k}})$
Or toutes les horloges sont en phases donc $t_{j_{x_k}}$ est entier (car j_{x_k} est l'indice de la dernière remise à zéro si elle a lieu sinon c'est l'indice de départ et dans les deux cas la valeur de x est entière donc celle de t aussi). $t_{j_{x_k}}$ est entier donc $[t_{j_{x_k}}]_{\vec{\epsilon}} = t_{j_{x_k}}$. D'où nous obtenons $E^{\vec{\epsilon}}_k(\sigma, x) = ([t_k]_{\vec{\epsilon}} - t_k)$. Donc $|E^{\vec{\epsilon}}_k(\sigma, x)| < 1$. ■

Puis dans le lemme qui va suivre nous énonçons

Lemme 5.7 Soit l'intervalle $\mathcal{I}_u = [u, u+1]$ avec $u \leq \lfloor t_n \rfloor$ et considérons l'ensemble maximal d'indices $J = \{k, k+1, \ldots, k+m\}$ tel que $\forall j \in J$, $t_j \in \mathcal{I}_u$. Alors, pour tout $\vec{\epsilon}$, si $|E^{\vec{\epsilon}}_k(\sigma, y)| < 1$, il existe $\vec{\epsilon'}$ tel que

- $\vec{\epsilon'}(i) = \vec{\epsilon}(i)$ pour tout $i < u$,
- et $\forall j \in J$, $|E^{\vec{\epsilon'}}_j(\sigma, y)| < 1$.

■

Il est à noter que, puisque σ est supposée correspondre à une séquence d'exécution temporisée complète et puisque l'ensemble des indices J est un ensemble *maximal* correspondant aux valeurs du temps dans l'intervalle $\mathcal{I}_u = [u, u+1]$, nous avons nécessairement $t_k = u$ et $t_{k+m} = u+1$ ou $t_{k+m} = t_n$.

Le lemme 5.7 assure qu'étant donnée une séquence de numérisation $\vec{\epsilon}(0) \cdot \vec{\epsilon}(1) \cdots \vec{\epsilon}(u-1)$ qui permet de numériser "*de manière cohérente*" σ jusqu'à la position k (considérant tous les intervalles de temps précédent $\mathcal{I}_u$), il est possible de prolonger cette séquence afin de numériser σ dans l'intervalle $\mathcal{I}_u$ aussi.

En utilisant conjointement le lemme 5.7 et le lemma 5.6, nous prouvons directement qu'il existe une séquence $\vec{\epsilon}$ de valeurs dans $[0, 1[$ telle que la séquence numérisée $[\overrightarrow{\rho}]_{\vec{\epsilon}}$ correspond à une séquence d'exécution temporisée de $\mathcal{H}$.

Preuve 5.10 Pour simplifier l'exposition de la preuve nous supposons que l'intégrateur y n'est pas remis à zéro dans l'intervalle $\mathcal{I}_u$. Tout en sachant que la possible remise à zéro de y ne présente aucune difficulté.

Notons par J^+ le sous-ensemble de $J - \{k+m\}$ tel que pour tout $j \in J^+$, nous avons $\Delta^{\sigma}(j) = t_{j+1} - t_j \neq 0$. Intuitivement, J^+ est l'ensemble des indices de transitions

correspondant à la progression du temps dans l'intervalle $\mathcal{I}_u$, en d'autres termes, les transitions qui (peuvent) modifier les valeurs des variables. Soient $\imath_1$ et $\imath_2$ qui dénotent respectivement le minimum et le maximum de J^+. Il est à noter que nous avons donc $t_{\imath_1} = t_k = u$ et $t_{\imath_2+1} = t_{k+m} = u+1$ ou $t_{\imath_2+1} = t_n$. Nous supposons ici que nous sommes dans le cas de $t_{\imath_2+1} = t_{k+m} = u+1$. Le cas $t_{\imath_2+1} = t_n$ peut être développé de manière similaire.

Maintenant supposons que $|E_k^{\vec{\epsilon}}| < 1$. Nous montrons que nous pouvons choisir une valeur $\epsilon \in [0,1[$ telle que pour toute séquence $\vec{\epsilon'} = \vec{\epsilon}(0)\cdots\vec{\epsilon}(u-1)\cdot\epsilon\cdots$, nous avons pour tout $j \in J$, $|E_j^{\vec{\epsilon'}}| < 1$. Il suffit de considérer les valeurs $E_{j+1}^{\vec{\epsilon'}}$ avec $j \in J^+$, c'est à dire juste après une transition correspondant à la progression du temps, puisque ce sont les seules transitions pouvant modifier les valeurs des variables. Nous rappelons que par définition, nous avons

$$E_{j+1}^{\vec{\epsilon'}}(\sigma,y) = E_k^{\vec{\epsilon}}(\sigma,y) + \sum_{i\in J^+, i\leq j} \partial(\ell_i,y)\cdot(\Delta_{\vec{\epsilon'}}^{\sigma}(i) - \Delta^{\sigma}(i)) \tag{5.4}$$

La preuve se divise en deux cas selon que $E_k^{\vec{\epsilon}}(\sigma,y)$ est positive ou négative.

Nous allons commencer avec le cas $0 \leq E_k^{\vec{\epsilon}}(\sigma,y) < 1$. Dans ce cas il y a trois sous cas à considérer selon que toute les pentes de y dans les nœuds ℓ_j avec $j \in J^+$ sont égales à 1, ou toutes sont égales à 0, ou bien quelques unes d'entre elles sont égales à 1 et les autres sont égales à 0.

1.1. Supposons que pour tout $j \in J^+$, nous avons $\partial(\ell_j,y) = 0$. Nous pouvons alors prendre $\vec{\epsilon'}$ tel que $\vec{\epsilon'}(u)$ prend n'importe quelle valeur dans $[0,1[$. En effet dans ce cas d'après (5.4) nous avons pour tout $j \in J^+$,

$$E_{j+1}^{\vec{\epsilon'}}(\sigma,y) = E_k^{\vec{\epsilon'}}(\sigma,y)$$

1.2. Supposons que pour tout $j \in J^+$ nous avons $\partial(\ell_j,y) = 1$. Nous montrons que nous pouvons prendre n'importe quel $\vec{\epsilon'}$ tel que $\vec{\epsilon'}(u) = \epsilon \in [\mathit{fract}(t_{\imath_2}),1[$.

Puisque les pentes de y sont égales à 1 dans tous les nœuds ℓ_j, par définition de la numérisation, nous avons pour tout $i \in J^+$, $[t_i]_\epsilon = u$. Donc pour tout $i \in J^+$ avec $i \neq \imath_2$, nous avons $\Delta_{\vec{\epsilon'}}^{\sigma}(i) = 0$ et $\Delta_{\vec{\epsilon'}}^{\sigma}(\imath_2) = 1$.

Donc d'après (5.4), nous avons pour tout $j \in J^+$ tel que $j \neq \imath_2$,

$$E_{j+1}^{\vec{\epsilon'}}(\sigma,y) = E_k^{\vec{\epsilon'}}(\sigma,y) - \sum_{i\in J^+, i\leq j} \Delta^{\sigma}(i)$$

De plus, puisque

$$0 < \sum_{i\in J^+, i\leq j} \Delta^{\sigma}(i) = t_{j+1} - t_{\imath_1} < 1$$

et puisque $0 \leq E_k^{\vec{\epsilon}}(\sigma, y) < 1$ par hypothèse, nous avons donc nécessairement

$$-1 < E_{j+1}^{\vec{\epsilon}'}(\sigma, y) < E_k^{\vec{\epsilon}}(\sigma, y) < 1 \tag{5.5}$$

Nous obtenons d'autre part aussi

$$E_{i_2+1}^{\vec{\epsilon}'}(\sigma, y) = E_k^{\vec{\epsilon}'}(\sigma, y) + 1 - \sum_{i \in J^+} \Delta^\sigma(i)$$

donc puisque

$$\sum_{i \in J^+} \Delta^\sigma(i) = t_{i_2+1} - t_{i_1} = 1$$

nous avons

$$E_{i_2+1}^{\vec{\epsilon}'}(\sigma, y) = E_k^{\vec{\epsilon}'}(\sigma, y) \tag{5.6}$$

Donc d'après (5.5) et (5.6), nous obtenons pour tout $j \in J^+$,

$$-1 < E_{j+1}^{\vec{\epsilon}'}(\sigma, y) \leq E_k^{\vec{\epsilon}}(\sigma, y) < 1$$

1.3. Supposons que y a quelques pentes égales à 1 et les autres égales à 0. Donc soit J_1^+ (respectivement J_0^+) l'ensemble des indices $j \in J^+$ tels que $\partial(\ell_j, y) = 1$ (respectivement $\partial(\ell_j, y) = 0$).

Soit p un indice quelconque dans J_0^+. Nous montrons que nous pouvons choisir $\vec{\epsilon}'$ de sorte que $\vec{\epsilon}'(u)$ soit dans $[fract(t_p), fract(t_{p+1})[$ si $t_{p+1} \neq u + 1$, sinon dans $[fract(t_p), 1[$.

En effet d'après (5.4), nous avons pour tout $j \in J^+$,

$$E_{j+1}^{\vec{\epsilon}'}(\sigma, y) = E_k^{\vec{\epsilon}'}(\sigma, y) - \sum_{i \in J_1^+, i \leq j} \Delta^\sigma(i)$$

Il est donc clair que nous avons

$$0 \leq \sum_{i \in J_1^+, i \leq j} \Delta^\sigma(i) < 1$$

et puisque $0 \leq E_k^{\vec{\epsilon}'}(\sigma, y) < 1$, nous en déduisons que pour tout $j \in J^+$,

$$-1 < E_{j+1}^{\vec{\epsilon}'}(\sigma, y) \leq E_k^{\vec{\epsilon}'}(\sigma, y) < 1$$

Ce qui termine le cas $0 \leq E_k^{\vec{\epsilon}}(\sigma, y) < 1$. Nous considérons maintenant le second cas, celui où $-1 < E_k^{\vec{\epsilon}'}(\sigma, y) \leq 0$.

Nous devons dans ce cas aussi considérer les mêmes trois sous cas selon les pentes de y, ces sous cas correspondent à ceux détaillés ci-dessus.

Le premier cas est exactement le même que précédemment (voir le cas 1.1). Nous allons donc considérer les autres cas.

2.1. Supposons que pour tout $j \in J^+$, nous avons $\partial(\ell_j, y) = 1$. Ce cas est parfaitement symétrique au cas (1.2) considéré précédemment: Nous montrons que nous pouvons choisir $\vec{\epsilon}'(u) = \epsilon \in [0, \mathit{fract}(t_{\imath_1+1})[$ si $t_{\imath_1+1} \neq u+1$, sinon nous prenons $\vec{\epsilon}'(u) = \epsilon \in [0, 1[$.

En effet nous avons dans ce cas $[t_{\imath_1}]_\epsilon = u$ et pour tout $i \in J^+$ tel que $i \neq \imath_1$, nous avons $[t_i]_\epsilon = u+1$. Donc d'après (5.4), pour tout $j \in J^+$ nous avons

$$E_{j+1}^{\vec{\epsilon}'}(\sigma, y) = E_k^{\vec{\epsilon}'}(\sigma, y) + 1 - \sum_{i \in J^+, i \leq j} \Delta^\sigma(i)$$

donc puisque

$$0 < \sum_{i \in J^+, i \leq j} \Delta^\sigma(i) \leq 1$$

et de plus $-1 < E_k^{\vec{\epsilon}'}(\sigma, y) \leq 0$ par hypothèse, nous avons nécessairement pour tout $j \in J^+$,

$$-1 < E_k^{\vec{\epsilon}'}(\sigma, y) \leq E_{j+1}^{\vec{\epsilon}'}(\sigma, y) < 1$$

2.2. Supposons dans ce cas que y a quelques pentes égales à 1 et les autres égales à 0. Alors soit J_1^+ et J_0^+ la partition de J^+ définie exactement comme dans le cas (1.3) étudié précédemment. Soit p le minimum de l'ensemble J_1^+. Nous devons alors choisir $\vec{\epsilon}'$ tel que $\vec{\epsilon}'(u)$ est dans l'intervalle $[\mathit{fract}(t_p), \mathit{fract}(t_{p+1})[$ si $t_{p+1} \neq u+1$, sinon dans $[\mathit{fract}(t_p), 1[$.

En effet dans ce cas pour tout $j \in J^+$ tel que $j < p$ (j est nécessairement dans J_0^+ si il existe), nous avons

$$E_{j+1}^{\vec{\epsilon}'}(\sigma, y) = E_k^{\vec{\epsilon}'}(\sigma, y) \tag{5.7}$$

D'autre part pour tout $j \in J^+$ tel que $j \geq p$, nous avons

$$E_{j+1}^{\vec{\epsilon}'}(\sigma, y) = E_k^{\vec{\epsilon}'}(\sigma, y) + 1 - \sum_{i \in J_1^+, i \leq j} \Delta^\sigma(i)$$

donc puisque

$$0 < \sum_{i \in J_1^+, i \leq j} \Delta^{\sigma}(i) < 1$$

et que $-1 < E_k^{\vec{\epsilon}}(\sigma, y) \leq 0$ par hypothèse, nous avons nécessairement pour tout $j \in J^+$ avec $j \geq p$,

$$-1 < E_k^{\vec{\epsilon}}(\sigma, y) < E_{j+1}^{\vec{\epsilon}}(\sigma, y) < 1 \tag{5.8}$$

Par conséquent d'après (5.7) et (5.8), nous obtenons que pour tout $j \in J^+$,

$$-1 < E_k^{\vec{\epsilon}}(\sigma, y) \leq E_{j+1}^{\vec{\epsilon}}(\sigma, y) < 1$$

Ce qui termine le cas $-1 < E_k^{\vec{\epsilon}}(\sigma, y) \leq 0$ et donc la preuve. ■

Preuve 5.11 Preuve de la proposition 5.4
La preuve de cette proposition découle directement de l'utilisation conjointe du lemme 5.5, du lemme 5.6 et du lemme 5.7. En effet, le lemme 5.5 dit que si une séquence d'exécution temporisée σ d'un 1-SIP $\mathcal{H}$ alors toute séquence σ' ayant pour toutes ses variables pour toutes ses configurations une erreur inférieure à 1 en valeur absolue par rapport aux configurations correspondantes de σ alors σ' est une séquence d'exécution temporisée du système $\mathcal{H}$.

Or le lemme 5.7 énonce qu'à partir de σ nous pouvons construire un vecteur de numérisation $\vec{\epsilon}$ de sorte que la séquence obtenue $[\sigma]_{\vec{\epsilon}}$ ait une erreur inférieure à 1 pour l'intégrateur y pour ses configurations et ce par rapport à σ. De plus le lemme 5.6 montre que quelque soit le vecteur de numérisation alors pour la séquence numérisée toutes les horloges du systèmes ont une erreur inférieure à 1 par rapport à la séquence d'origine (σ), en particulier pour le vecteur de numérisation obtenu par la construction donnée dans la preuve du lemme 5.7. La séquence obtenue par numérisation à l'aide de la construction donnée dans la preuve du lemme 5.7 est alors une séquence numérisée qui est une séquence de $\mathcal{H}$.

■

Tenant compte de la proposition 5.1, le problème de la vérification de formules d'atteignabilités larges peut être résolu en ne considérant que les valuations discrètes des variables continues. Donc toutes les variables continues peuvent être vues comme des compteurs de contrôles monotones. Ainsi le problème de la vérification de formules

d'atteignabilités est réduit au problème de la vérification de formules d'atteignabilités dans un automate à pile muni de compteurs de contrôles monotones. Ce problème est décidable (nous l'avons vu dans la partie 5.2.1). Nous obtenons donc le résultat suivant:

Théorème 5.6 Soient $\mathcal{H}$ un 1-SIP, α une configuration entière de $\mathcal{H}$ et φ une formule d'atteignabilité large. Alors le problème de la vérification de $\alpha \models \varphi$ est décidable. □

5.3.4 Le cas général

Juste avant de conclure, nous essayons de regrouper les résultats que nous avons obtenu et que nous avons décrit dans ce chapitre. Nous avons vu que nous pouvons munir les systèmes temporisés à pile de compteurs monotones de contrôles et de compteurs d'observations. Il faut noter que nous pouvons aussi munir ces modèles obtenus par une variable continue, soit un intégrateur de contrôle soit une variable linéaire d'observation.
Nous obtenons donc les deux résultats suivants:

Théorème 5.7 Le problème d'atteignabilité sur un système temporisé large à pile muni de compteurs monotones de contrôles, de compteurs d'observations et d'une variable linéaire d'observation est décidable. □

Preuve 5.12 La preuve de ce théorème se déduit directement des deux faits suivants réunis: le résultat énoncé dans le théorème 5.5 et du fait que la numérisation préserve les séquences d'opérations qui s'effectuent sur les structures de données discrètes. En effet une numérisation entraîne que la séquence d'exécution que nous obtenons possède le même séquencement des opérations sur les structures de données discrètes que le séquencement des ces opérations dans la séquence d'exécution d'origine. ■

Théorème 5.8 Le problème d'atteignabilité sur un système temporisé simple et large à pile muni de compteurs monotones de contrôles, de compteurs d'observations et un intégrateur de contrôle linéaire d'observation est décidable. □

Preuve 5.13 La preuve de ce théorème est basée d'une part comme pour le théorème précédent sur le fait que la numérisation (même non uniforme) préserve le séquences d'opérations qui s'effectuent sur les structures de données discrètes puis d'autre part sur le résultat énoncé dans le théorème 5.6. ■

5.4 Conclusion

Nos investigations se sont portées sur la vérification de propriétés invariance/atteignabilité pour des systèmes hybrides modélisés par un graphe ayant un ensemble fini de nœuds de contrôles munis de structures de données non bornées et de variables continues ayant des pentes variables.

Nous avons montré que ce problème est décidable pour plusieurs cas de ces systèmes. Les résultats de décidabilités que nous présentons sont basés sur deux faits. Le premier est que dans tous les cas que nous considérons, nous pouvons montrer que le problème de la vérification est réductible au problème de la vérification sur une structure discrète. Nous obtenons cette réduction soit en adoptant un partitionnement adéquat de l'espace dense des configurations préservant la satisfaction des propriétés d'atteignabilité, soit en utilisant des principes de discrétisation (numérisation uniforme ou non uniforme) afin de montrer que pour toute séquence d'exécution temporisée du système, il existe une séquence d'exécution temporisée *entière* qui visite les mêmes nœuds de contrôles en utilisant les mêmes transitions.
Puis le second fait, est que le problème de vérification sur une structure discrète que nous obtenons est réduit à un problème d'atteignabilité (avec des contraintes) d'un nœud de contrôle dans un automate à pile. Lequel problème peut être résolu soit comme le problème du langage vide d'un langage hors-contexte soit comme un problème de programmation linéaire dans les entiers dans le cas de l'atteignabilité avec contraintes.
De plus, nous verrons dans le prochain chapitre comment d'autres cas de systèmes hybrides peuvent être réduits au problème du langage vide pour un langage hors-contexte. Nous nous intéresserons aux systèmes munis des variables de contrôle prenant leurs pentes dans l'ensemble $\{-1, 0, 1\}$.

Chapitre 6

Vérification de propriétés ω-régulières sur des graphes à intégrateurs étendus

Dans ce chapitre notre intérêt se porte sur une classe de systèmes hybrides pour laquelle le problème de la vérification est réduit au problème du langage vide pour un langage hors contexte comme c'est le cas pour les classes que nous avons considérées dans le chapitre précédent. De plus dans ce chapitre nous nous intéressons non seulement au problème de la vérification de formules d'atteignabilité mais plus généralement au problème de la vérification des propriétés ω-régulières pour une sous-classe de systèmes hybrides linéaires. Les systèmes que nous considérons sont des systèmes ayant deux variables, l'une d'elles doit être monotone (elle aura pour pente 0 ou 1) alors que l'autre peut avoir pour pente -1, 0, ou 1. L'intérêt de ces systèmes est qu'ils illustrent une classe de systèmes hybrides pour laquelle le problème de la vérification est décidable sans pour autant qu'il n'y est une bisimulation (voir définition 3.1) à *borne finie*. C'est à dire qu'il n'existe pas de bisimulation finie pour toute partie bornée du plan des valuations [Hen95, BR95]. En effet, nous prouverons qu'il n'existe pas d'équivalence sur les valuations qui induit une bisimulation sur les configurations du système que nous considérons.

Nous prouvons pour ces systèmes, que le problème de la vérification de propriétés ω-régulières est décidable. Pour cela, nous montrons que ces systèmes génèrent des ensembles de séquences ω-hors-contextes. Ce fait est établi en utilisant la technique de partitionnement de l'ensemble des configurations de ces systèmes, de sorte que ce partitionnement préserve l'équivalence des traces.

Dans ce chapitre, nous nous intéressons au problème de la vérification de *graphes à intégrateurs étendus* (que nous notons GIE). La particularité des GIE est que leurs

variables peuvent avoir des pentes de -1, 0, ou 1 (nous appellerons ces variables des $\{-1,0,1\}$-variables). Le fait de considérer des pentes négatives permet par exemple de raisonner sur des différences entre des durées.

Les résultats relatifs à la vérification de sous-classes de systèmes hybrides linéaires traitent, dans leur majorité, le cas des propriétés d'invariance. Nous avons vu que le problème de la vérification pour ces propriétés est réduit au (complément du) problème d'atteignabilité dans les classes de systèmes considérées. Nous rappelons que ce problème est indécidable même pour des graphes à intégrateurs munis d'un seul intégrateur, ce résultat a été établi dans [HKPV95]. Dans une première partie dans [Kop95b] il est montré que ce problème est indécidable pour des graphes à intégrateurs ayant un seul intégrateur et quatre horloges; puis dans une seconde partie dans [Kop95a], il est montré que pour des graphes à intégrateurs étendus ayant trois horloges et une variable prenant ses pentes dans $\{-1,0,1\}$ le problème de l'atteignabilité est indécidable. De même qu'il est très facile de montrer que pour des GIE, il suffit de considérer la sous-classe de systèmes ayant une seule horloge et deux variables du type $\{-1,0,1\}$-variable pour simuler toute machine à 2 compteurs [KPSY93].

Dans le travail que nous présentons, nous considérons le cas des GIE ayant uniquement deux variables, un intégrateur et une variable du type $\{-1,0,1\}$-variable; nous notons ces systèmes 1IE-1I-GIE pour signifier graphe à intégrateurs étendus possédant un intégrateur étendu et un intégrateur. Nous allons montrer que le problème de la vérification des propriétés ω-régulières non temporisées (définissable par des ω-automates de Muller) sur de tels systèmes est décidable. Nous considérons des propriétés non temporisées afin de ne pas augmenter la dimension du système que nous considérons pour ne pas plonger dans le cadre de systèmes indécidables. Nous obtenons ce résultat en prouvant que l'ensemble des séquences (traces non temporisées) générées par ces systèmes sont ω-hors-contexte (définissables par des ω-automates à pile). C'est alors que nous utilisons le résultat énonçant que le problème de l'inclusion de langages ω-hors-contextes dans des langages ω-réguliers est décidable [CG77].

Afin de prouver que les 1IE-1I-GIE génèrent des ensembles de séquences ω-hors-contextes, nous adoptant une technique basée sur le partitionnement. Nous définissons un partitionnement *infini* de l'espace dense des valuations des variables ($\mathbb{R}^2$). Ce partitionnement est à *borne finie* c'est à dire que ce partitionnement est *fini* pour tout sous-plan de $\mathbb{R}^2$ *borné*. Puis nous prouvons que commençant à partir d'un nœud de contrôle ayant deux valuations dans une même classe, nous avons alors les mêmes ensembles de séquences d'états qui sont générées. En d'autres termes, la partition que nous considérons est compatible avec l'équivalence de trace. En dépit du fait que cette partition est infinie, elle est codable en utilisant un compteur non borné. Plus exactement cette partition infinie est codable par *une machine à un compteur.* De plus, nous montrons que cette partition n'est pas une bisimulation et ne peut pas être utilisée pour raisonner sur des propriétés arborescentes en général. Plus encore, nous

montrons qu'il existe un 1IE-1I-GIE tel qu'il n'existe pas de bisimulation à *borne finie* sur son graphe de transition.

Le reste de ce chapitre est comme suit: Dans la partie 6.2, nous introduisons un partitionnement des valuations dans $\mathbb{R}^2$ (en dimension 2).
Les parties 6.3 et 6.4 concernent l'existence de propriétés préservant les partitionnement à *borne finie*.
Dans la partie 6.3, nous étudierons le cas simple de deux intégrateurs, puis dans la partie 6.4, nous présentons un cas plus général de 1IE-1I-GIE.
La partie 6.5 est dédiée au problème de la vérification.
Finalement, en guise de conclusion nous donnons un certain nombre de remarques dans la partie 6.6.

6.1 Graphes à intégrateurs étendus

Nous rappelons dans cette partie des modèles de systèmes hybrides, que nous appelons *graphes à intégrateurs étendus* et que nous avons noté GIE. Les GIE sont des cas particuliers de *systèmes hybrides linéaires* introduits dans le chapitre 2 et que nous retrouvons dans [MMP92, ACHH93, NOSY93]. De manière informelle les GIE sont définis comme un système de transition muni de variables réelles qui évoluent de manière continue et linéaire dans chaque nœud de contrôle avec des pentes dans $\{-1, 0, 1\}$. Ces modèles généralisent d'une part les *systèmes temporisés* [ACD93] où toutes les variables évoluent avec une pente 1 et d'autre part les *systèmes à intégrateur* où les variables prennent leurs pentes dans $\{0, 1\}$ [ACHH93, KPSY93]. Nous dénotons par GI (graphe à intégrateurs) tout système hybride linéaire ayant uniquement des variables continues prenant leur pentes d'évolution dans l'ensemble $\{1, 0\}$. Finalement nous notons par n-dim GIE (respectivement n-dim GI) avec $n \geq 1$, un GIE (respectivement un GI) avec exactement n variables.

La sémantique des GIE est donnée par l'ensemble des séquences d'exécutions. Pour la définir nous avons besoin de rappeler les notions de configurations, configurations temporisées et séquences d'exécutions temporisées. Il est à noter que pour la vérification nous considérons la condition $\lim_{i\to\infty} t_i = \infty$ qui assure que le temps peut toujours avancer et peut donc aller au delà de n'importe quelle valeur. Ces séquences d'exécutions temporisées sont alors dites *divergentes* (nous retrouvons cette notion dans la littérature comme étant la notion de *non-Zeno* [AL92]).

Dans ce chapitre comme nous l'avons précédemment évoqué, nous voulons vérifier des propriétés ω-régulières sur des GIE. Mais il ne faut considérer que les séquences d'exécutions temporisées divergentes, c'est à dire telles que $\lim_{i\to\infty} t_i = \infty$. Pour cela nous utilisons les ω-automates de Muller qui permettent d'exprimer cette notion de divergence pour le cas que nous considérons. Nous verrons par la suite dans la

partie 6.5 comment nous exprimons à l'aide des ω-automates de Muller le fait que nous restreignons notre vérification aux séquences d'exécutions divergentes.

6.2 Partitions de bases

Dans ce qui suit nous nous intéressons au cas des 2-dim GIE. Soient $\mathcal{H}$ un tel système, x et y ses deux variables. Nous introduisons ici quelques équivalences de bases sur $[\{x,y\} \to I\!R]$ que nous combinerons dans la prochaîne partie pour définir des équivalences compatibles avec les fragments de ECTL*. Les équivalences de bases sont paramétrisées par un nombre naturel n; qui correspondent à la division du plan $I\!R^2$ par rapport à l'unité $1/2^n$, soit verticalement ou horizontalement, ou bien le long des diagonales positives ou négatives. Tout en notant que nous appelons diagonale positive (respectivement négative) toute droite parallèle à la droite $y = x$ (respectivement à la droite $y = -x$) avec x et y les deux variables de GIE.

Nous commençons par définir la division horizontale. Pour tout $n \in I\!N$, nous définissons une équivalence $\rightarrow_n$ entre les valuations de la manière suivante:
$\forall \nu, \nu' \in [\{x,y\} \to I\!R]$, $\nu \rightarrow_n \nu'$ si $\forall k \in Z\!\!Z$,

- $\nu(y) = k/2^n$ si et seulement si $\nu'(y) = k/2^n$ et
- $k/2^n < \nu(y) < (k+1)/2^n$ si et seulement si $k/2^n < \nu'(y) < (k+1)/2^n$.

La division verticale est définie de manière similaire. Pour cela, nous introduisons, pour tout $n \in I\!N$, une équivalence entre les valuations dénotée par $\uparrow_n$ et laquelle est definie comme la relation $\rightarrow_n$ ci-dessus en considérant la variable x à la place de y.

Puis nous définissons la division le long des diagonales positives. Elle correspond à l'équivalence $\nearrow_n$ qui est définie pour tout $n \in I\!N$ de la manière suivante:
$\forall \nu, \nu' \in [\{x,y\} \to I\!R]$, $\nu \nearrow_n \nu'$ si $\forall k \in Z\!\!Z$,

- $\nu(y) - \nu(x) = k/2^n$ si et seulement si $\nu'(y) - \nu'(x) = k/2^n$ et
- $k/2^n < \nu(y) - \nu(x) < (k+1)/2^n$ si et seulement si $k/2^n < \nu'(y) - \nu'(x) < (k+1)/2^n$.

La division le long des diagonales négatives correspond à une équivalence $\searrow_n$ qui est définie pour tout $n \in I\!N$ comme la relation $\nearrow_n$ ci-dessus mais en remplaçant "$\nu(y) - \nu(x)$" (resp. "$\nu'(y) - \nu'(x)$") par "$\nu(y) + \nu(x)$" (resp. "$\nu'(y) + \nu'(x)$").

Nous définissons d'autres équivalences de bases obtenues à partir de celles que nous avons défini ci-dessus en introduisant une contrainte qui détermine le sous-plan où la division est appliquée. Soit $\rightsquigarrow$ qui dénote soit $\rightarrow$, $\uparrow$, $\nearrow$, ou bien $\searrow$.
Etant donnée $f \in \mathcal{C}_{\{x,y\}}$, nous avons alors $\forall \nu, \nu' \in [\{x,y\} \to I\!R]$, $\nu \rightsquigarrow_n^f \nu'$ si

- $\nu \models f$ si et seulement si $\nu' \models f$ et
- si $\nu \models f$ alors $\nu \rightsquigarrow_n \nu'$.

6.3 Le cas simple: 2-dim graphes à intégrateurs

Nous commençons par considérer le cas relativement simple du 2-dim GI (graphes à intégrateurs à deux dimensions). Nous montrons qu'il existe une équivalence finie $\approx$ sur les valuations induisant sur les configurations de tout 2-dim GI une équivalence qui préserve toutes les formules de ECTL*, c'est à dire telle que les configurations ayant le même nœud de contrôle et dont les valuations sont équivalentes par rapport à $\approx$ satisfont les mêmes formules de ECTL*.

Soient $\mathcal{H}$ un 2-dim GI, x et y ses deux intégrateurs. Nous commençons par introduire quelques notations qui nous servirons par la suite. Notons z pour signifier la variable x ou la variable y. Nous dénotons alors par c_z la plus grande constante à laquelle z est comparée dans les gardes de $\mathcal{H}$ et nous dénotons par f_z la contrainte $0 \leq z \leq c_z$.

L'équivalence $\approx$ est alors définie par $\approx = \rightarrow_0^{f_y} \cap \uparrow_0^{f_x} \cap \nearrow_0^{f_x \wedge f_y}$. Il est à noter que $\approx$ est la même équivalence considérée dans [ACD93] qui permet de vérifier des graphes temporisés.

Nous pouvons voir que pour toutes valuations ν et ν', et toute garde g de $\mathcal{H}$, si $\nu \approx \nu'$ alors $\nu \models g$ si et seulement si $\nu' \models g$. Cela est dû au fait que les variables sont comparées avec des constantes entières. De plus, puisque les variables x et y sont monotones (ne décroissent jamais), toutes leurs valeurs au delà des constantes c_x et c_y satisfont respectivement les mêmes gardes. Nous pouvons alors prouver facilement que $\approx$ induit une bisimulation sur le graphe de transition $\mathcal{G}_\mathcal{H}$.

Lemme 6.1 Pour tous nœuds ℓ et ℓ', toutes valuations ν, ν' et μ, $\nu \approx \nu'$ implique que si $\langle \ell, \nu \rangle \triangleright \langle \ell', \mu \rangle$ alors il existe une valuation μ' telle que $\langle \ell, \nu' \rangle \triangleright \langle \ell', \mu' \rangle$ et $\mu \approx \mu'$. ■

Puisque $\sim$ est la plus grande bisimulation sur $\mathcal{G}_\mathcal{H}$, nous déduisons à partir du lemme 6.1 que si $\nu \approx \nu'$, alors $\langle \ell, \nu \rangle \sim \langle \ell, \nu' \rangle$ pour tout nœud ℓ. Par la proposition 3.1, nous obtenons alors le résultat suivant:

Proposition 6.1 Pour tout nœud ℓ de $\mathcal{H}$, toutes valuations ν, ν' et toute formule φ de ECTL*, $\nu \approx \nu'$ implique que $\langle \ell, \nu \rangle \in [\![\varphi]\!]$ si et seulement si $\langle \ell, \nu' \rangle \in [\![\varphi]\!]$. ■

6.4 Un intégrateur et une variable non monotone

Nous considérons dans cette partie le cas de 2-dim GIE ayant un intégrateur et une $\{-1,0,1\}$-variable (nous appelons ces graphes simple-intégrateur 2-dim GIE). Nous allons montrer en général, qu'il n'existe pas de partition de l'ensemble des valuations ($\mathbb{R}^2$) qui préserve ECTL* et qui soit à *borne finie*, c'est à dire finie pour tout sous-plan borné de $\mathbb{R}^2$.

Notation 6.4.1 *Nous notons par CTL^*_n avec $n \in \mathbb{N}$ le fragment de la logique CTL* qui comporte au plus n imbrications de quantificateurs de chemins.*

Nous montrons ce fait même si nous considérons seulement le fragment CTL^*_2. En effet, d'une part nous exhibons un système où les formules de CTL^*_2 peuvent distinguer entre les configurations avec des valuations arbitraires qui sont des points de $\mathbb{R}^2$. Puis d'autres part, nous prouvons qu'il existe une équivalence à *borne finie* sur les valuations qui préserve toutes les formules de $ECTL^*_1$ et ce pour tout système. Cela permet de considérer toutes les propriétés ω-régulières.

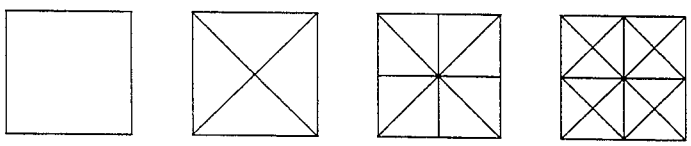
Pour le reste de cette partie nous considérons un 2-dim GIE $\mathcal{H}$ avec deux variables x et y. Nous supposerons que x est l'intégrateur et y étant la $\{-1,0,1\}$-variable.

6.4.1 Une famille de partitions

Nous introduisons une famille décroissante $(\simeq_n)_{n\in\mathbb{N}}$ d'équivalences sur les valuations définies comme suit:

- $\simeq_0 = \mathbb{R}^2$,
- $\forall n \geq 0$, $\simeq_{2n+1} = \simeq_{2n} \cap \rightarrow_n \cap \uparrow_n^{f_x}$,
- $\forall n \geq 1$, $\simeq_{2n} = \simeq_{2n-1} \cap \nearrow_n^{f_x} \cap \searrow_n^{f_x}$.

La figure suivante représente quelques membres de la famille $(\simeq_n)_{n\in\mathbb{N}}$



$\simeq_1$ $\simeq_2$ $\simeq_3$ $\simeq_4$

Il faut noter que chaque équivalence $\simeq_n$ est à *borne finie*. Pour tout $n \in \mathbb{N}$, nous appelons $\simeq_n$-*région bornée* toute classe d'équivalence de $\simeq_n$ qui soit un point ou une

ligne. C'est une extension de la notion de région bornée définie dans la définition 4.4. Il faut noter que les $\simeq_1$-régions bornées sont des classes d'équivalence de $\simeq_2$. Plus encore, nous observons que $\simeq_1$ est compatible avec l'équivalence induite par les gardes de $\mathcal{H}$, c'est à dire que pour toutes valuations ν et ν' $\nu \simeq_1 \nu'$ implique que pour toute garde g de $\mathcal{H}$, $\nu \models g$ si et seulement si $\nu' \models g$. Cela est dû au fait que les variables sont testées à des constantes entières et aussi au fait que toutes les valeurs de x au-delà de c_x satisfont les mêmes gardes car x est une variable monotone. Il est clair que nous ne pouvons borner la partition $\simeq_1$ dans la dimension de y comme nous l'avons fait pour x car y est n'est d'une part pas monotone et d'autre part elle est non bornée.

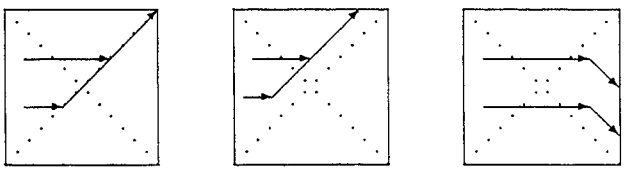
6.4.2 Une partition compatible pour ECTL_1^*

Nous montrons dans ce qui suit que $\simeq_2$ induit une relation d'équivalence sur les configurations qui est incluse dans $\sim_1$ et par conséquent elle préserve toutes les formules de ECTL_1^*.

Pour débuter, en utilisant le fait que $\simeq_1$ est compatible avec l'équivalence induite par les gardes de $\mathcal{H}$ et que $\simeq_1$-régions bornées sont des classes d'équivalence de $\simeq_2$, nous pouvons prouver le lemme suivant:

Lemme 6.2 Pour toutes valuations ν et ν', si $\nu \simeq_2 \nu'$ alors lorsque $\langle \ell, \nu \rangle = \langle \ell_0, \nu_0 \rangle \rhd \cdots \langle \ell_m, \nu_m \rangle \rhd \langle \ell', \mu \rangle$ avec $\forall i \in \{0, \cdots, m\}$. $\nu \simeq_1 \nu_i$ et μ est dans une $\simeq_1$-région bornée, nécessairement $\langle \ell, \nu' \rangle = \langle \ell_0, \nu'_0 \rangle \rhd \cdots \langle \ell_m, \nu'_m \rangle \rhd \langle \ell', \mu' \rangle$ où $\forall i \in \{0, \cdots, m\}$. $\nu'_0 \simeq_1 \nu'_i$ et $\mu \simeq_2 \mu'$. ■

Les figures suivantes illustrent le lemme 6.2 sur quelques exemples typiques.



Il est à noter que le Lemme 6.2 n'est pas vrai lorsque nous considérons la configuration cible $\langle \ell', \mu \rangle$ où μ n'est pas nécessairement dans une $\simeq_1$-région bornée.

Maintenant, soient $\alpha = \langle \ell, \nu \rangle$ et $\alpha' = \langle \ell, \nu' \rangle$ deux configurations telles que $\nu \simeq_2 \nu'$. Puis considérons une séquence d'exécution $\rho = (\langle \ell_i, \nu_i \rangle)_{i \in I\!N}$ commençant à α. Supposons sans pour autant que ce soit une restriction que ρ est telle que quelques soient les deux configurations $\langle \ell_i, \nu_i \rangle$ et $\langle \ell_j, \nu_j \rangle$ avec $j > i$, telles que ν_i et ν_j sont dans deux $\simeq_1$-régions non bornées différentes, il existe une configuration $\langle \ell_k, \nu_k \rangle$ avec $i < k < j$,

telle que ν_k est dans une $\simeq_1$-région bornée. Ce n'est pas une restriction car nous pouvons toujours compléter la séquence ρ de sorte que nous ayons la configuration $\langle \ell_k, \nu_k \rangle$. Nous décrivons dans ce qui suit les différents types de séquences d'exécutions divergentes que nous pouvons avoir.

- Par le fait que le temps croît au-delà de tout entier dans une séquence d'exécution (elles sont générées par des séquences d'exécutions temporisées à temps divergent) alors une variable ayant une pente 1 ou -1 sera inévitablement égale à une valeur entière (à moins qu'elle ne soit remise à zéro et dans ce cas elle devient explicitement entière), nous pouvons voir que nécessairement nous avons, soit une séquence infinie d'indices $(i_j)_{j \in \mathbb{N}}$ telle que $\forall j \in \mathbb{N}$. ν_{i_j} est dans une $\simeq_1$-région bornée en appliquant le lemme 6.2 nous pouvons voir que nous pouvons construire facilement une séquence d'exécution commençant à $\alpha' = \langle \ell, \nu' \rangle$ et passant par les mêmes nœuds que la séquence que nous avons considéré.

- Un autre type, est qu'on ait une séquence finie d'indices $(i_j)_{j=0}^{m}$, telle que $\forall j.\ 0 \leq j \leq m-1$. ν_{i_j} est dans une $\simeq_1$-région bornée, $\nu_{i_m}(x) > c_x$, $fract(\nu_{i_m}(y)) \neq 0$ et $\forall i > i_{m-1}$. $\nu_i \simeq_1 \nu_{i_m}$, c'est le cas où l'horloge x dépasse c_x et la variable y évolue sans atteindre d'entier. Cela veux dire qu'il existe $u \in \mathbb{Z}$ telle que $\forall i > i_{m-1}\ u < \nu_i(y)$. Nous pouvons aussi dans ce cas construire facilement une séquence d'exécution commençant à $\alpha' = \langle \ell, \nu' \rangle$ et passant par les mêmes nœuds que la séquence que nous avons considérée. En effet il suffit de prendre des variations plus fines dans la séquence que nous construisons de sorte à garantir que $\forall i > i_{m-1}\ u < \nu_i(y)$ et $\nu_i(x) > c_x$, il est simple de voir que ce choix est possible.

- Il y a aussi le type de séquences qui à partir d'un moment restent dans une même région bornée (avec x ayant pour pente 0 et y prend alternativement les pentes 1 et -1). Dans ce cas nous agissons comme dans le cas précédent; en effet nous pouvons construire une séquence d'exécution commençant à $\alpha' = \langle \ell, \nu' \rangle$ et passant par les mêmes nœuds que la séquence que nous avons considérée, il suffit de prendre des variations plus fines dans la séquence que nous construisons de sorte à garantir que nous restons dans la même région bornée.

En utilisant le lemme 6.2 et le fait que l'équivalence $\simeq_1$ sur les valuations satisfait les mêmes gardes, nous pouvons prouver que dans les cas que nous présentons, il existe une séquence d'exécution commençant à α' qui visite exactement les mêmes nœuds, c'est à dire que nous avons le résultat suivant:

Lemme 6.3 Pour tout nœud ℓ, toutes valuations ν et ν', $\nu \simeq_2 \nu'$ implique que $\forall (\langle \ell_i, \nu_i \rangle)_{i \in \mathbb{N}} \in \mathcal{SE}(\langle \ell, \nu \rangle, \mathcal{H}).\ \exists (\langle \ell_i, \nu'_i \rangle)_{i \in \mathbb{N}} \in \mathcal{SE}(\langle \ell, \nu' \rangle, \mathcal{H})$. ∎

Il est à noter que pour toutes valuations ν, ν' et pour tout nœud contrôle ℓ, nous avons trivialement $\langle \ell, \nu \rangle \sim_0 \langle \ell, \nu' \rangle$. En utilisant le lemme 6.3, nous déduisons que pour toutes valuations ν et ν', $\nu \simeq_2 \nu'$ implique que $\langle \ell, \nu \rangle \sim_1 \langle \ell, \nu' \rangle$ pour tout nœud de contrôle ℓ. En d'autres termes, $\simeq_2$ est inclus dans une équivalence de trace, c'est à dire $\nu \simeq_2 \nu'$ implique que $\mathcal{T}(\langle \ell, \nu \rangle, \mathcal{H}) = \mathcal{T}(\langle \ell, \nu' \rangle, \mathcal{H})$. Par conséquent, par la proposition 3.2, nous déduisons:

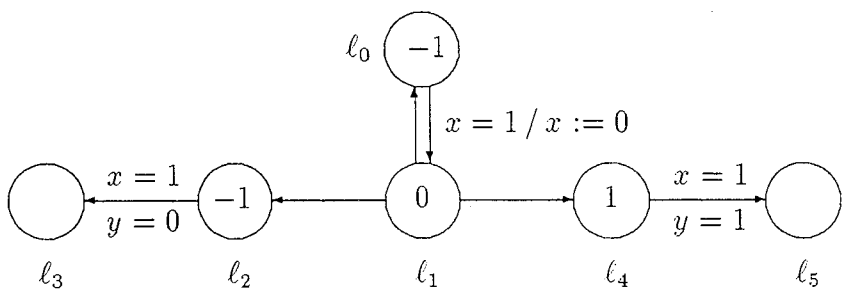
Proposition 6.4.1 *Soit $\mathcal{H}$ un 2-dim GIE où l'une des deux variables est un intégrateur et l'autre est un intégrateur étendu ($\{-1, 0, 1\}$ − variable). Alors pour tout nœud ℓ, toutes valuations ν, ν' et toute formule φ de $ECTL_1^*$, $\nu \simeq_2 \nu'$ implique que $\langle \ell, \nu \rangle \in [\![\varphi]\!]$ si et seulement si $\langle \ell, \nu' \rangle \in [\![\varphi]\!]$.*

6.4.3 Incompatibilité avec CTL_2^*

Nous pouvons observer que, pour tout $n \in \mathbb{N}$, il existe un système où l'équivalence $\simeq_n$ n'induit pas de bisimulation sur les configurations du système. Par exemple, nous pouvons voir sur les figures illustrant le lemme 6.2 qu'à partir de deux points $\simeq_2$-équivalents nous pouvons atteindre différentes régions $\simeq_2$-bornées en empruntant des nœuds de contrôles où y a une pente de 0. Par conséquent, aucune équivalence $\simeq_n$ ne suffit pour la vérification de propriétés générales de branchement.

Nous prouvons qu'il n'existe pas de partitionnement à *borne finie* de l'ensemble des valuations qui induit une bisimulation sur les configurations de tous systèmes. En effet, nous montrons qu'il existe un système $\mathcal{S}$ et une famille de formules $(\varphi_n)_{n \geq 2}$ de CTL_2^* telle que pour tout $n \geq 2$, il existe des configurations $\mathcal{S}$ ayant des valuations $\simeq_n$-équivalentes qui sont distinguées par la formule φ_n.

Le système $\mathcal{S}$ est représenté dans la figure ci-dessous. Dans le système la variable x est une horloge, nous pouvons donc ommètre de spécifier sa pente à chaque nœud de contrôle (elle est toujours égale à 1). Par contre nous donnons les pentes de la variable y (lorsque cette pente est importante, c'est à dire lorsque la valeur de cette pente a une importance dans le raisonnement que nous effectuons).



Maintenant définissons la famille des formules $(\varphi_n)_{n\geq 2}$. Mais au préalable nous donnons la sémantique de l'opérateur $\mathcal{U}$ que nous utilisons. Nous avons $\alpha \models \exists(\phi_1 \mathcal{U} \phi_2)$ si et seulement si $\exists(\langle \ell_i, \nu_i \rangle)_{i\in I\!N} \in \mathcal{SE}(\alpha, \mathcal{H})$ telle que $\exists j \in I\!N : \langle \ell_j, \nu_j \rangle \models \phi_2$ et $\forall u : 0 \leq u \leq j \ \langle \ell_u, \nu_u \rangle \models \phi_1$.
Nous commençons par introduire les formules de CTL_1^* $\phi_0 = \exists((\text{at-}\ell_1 \vee \text{at-}\ell_2)\mathcal{U}\,\text{at-}\ell_3)$ et $\phi_1 = \exists((\text{at-}\ell_1 \vee \text{at-}\ell_4)\mathcal{U}\,\text{at-}\ell_5)$. Puis pour tout $n \geq 2$, nous définissons les formules $\varphi_n = \exists\psi_n$ de CTL_2^* où

- $\psi_2 = \phi_0 \mathcal{U}(\text{at-}\ell_4 \wedge \phi_1)$,
- $\forall k \geq 1.\ \psi_{2k+1} = \text{at-}\ell_0 \mathcal{U}(\text{at-}\ell_1 \wedge \psi_{2k})$,
- $\forall k \geq 2.\ \psi_{2k} = \text{at-}\ell_1 \mathcal{U}(\text{at-}\ell_1 \wedge \neg\phi_1 \wedge (\text{at-}\ell_1 \mathcal{U} \text{at-}\ell_0 \wedge \psi_{2k-1}))$.

Nous montrons comment la formule φ_2 distingue entre les configurations à valuations $\simeq_2$-équivalentes. Considérons les formules ϕ_0 et ϕ_1 qui apparaissent dans φ_2. La formule ϕ_0 (respectivement ϕ_1) est satisfaite par les configurations à partir desquelles la configuration $\langle \ell_3, (1,0) \rangle$ (respectivement $\langle \ell_5, (1,1) \rangle$) est atteignable sans visiter ℓ_0. ϕ_0 (respectivement ϕ_1) est alors satisfaite par précisément les configurations qui sont soit au nœud ℓ_1 et satisfont $y + x \leq 1$ (respectivement $y - x \geq 0$) ou bien au nœud ℓ_2 (respectivement ℓ_4) et satisfaisant $y + x = 1$ (respectivement $y - x = 0$).

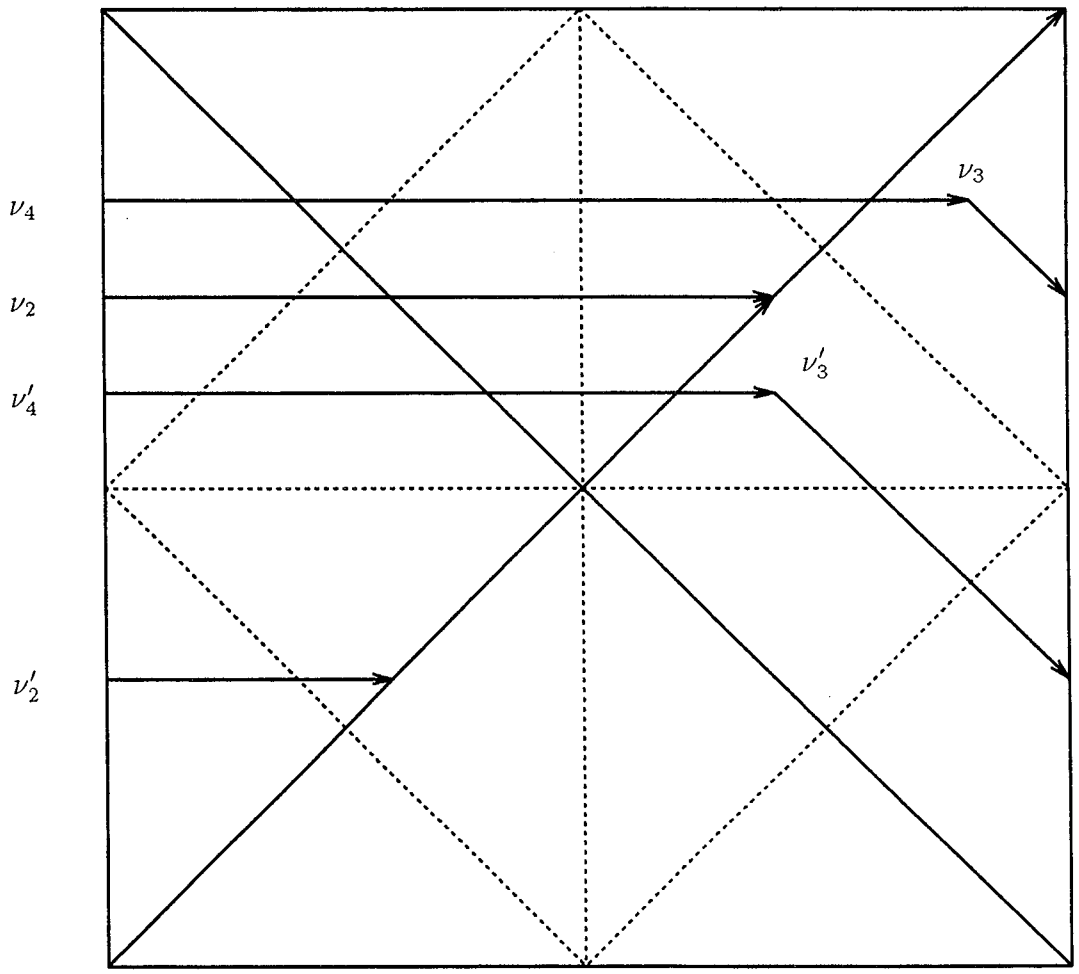
La formule φ_2 correspond aux configurations qui peuvent atteindre une configuration ayant pour nœud de contrôle ℓ_4 satisfaisant ϕ_1 (donc satisfaisant $y - x = 0$) et toutes les configurations intermédiaires satisfont ϕ_0 (ces configurations intermédiaires satisfont donc $y + x \leq 1$).

Considérons les valuations $\nu_2 = (0, 0.7)$ et $\nu_2' = (0, 0.3)$. Il est clair que $\nu_2 \simeq_2 \nu_2'$. Quoi qu'il en soit, nous pouvons voir que $\langle \ell_1, \nu_2 \rangle$ ne satisfait pas φ_2 car afin d'atteindre une configuration ayant pour nœud de contrôle ℓ_4 qui satisfait ϕ_1, nous devons attendre dans le nœud ℓ_1 jusqu'au point $(0.7, 0.7)$ qui ne satisfait pas ϕ_0. D'autre part il est facile de voir que $\langle \ell_1, \nu_2' \rangle$ satisfait φ_2.

Considérons la formule φ_3. Cette formule est satisfaite par les configurations ayant pour nœud ℓ_0 à partir desquelles il est possible d'atteindre ℓ_1 avec une configuration qui satisfait φ_2. Il est à noter que dans la définition de φ_3 ci-dessus, nous utilisons la formule de chemin ψ_2 à la place de la formule d'état φ_2; nous pouvons voir que cela donne une formule équivalente qui est dans CTL_2^* alors que l'utilisation de φ_2 donne une formule de CTL_3^*. Considérons les valuations $\nu_3 = (0.9, 0.8)$ et $\nu_3' = (0.7, 0.6)$. Ces valuations sont $\simeq_3$-équivalentes, cependant nous pouvons voir que $\langle \ell_0, \nu_3 \rangle$ ne satisfait pas φ_3 alors que $\langle \ell_0, \nu_3' \rangle$ la satisfait.

Finalement considérons la formule φ_4. Elle est satisfaite par les configurations qui sont dans ℓ_1 et Restant dans ℓ_1 jusqu'à atteindre un point où ϕ_1 est fausse, c'est à

dire où $x > y$; ces configurations peuvent atteindre une configuration qui satisfait φ_3. Nous considérons alors les valuations $\nu_4 = (0, 0.8)$ et $\nu'_4 = (0, 0.6)$. Nous pouvons aussi voir que ces valuations sont $\simeq_4$-équivalentes mais les configurations $\langle \ell_1, \nu_4 \rangle$ et $\langle \ell_1, \nu'_4 \rangle$ sont distinguées par φ_4. Il est facile de voir que nous pouvons répéter ce procédé indéfiniment. La figure suivante illustre la discussion que nous venons de donner ci-dessus.



Proposition 6.2 $\forall n \geq 2$, il existe un nœud de contrôle ℓ dans $\mathcal{S}$ (ℓ_0 ou ℓ_1) et deux valuations ν_n et ν'_n telles que $\nu_n \simeq_n \nu'_n$, $\langle \ell, \nu_n \rangle \notin [\![\varphi_n]\!]$, et $\langle \ell, \nu'_n \rangle \in [\![\varphi_n]\!]$. ■

Nous pouvons voir que pour tous deux points dans $[0, 1]^2$, il existe $n \in \mathbb{N}$ tel que ces points sont distinguables par $\simeq_n$. Nous obtenons donc comme conséquence de la proposition 6.2 et de la proposition 3.1 qu'il n'existe pas d'équivalence sur les valuations qui soit à *borne finie* et qui induit une bisimulation sur les configurations

de $\mathcal{S}$. En effet car s'il exite une équivalence à borne finie $\cong$ alors $\cong$ serait finie sur $[0,1]^2$ donc il existe $k \in I\!N$ tel que d'une part $\cong$ est égale à l'intersection de $\simeq_k$ et d'une autre équivalence finie $\cong'$, puis d'autre part $\cong$ est différente de l'intersection entre $\simeq_{k+1}$ et $\cong'$. A partir de ce rang $(k+1)$ il existe une formule φ_{k+1} qui distingue entre certains points d'une même classe obtenue par l'équivalence $\cong$. Donc $\cong$ n'induit pas de bisimulation sur les configurations de $\mathcal{S}$. Il n'existe donc pas d'équivalence à borne finie.

Théorème 6.1 Il existe un système 2-dim GEI $\mathcal{S}$ ayant une seule horloge tel que pour toute équivalence à *borne finie* $\cong$ sur les valuations, il existe un nœud ℓ de contrôle dans $\mathcal{S}$ et deux valuations ν et ν', tel que $\nu \cong \nu'$ et $\langle \ell, \nu \rangle \not\sim \langle \ell, \nu' \rangle$. □

6.5 Le problème de vérification

Dans cette partie, nous présentons des résultats de décidabilités pour le problème de la vérification de systèmes que nous avons considérés dans les parties précédentes. Nous prouvons principalement que le problème de la vérification de propriétés ω-régulières (formules de ECTL^*_1) pour les sytèmes 2-dim GIE ayant un seul intégrateur est décidable. Pour cela nous montrons que ce problème peut être réduit au problème d'inclusion des langages ω-hors-contextes dans un langage ω-régulier.

Commençons par le cas simple d'un 2-dim GI (nous rappelons que c'est un graphe à intégrateur à deux variables et que ses deux variables sont des intégrateurs). Nous montrons dans ce cas que le problème de la vérification de formules de ECTL* est décidable. En effet, en utilisant la proposition 6.1, la vérification de formules de ECTL* pour de tels systèmes $\mathcal{H}$ peut être effectuée en considérant le graphe de transition *discret* $\mathcal{R}_{\mathcal{H}}$ obtenu comme le quotient du graphe de transition *dense* $\mathcal{G}_{\mathcal{H}}$ par rapport à l'équivalence sur les configurations induites par $\approx$ (voir la définition dans la partie 6.3). Plus encore, puisque l'équivalence $\approx$ est finie alors le graphe $\mathcal{R}_{\mathcal{H}}$ est fini, il correspond exactement au *graphe des régions* qui est utilisé dans [ACD93] pour la vérification de graphes temporisés. Par contre, tous les chemins dans $\mathcal{R}_{\mathcal{H}}$ ne correspondent pas nécessairement à une séquence d'exécution puisque les seules séquences d'exécutions que nous considérons doivent être générées par des séquences d'exécutions temporisées ayant un temps divergent. Pour cela nous définissons une contrainte de chemin (propriété ω-régulière) $nonZeno$ qui est satisfaite par un chemin de $\mathcal{R}_{\mathcal{H}}$ si et seulement si il correspond à une séquence d'exécution. Nous détaillerons plus tard la définition dans un cadre plus général que celui des systèmes 2-dim GIE à un seul intégrateur. Etant donnée une formule φ de ECTL*, nous considérons la formule φ_{nz} qui est obtenue à partir de φ en remplaçant récursivement chaque sous formule de la forme $\exists\Omega$ par la formule $\exists(nonZeno \cap \Omega)$. Donc une configuration $\langle \ell, \nu \rangle$ de $\mathcal{H}$ satisfait la formule φ si

et seulement si le nœud $\langle \ell, [\nu]_{\approx} \rangle$ de $\mathcal{R}_{\mathcal{H}}$, où $[\nu]_{\approx}$ est la classe d'équivalence $\approx$ de ν, satisfaisant les formules φ_{nz}. Puisque le problème de la vérification de ECTL* pour un système d'état fini est décidable. Nous en déduisons le fait suivant:

Proposition 6.3 Le problème de la vérification de systèmes 2-dim GI par rapport à des formules de ECTL* est décidable. ■

Considérons le cas des systèmes 2-dim GIE à un seul intégrateur et soit $\mathcal{H}$ un tel système. Comme dans le cas précédent, par la proposition 6.4.1, le raisonnement sur les propriétés ω-régulières (propriétés de ECTL_1^*) des systèmes 2-dim GIE à un seul intégrateur peut être réalisé en considérant le quotient du graphe de transition par rapport à l'équivalence induite par $\simeq_2$. Par contre, contrairement au cas du système 2-dim GI, ce quotient est *infini* car $\simeq_2$ a un nombre infini de classes. Néanmoins, le graphe peut être codé par un automate à 1 compteur $\mathcal{A}_{\mathcal{H}}$, le compteur représente la partie entière de la $\{-1, 0, 1\}$-variable y (c'est à dire $\lfloor y \rfloor$) et l'ensemble fini de nœuds de contrôles correspond aux informations suivantes: un nœud de contrôle de $\mathcal{H}$, est $fract(y) = 0$ ou non, $x > c_x$ ou non; dans le cas où $x \leq c_x$ il faut connaître l'intervalle $[n, n]$ où $]n, n+1[$ (aux bornes entières) qui contient x, puis finalement la comparaison de $fract(x) - fract(y)$ et $fract(x) + fract(y)$ respectivement par rapport à 0 et à 1.

Nous rappelons que dans notre vérification il ne faut considérer que les séquences d'exécutions temporisées divergentes, c'est à dire telles que $\lim_{i \to \infty} t_i = \infty$. Pour cela nous utilisons les ω-automates de Muller qui permettent d'exprimer cette notion de divergence pour le cas que nous considérons. Nous verrons dans la partie 6.5 comment nous exprimons grace à des ω-automates de Muller le fait que nous restreignons notre vérification aux séquences d'exécutions divergentes. Il faut donc garantir que seuls les chemins correspondant au temps divergent sont considérés. Pour cela, nous introduisons une condition d'acceptance de Muller 3.2.1 que nous décrivons ci-dessous.

Pour commencer, nous considérons des nouvelles propositions atomiques afin de tenir compte dans la définition des nœuds de contrôles, des informations supplémentaires disant si un nœud correspond à une région $\simeq_1$-bornée et qu'il est nœud cible par une transition de progression du temps à partir d'un nœud qui correspond à une région non $\simeq_1$-bornée ou non. Les nœuds satisfaisant cette condition permettent d'observer la progression du temps. Donc un chemin qui visite infiniment souvent un tel nœud doit être accepté et par conséquent, tout ensemble contenant un tel nœud est considéré comme un ensemble de répétition (condition des automates de Muller).

Il existe d'autres sortes de chemins qui correspondent à des séquences d'exécutions temporisées à temps divergent. En effet, il existe les séquences à temps divergent qui ne rencontrent pas infiniment souvent des régions $\simeq_1$-bornées en utilisant des transitions associées à la progression du temps.

La première sorte de chemins que nous devons accepter sont ceux qui correspondent aux séquences d'exécutions qui, au-delà d'un certain point, restent à jamais dans une classe d'équivalence non bornée $\simeq_2$. C'est à dire où $x > c_x$ et y reste dans un intervalle ouvert $]n, n+1[$. Pour cela nous considérons comme ensemble de répétition tout ensemble de nœuds constituant une composante fortement connexe dans le graphe de transition de l'automate $\mathcal{A}_\mathcal{H}$ tel que tous ces nœuds correspondent à une même classe de ce type non bornée et soit qu'il existe un nœud correspondant à un nœud ℓ avec $\partial(\ell, y) = 0$ ou bien qu'il existe deux nœuds correspondant aux nœuds ℓ_1 et ℓ_2 avec $\partial(\ell_1, y) = 1$ et $\partial(\ell_2, y) = -1$. Dans les deux cas, nous pouvons en effet construire une séquence d'exécution à temps divergent où à partir d'un point, la valeur de y est soit toujours la même (en utilisant un nœud où la pente est 0), ou oscille entre deux valeurs fixées qui soient assez petites pour permettre de rester dans une même classe (cela est possible puisque la classe considérée est ouverte).

Les autres sortes de chemins que nous devons accepter sont ceux qui correspondent par exemple à des exécutions qui, au-delà d'un point vont d'une configuration $\langle \ell, \nu \rangle$ telle que $\nu(x) = 0$ vers une configuration $\langle \ell', \nu' \rangle$ où $\nu(y) = \nu'(y)$ sans visiter aucune région $\simeq_1$-bornée, remet alors à zéro la variable x puis retourne en arrière à la configuration $\langle \ell, \nu \rangle$. Nous considérons alors comme ensemble de répétition tout ensemble de nœud fortement connexe N dans le graphe de transition $\mathcal{A}_\mathcal{H}$ tel qu'il existe une transition entre deux nœuds dans N qui remet à zéro x. Soit qu'il existe un nœud dans N correspondant à un nœud ℓ ayant $\partial(\ell, y) = 0$, soit qu'il existe deux nœuds dans N correspondant aux nœuds ℓ_1 et ℓ_2 ayant $\partial(\ell_1, y) = 1$ et $\partial(\ell_2, y) = -1$. Nous considérons aussi le cas où y est remise à zéro et qu'il existe un nœud dans N ayant $\partial(\ell, x) = 0$. Nous prouvons que dans de tels cas, nous pouvons construire une séquence d'exécution à temps divergent qui prend dans chaque cycle, une quantité de temps fixée ϵ.

Il est à noter que la condition de Muller définie ci-dessus peut être codée en utilisant des propositions atomiques supplémentaires (une pour chaque ensemble de répétition) comme une propriété ω-régulière qui correspond à la propriété de *nonZeno* mentionnée ci-dessus dans le cas des 2-dim GI.

Une configuration $\langle \ell, \nu \rangle$ satisfait alors une propriété ω-régulière Ω, c'est à dire $\langle \ell, \nu \rangle \in [\![\forall\Omega]\!]$ si et seulement si le langage ω-hors-contexte reconnu par $\mathcal{A}_\mathcal{H}$ commençant à partir de $\langle \ell, [\nu]_{\simeq_2} \rangle$ est inclus dans Ω. Puisque ce problème d'inclusion entre langages ω-hors-contextes et ω-réguliers est décidable, nous en déduisons que le problème de la vérification de propriétés ω-régulières pour des systèmes 2-dim GIE à un seul intégrateur est décidable. Donc, par le fait que les formules de ECTL_1^* sont des combinaisons booléennes de formules de la forme $\forall\Omega$, nous obtenons le résultat de décidabilité suivant:

Théorème 6.2 Le problème de la vérification de systèmes 2-dim GIE à un seul inté-

grateur par rapport à des formules de ECTL_1^* est décidable. □

6.6 Conclusion

Nous avons présenté dans ce chapitre des résultats de décidabilité pour une sous-classe de systèmes hybrides linéaires. Ces systèmes sont les 2-dim GIE (graphes à intégrateurs étendus ayant deux variables) avec un intégrateur et une $\{-1, 0, 1\}$-variable. Nous avons établi ce résultat en montrant que ces systèmes génèrent des exemples de séquences d'états ω-hors-contextes.

Nos travaux dans ce chapitre montrent qu'il existe des systèmes hybrides dont le problème de la vérification est décidable, mais qui ne peut pas être réduit au problème de la vérification sur les systèmes d'états finis comme c'est habituellement le cas. Ce résultat établit un lien très important entre les systèmes à espace d'états infini et la vérification de systèmes hybrides.

Plus encore, ces travaux montrent qu'il existe des systèmes hybrides sur lesquels des propriétés linéaires peuvent être vérifiées alors qu'il n'existe pas de bisimulation à *borne finie* sur ses ensembles de configurations. Entre ces systèmes que nous considérons et les automates à pile il existe toujours une relation d'équivalence de traces, mais ils ne sont pas nécessairement bisimilaires. Par conséquent ces systèmes ne peuvent être vérifiés en utilisant une bisimulation finie basée sur l'approche décrite dans[Hen95].

Chapitre 7

Du calcul de durées vers les automates linéaires hybrides

Dans ce chapitre nous étudions les relations entre deux approches de spécification et de vérification de systèmes hybrides. Nous avons étudié la première dans ce manuscrit, elle est basée sur les automates hybrides. La seconde est basée sur la logique de durées [CHR91] (que nous dénotons CoD pour Calculus of Duration). Pour cette logique de durée, certains fragments ont été identifiés dans [HHF+94, Rav94] pour la description de systèmes de contrôle et de leurs exigences. Nous montrerons que pour ces fragments de cette logique de durée, le problème de la vérification "d'un système de contrôle décrit en CoD par rapport à des exigences décrites en CoD" est décidable. Pour montrer cette décidabilité nous montrons que nous pouvons réduire ce problème au problème d'atteignabilité pour une sous-classe d'automates hybrides linéaires pour lesquels ce même problème est décidable.

L'approche basée sur la logique des durées (CoD) qui a été introduite dans [CHR91] est une extension de la logique temporelle d'intervalles introduite dans [Mos85], cette extension est basée sur le concept de durées. La durée d'une propriété d'état dans un intervalle d'exécution est l'accumulation du temps durant lequel la propriété d'état considérée est vraie. Dans les principaux travaux concernant la logique CoD, aussi bien les systèmes de contrôle que leurs exigences sont décrits par des formules de cette logique. Récemment des formes spéciales de formules de CoD ont été identifiées pour spécifier les systèmes de contrôles et leurs exigences [HHF+94, Rav94]. Dans ce fragment de la logique CoD, le fait qu'un système de contrôle spécifique satisfait un ensemble d'exigence est exprimé à l'aide d'implications logiques. La vérification de telles relations d'implications peut être prouvée en utilisant un système déductif pour la validité des formules de CoD [CHR91, RRH93, Rav94]. La plupart des résultats de décidabilité existant pour la logique CoD en temps dense sont négatifs. Le seul résultat positif connu concerne le fragment propositionnel de la logique CoD sans contraintes

de temps ni de durées [CHS93]. Notre apport dans ce chapitre va aussi dans ce sens, puisque nous exhibons un fragment plus important qui est décidable.

Puis pour ce qui est de l'approche basée sur les automates hybrides, cette dernière a été intensivement étudiée récemment. Plus précisément plusieurs groupes de recherches [ACD93, ACH93, KPSY93, PV94, BER94b, BR95] ont porté leurs intérêts sur le problème d'atteignabilité dans les automates hybrides linéaires, leurs travaux ont donné lieu à l'émergence de plusieurs cas spéciaux où ce problème est décidable. Nous avons étudié certains cas dans les deux chapitres précédents. Le cas général est indécidable [ACHH93, KPSY93, HK94, Kop95b, Kop95a]. Pour le cas général des méthodes basées sur les techniques d'approximations ont été proposées [Hal93, ACH+95].

Dans ce chapitre nous exhibons un résultat reliant les deux approches CoD et automates hybrides. En effet nous prouvons que le problème de la vérification de si un système de contrôle décrit en CoD satisfait ou non des exigences décrites en CoD peut être réduit au problème d'atteignabilité pour une classe d'automates à intégrateurs pour lesquels ce problème est décidable. Grâce à cela nous obtenons une méthode algorithmique permettant de prouver la relation de satisfaction entre systèmes de contrôles et les exigences exprimés dans la logique CoD. Les résultats que nous décrivons concernent une classe permettant de représenter des systèmes de contrôle et des exigences se rapportant à ces systèmes, qui est plus général que le fragment de CoD qui permet de représenter les systèmes de contrôle et les exigences. En effet, nous commençons par introduire un langage de spécification appelé DIL (Duration Interval Logic), qui servira comme langage intermédiaire entre le CoD et les automates hybrides. DIL correspond au fragment propositionnel du CoD étendu avec des variables de durées introduites via des quantifications de remise à zéro qui associent à chaque variable x une formule d'état dont la durée est mesurée par x; cette variable est remise à zéro lors de la quantification. Ces variables de durées sont introduites pour nous permettre d'exprimer des contraintes linéaires sur les durées. Nous définissons deux fragments de DIL, appelés respectivement CDF (Control Design Formulas) et RF (Requirement Formulas), utilisés pour spécifier respectivement les systèmes de contrôle et les exigences. Puis nous montrons que ces fragments subsument les fragments de CoD qui ont été proposés dans [HHF+94, Rav94, BLR95] pour spécifier les systèmes de contrôle et les exigences.

Etant données une formule ψ_1 de CDF et une formule ψ_2 de RF, nous construisons deux automates à intégrateurs $\mathcal{H}_1$ et $\mathcal{H}_2$ tels que $\mathcal{H}_1$ reconnait l'ensemble des comportements satisfaisant ψ_1 et $\mathcal{H}_2$ reconnait l'ensemble des comportements falsifiant ψ_2. Nous utilisons cette construction pour montrer que le problème de vérifier si un système de contrôle donné satisfait ou non un ensemble d'exigences peut se réduire au complément du problème d'atteignabilité dans l'automate produit de $\mathcal{H}_1$ et $\mathcal{H}_2$.

Nous ferons un certain nombre de restrictions sur les fragments CDF et RF de sorte que ce problème de vérification soit décidable. Puis nous discuterons de la

minimalité de ces conditions. Nous montrons que sans ces conditions, la construction des automates $\mathcal{H}_1$ et $\mathcal{H}_2$ est en général impossible, car dans certains cas nous aurons besoin d'automates munis d'un nombre infini de variables.

Donc nous identifions des fragments de CDF et de RF, appelés CDF à *temps pur* et RF *simple*. Pour ces fragments, notre construction aboutira dans une classe particulière d'automates à intégrateurs, avec l'avantage de savoir que pour cette classe le problème d'atteignabilité a été montré décidable [ACH93].

Puis nous montrons que le fragment de CoD suggéré pour représenter les systèmes de contrôles et leurs exigences peut être traduit automatiquement dans le fragment CDF à *temps pur* ainsi que dans le fragment RF *simple*. Une conséquence directe est que la vérification de systèmes de contrôle de CoD par rapport à des exigences de CoD est décidable. Nous montrons aussi que pour tout graphe temporisé et toute formule de RF *simple*, le problème de la vérification est décidable. Nous aurons donc identifié un fragment de la logique temporelle de durée linéaire pour lequel le problème de la vérification pour les graphes temporisés est décidable. Nous prouvons aussi que le problème de la validité des formules de RF est décidable alors que le problème de leur satisfaisabilité ne l'est pas. Plus encore, nous montrerons que ce problème est Π^0_1-hard.

Le reste du chapitre est organisé comme suit. Dans la partie 7.1, nous présentons certaines notations qui sont utilisées dans ce chapitre.
Dans la partie 7.2, nous présentons la définition des automates à intégrateurs que nous utiliserons par la suite.
Dans la partie 7.3, nous définissons de la logique de calcul de durée (CoD) en donnant sa syntaxe et sa sémantique. Dans cette même partie nous donnerons aussi comme exemple de motivation, l'exemple du brûleur à gaz introduit dans [CHR91].
Dans la partie 7.4, nous présentons la syntaxe et la sémantique du langage de spécification DIL, puis nous le comparons avec le CoD.
Dans la partie 7.5, nous introduisons les fragments CDF et RF de DIL et nous montrons qu'ils subsument les fragments correspondant de CoD.
Dans la partie 7.7, nous posons le problème de vérification auquel nous nous intéressons et nous montrons qu'il peut être réduit au complément du problème d'atteignabilité dans un automate à intégrateurs.
Dans la partie 7.8, nous présentons nos résultats de décidabilité concernant le problème de la vérification aussi bien pour le problème de validité que pour celui de la satisfaisabilité des formules de RF.
Puis nous donnons un certain nombre de remarques en guise de conclusion dans la partie 7.9.

7.1 Préliminaires

Dans cette partie nous rappellerons brièvement certaines notions introduites dans les chapitres précédents, nous introduirons aussi de nouvelles définitions. Ces notions et ces définitions nous sont utiles tout au long de ce chapitre.

7.1.1 Contraintes générales linéaires

Soit $\mathcal{V}$ un ensemble des variables réelles. Une contrainte générale linéaire sur $\mathcal{V}$ est une combinaison linéaire d'inégalités de la forme $e(x_1, \cdots, x_n) \leq c$ où les x_i sont des variables dans $\mathcal{V}$, c est une constante entière et $e(x_1, \cdots, x_n) = \sum_{i=1}^{n} k_i \cdot x_i$, les k_i étant des constantes entières. Nous dénotons par $\mathcal{CG}_{\mathcal{V}}$ l'ensemble de toutes les contraintes générales linéaires sur $\mathcal{V}$.

Une valuation des variables dans $\mathcal{V}$ est une fonction $\nu \in [\mathcal{V} \rightarrow \mathbb{R}]$. Etant donné un ensemble de variable $X \subseteq \mathcal{V}$, nous dénotons par $\nu[X \mapsto 0]$ la valuation qui associe 0 à chaque variable de X et coïncide avec ν pour le reste des variables. Etant donné une contrainte linéaire g, nous dénotons par $\nu \models g$ le fait que g est satisfaite par la valuation ν.

Une contrainte générale linéaire est dite contrainte simple si c'est une combinaison booléenne de contraintes de la forme $e \leq c$ où e est soit x ou bien $x - y$ avec $x, y \in \mathcal{V}$.

7.1.2 Trajectoires, états temporisés et séquences d'états temporisés

Nous rappelons que le domaine du temps est l'ensemble des réels positifs. Nous définissons une trajectoire comme une application $\rho : \mathbb{R}_{\geq 0} \rightarrow \Sigma$ associant un état à chaque valeur du temps. De sorte que dans tout intervalle de temps fini ρ change de valeur un nombre fini de fois. Donc dans tout intervalle de temps fini, le nombre d'états vus par ρ est fini. Cette propriété est souvent appelée propriété de variabilité finie [BKP86, Nic92, HNSY94] (ρ a un nombre fini de points de discontinuité dans tout intervalle de temps fini).

Etant donnée une trajectoire ρ et une formule d'état π, la fonction caractéristique de π sur la trajectoire ρ est la fonction $\tilde{\rho}_\pi : \mathbb{R}_{\geq 0} \rightarrow \{0, 1\}$ définie par
$\forall t \in \mathbb{R}_{\geq 0}$. si $\rho(t) \models \pi$ alors $\tilde{\rho}_\pi = 1$ sinon $\tilde{\rho}_\pi = 0$.

Une trajectoire ρ est dites continue à droite si

$$\forall t \in \mathbb{R}_{\geq 0}. \exists \epsilon > 0. \forall t' \in [t, t + \epsilon[. \rho(t) = \rho(t')$$

Nous pouvons voir que pour toute trajectoire ρ continue à droite et à variation finie, il existe une séquence discrète (donc énumérable) $\{t_i\}_{i\in I\!N}$ telle que $t_0 = 0$, $\forall i \in I\!N.t_{i+1} > t_i$, $lim_{i\to\infty} t_i = \infty$ et $\forall t \in [t_i, t_{i+1}[.\rho(t) = \rho(t_i)$.

Nous définissons les états temporisés comme une paire $\langle s, t\rangle \in \Sigma \times I\!R_{\geq 0}$. Puis nous définissons la notion de séquence temporisée comme la séquence infinie d'états temporisés $\sigma = \{\langle s_i, t_i\rangle\}_{i\in I\!N}$ telle que $t_0 = 0$, $\forall i \in I\!N.t_{i+1} > t_i$ et $lim_{i\to\infty} t_i = \infty$.

Chaque séquence d'états temporisée $\sigma = \{\langle s_i, t_i\rangle\}_{i\in I\!N}$ génère une trajectoire continue à droite ρ_σ définie par $\forall t \in I\!R_{\geq 0}.\forall i \in I\!N.t_i \leq t < t_{i+1} \Rightarrow \rho_\sigma(t) = s_i$.

7.2 Automates à intégrateurs

Nous présentons dans cette partie les automates à intégrateurs que nous noterons par AI. Ces automates à intégrateurs étendent les systèmes à intégrateurs introduits dans [ACHH93, NOSY93, KPSY93] et que nous avons défini dans le chapitre 2 (mais en enlevant les structures de données discrètes), cette extension va dans le sens où nous rajoutons une simple condition d'acceptance. La condition d'acceptance que nous considérons correspond à la 1-acceptance de la théorie des ω-automates [Tho90]. C'est une condition qui consiste à n'accepter que les séquences d'exécutions qui passent au moins une fois par un nœud de contrôle appartenant au sous-ensemble de nœuds de contrôle d'acceptation.
Cette condition d'acceptance est très pratique pour exprimer le complément du problème d'atteignabilité dans un graphe à intégrateur comme un problème de langage vide. D'autres sortes de conditions d'acceptance peuvent être considérées, mais pour les problèmes auxquels nous nous intéressons dans ce chapitre nous pouvons nous suffire avec la 1-acceptance.

La définition des automates à intégrateurs (AI) est comme suit:

Définition 7.1 Soit $\mathcal{P}$ un ensemble fini de propositions atomiques et $\Sigma = 2^{\mathcal{P}}$. Un automate à intégrateurs $\mathcal{H}$ sur Σ est défini par les composantes suivantes:

- $\mathcal{X}$, un ensemble fini de variables continues.
- $\mathcal{L}$, un ensemble fini de nœuds de contrôle.
- $Init$, un ensemble fini de nœuds de contrôle ($Init \subseteq \mathcal{L}$).
- Acc, un ensemble fini de nœuds de contrôle d'acceptations (nœuds finaux).
- ε, un ensemble d'arcs. Chaque arc est un tuple $e = (q, g, X, q')$ où $q, q' \in \mathcal{L}$ sont respectivement le nœud source et le nœud cible, $g \in \mathcal{CG}_{\mathcal{X}}$ est une garde

qui doit être satisfaite pour que la transition puisse être effectuée et $X \subseteq \mathcal{X}$ est l'ensemble des variables qui doivent être remises à zéro lorsque la transition e est effectuée.

- Π, une fonction dans $[\mathcal{L} \rightarrow \Sigma]$, associant un état à chaque nœud de contrôle.
- I, une fonction de $[\mathcal{L} \rightarrow \mathcal{CG}_{\mathcal{X}}]$, associant à chaque nœud de contrôle q une condition d'invariance sous laquelle il est permis de rester dans q.
- ∂, une fonction dans $[\mathcal{L} \times \mathcal{X} \rightarrow \{0,1\}]$, associant à chaque nœud de contrôle q et chaque variable x, une pente dans $\{0,1\}$ à laquelle x évolue de manière continue lors du séjour dans q.

□

Dans ce qui suit, nous définissons l'ensemble des trajectoires continues à droite acceptées par un AI. Nous définissons le séquencement entre configurations temporisées. Une exécution de $\mathcal{H}$ partant d'une configuration temporisée initiale $\langle q, \nu, 0\rangle$ est une séquence infinie $\{\langle q_i, \nu_i, t_i\rangle\}_{i\in\mathbb{N}}$ telle que $\langle q, \nu, 0\rangle = \langle q_0, \nu_0, t_0\rangle$, $lim_{i\rightarrow\infty} t_i = \infty$ et

- $\forall i \in \mathbb{N}.\ t_{i+1} > t_i$ et
- $\forall i \in \mathbb{N}.\ \forall t.\ t_i \leq t < t_{i+1}.\ [\nu_i + (t - t_i)]_{q_i} \models I(q_i)$ et
- $\forall i \in \mathbb{N}.$
  - soit que $q_i = q_{i+1}$ et $\nu_{i+1} = [\nu_i + (t_{i+1} - t_i)]_{q_i}$,
  - ou bien que $\exists e = (q_i, g, X, q_{i+1}) \in \varepsilon.\ \nu_{i+1} = [\nu_i + (t_{i+1} - t_i)]_{q_i}[X \mapsto 0]$ et $[\nu_i + (t_{i+1} - t_i)]_{q_i} \models g$.

Il faut noter, que nous adoptons une sémantique bien particulière. En effet nous considérons un temps strictement monotone.

Toute exécution génère une séquence d'état temporisée et par conséquence une trajectoire continue à droite. La séquence d'états temporisée génèrée par $\{\langle q_i, \nu_i, t_i\rangle\}_{i\in\mathbb{N}}$ est la séquence $\{\langle \Pi(q_i), t_i\rangle\}_{i\in\mathbb{N}}$.

Donc, une trajectoire continue à droite ρ est acceptée par $\mathcal{H}$ s'il existe une exécution $\{\langle q_i, \nu_i, t_i\rangle\}_{i\in\mathbb{N}}$ de $\mathcal{H}$ partant d'une configuration temporisée initiale, qui génère ρ et telle que $\exists i \in \mathbb{N}.\ q_i \in Acc$. Nous notons $\mathcal{T}(\mathcal{H})$ l'ensemble des trajectoires (continues à droite) acceptées par $\mathcal{H}$.

7.2.1 Cas particuliers

Nous définissons dans cette partie quelques sous-classes d'automates à intégrateurs que nous considérons pour établir dans ce chapitre des résultats de décidabilités.

Nous rappelons qu'une horloge x est une variable telle que pour tout nœud q, $\partial(q,x) = 1$. Nous rappelons aussi qu'un automate temporisé est un automate à intégrateur où toutes les variables sont des horloges, toutes les gardes et les invariants sont des contraintes simples.

Nous définissons un automate à intégrateur simple comme un AI qui satisfait les conditions suivantes:

- Tous les invariants sont des contraintes simples et les variables qui apparaîssent dans ces contraintes sont uniquement des horloges.
- Toutes les gardes sont simples, contiennent uniquement des horloges et elles contiennent au plus une conjonction qui n'est pas simple. Dans le cas où ce n'est pas simple, c'est de la forme $c \sim \sum_{i=1}^{n} k_i x_i \sim d$ où $\sim \in \{<, \leq\}, k_i \in I\!N$ et $c, d \in \mathbb{Z}$.
- Pour tout chemin partant d'un nœud initial allant vers un nœud accepteur, ce chemin contient au plus une garde qui n'est pas simple.

Finalement un automate à intégrateur (respectivement un automate temporisé) est appelé graphe à intégrateur (respectivement graphe temporisé), si tous ses nœuds sont des nœuds accepteurs (c'est à dire $Acc = \mathcal{L}$).

7.3 La logique de calcul de durées (CoD)

La logique de calcul de durées (que l'on note CoD) a été introduite dans [CHR91] comme une notation permettant de spécifier des systèmes temps réels et comme un calcul permettant de vérifier des théorèmes relatifs à ces systèmes. Le CoD est une extension de la logique d'intervalle ITL introduite dans [Mos85]. Le CoD a été à l'origine conçu pour formaliser le raisonnement sur les exigences temps-réel de systèmes de contrôle; mais il a aussi été utilisé a bien d'autres niveaux d'abstractions.

Dans cette partie nous introduisons la syntaxe et la sémantique de la logique CoD ainsi qu'un exemple d'illustration.
Pour commencer nous donnons un exemple de motivation extrait de [CHR91].

Exemple 7.1 L'exemple que nous décrivons ici est celui du *brûleur à gaz* introduit dans [CHR91] et que nous présentons dans le chapitre 2. Dans cet exemple il y a un

état que nous appelons *fuite* qui dénote un état indésirable mais inévitable du système qui est présent lorsque qu'il y a un flux de gaz non brûlé se dégageant du gicleur du *brûleur à gaz.* Un tel état peut être représenté par une fonction du temps vers l'ensemble $\{0,1\}$, où le 1 dénote que le système est dans l'état *fuite* alors que 0 dénote que le système n'est pas dans cet état.
Pour des raisons de sécurité, un bon ingénieur peut calculer la ventilation nécessaire pour une combustion normale afin de se prémunir d'une accumulation dangereuse de gaz; nous supposerons que ces calculs donnent que l'accumulation des temps passés dans le nœud *fuite* doit être inférieure à 4% du temps global.
L'intervalle de temps durant lequel le système est observé est supérieur à 2 minutes et 30 secondes, sinon les exigences que nous venons d'évoquer pourraient être faussées lors du début de l'état *fuite.*
Cette exigence est formalisée par:
Req-1

$$(e-b) \geq 150sec. \Rightarrow 25 \int_b^e fuite(t)dt \leq (e-b)$$

pour tout intervalle $[b,e]$, $b \leq e$.
$\int_b^e fuite(t)dt$ est la mesure de la durée de *fuite* dans l'intervalle $[b,e]$.

Nous décrivons maintenant comment la conception du *brûleur à gaz* est établie afin que les exigences décrites ci-dessus soient satisfaites.
Par exemple la *fuite* doit être détectable et interrompable en moins d'une seconde; il faut alors faire une pause d'au moins 30 secondes avant de risquer une autre *fuite* en remettant le gaz. Nous utilisons c et f pour dénoter les extrémités d'un sous-intervalle arbitraire de $[b,e]$, les deux décisions peuvent être formalisées par:

Des-1

$$\int_c^f fuite(t)dt = (f-c) \Rightarrow (f-c) \leq 1sec.$$

Des-2

$$fuite(c) \wedge fuite(f) \wedge (\exists t : c < t < f \wedge \neg fuite(t)) \Rightarrow 30sec. \leq (f-c)$$

L'ingénieur concepteur devra s'assurer que la conjonction des deux formules de conception implique la formule d'exigence avant l'implantation. □

7.3.1 Syntaxe du CoD

On commence par définir la syntaxe du CoD. Pour cela on introduit l'alphabet des symboles qui est le suivant:

- Un symbole spécial ℓ qui est utilisé pour spécifier la longueur d'un intervalle.
- Un ensemble de variables globales $x, y, z, \cdots$.
- Un ensemble de variables d'états $X, Y, Z, \cdots$.
- Un ensemble de fonctions $n - aires\ f_i^n$, $i, n \geq 0$.
- Un ensemble de prédicats $n - aires\ A_i^n$, $i, n \geq 0$.
- Les symboles spéciaux $vraie$, $fausse$.
- Les opérations booléennes $\vee$ et $\neg$.
- Le quantificateur $\forall$ et la modalité ";".

On définit l'ensemble des **états** comme suit:

1. $fausse, vraie$ et toute variable d'état sont des états.
2. Toute combinaison booléenne d'états est un état.

On définit les **durées** comme suit:
Pour tout état P: $\int P$ est une durée.

L'ensemble des **termes** sont générés par:

1. ℓ, la durée $\int P$ et les variables globales x sont des termes.
2. Si $r_1, r_2, \cdots, r_n$ sont des termes et f_i^n est une fonction $n - aire$ alors $f_i^n(r_1, r_2, \cdots, r_n)$ est un terme.

On définit les **formules atomiques de durée** comme suit:
Si A_i^n est un prédicat $n - aire$ et que $r_1, r_2, \cdots, r_n$ sont des termes alors $A_i^n(r_1, r_2, \cdots, r_n)$ est une formule atomique de durée.

Finalement les **formules de durées** sont générées par:

1. $vrai$, $faux$ et les formules atomiques de durées sont des formules de durées.

2. Si $\mathcal{D}_1$ et $\mathcal{D}_2$ sont des formules de durées alors $\neg\mathcal{D}_1$, $\mathcal{D}_1 \vee \mathcal{D}_2$, $\mathcal{D}_1;\mathcal{D}_2$ et $(\forall x)\mathcal{D}_1$ avec x une variable globale sont des formules de durées.

Dans la logique CoD il existe un nouvel opérateur très important, le ";". Il est appelé séparateur ("chop" en anglais). Intuitivement, un intervalle satisfait une formule $\varphi_1;\varphi_2$ si il peut être divisé en deux sous-intervalles I_1 et I_2 tels que I_1 satisfait φ_1 et I_2 satisfait φ_2.
Les notations suivantes sont utilisées:
$\Diamond\mathcal{D}$ pour noter $vraie;\mathcal{D};vraie$ et
$\Box\mathcal{D}$ pour noter $\neg\Diamond\neg\mathcal{D}$.

7.3.2 Sémantique du CoD

On définit la sémantique du CoD comme suit:
Nous supposons qu'il existe une fonction totale $\underline{f_i^n} : \mathbb{R}^n \mapsto \mathbb{R}$ qui est associée à chaque symbole de fonction $n-aire$ f_i^n et une fonction totale $\underline{A_i^n} : \mathbb{R}^n \mapsto \{vraie, fausse\}$ qui est associée à chaque symbole de prédicat A_i^n.
Les variables d'états sont interprétées comme une fonction à deux valeurs possibles.
Les variables globales sont interprétées comme des réels.

Nous rappelons qu'une trajectoire ρ est définie comme suit: $\rho : \mathbb{R} \mapsto 2^{\mathcal{P}}$ avec $\mathcal{P}$ un ensemble de propositions atomiques. ρ a un nombre fini de points de discontinuité dans tout intervalle.

Nous disons que (ρ, b, e) et (ρ', b, e) sont x-équivalentes si elles associent la même valeur à toute variable globale y différente de x.
Etats:

$(\rho, t) \models vraie$

$(\rho, t) \models X$ si et seulement si $X \in \rho(t)$

$(\rho, t) \models \neg P$ si et seulement si $(\rho, t) \not\models P$

$(\rho, t) \models P \vee Q$ si et seulement si $(\rho, t) \models P$ et $(\rho, t) \models Q$

Les durées, les termes et les formules de durée sont interprétés sur des intervalles de temps.
Termes:

$T_{(\rho,b,e)}[\![\ell]\!] = e - b$

$$T_{(\rho,b,e)}[\![\int P]\!] = \int_b^e \mathcal{I}_{\mathcal{D}}[\![P]\!](t)dt \text{ avec } \mathcal{I}_{\mathcal{D}}[\![P]\!](t) = 1 \text{ si } P \in \rho(t) \text{ sinon } 0$$

$$T_{(\rho,b,e)}[\![x]\!] = \underline{x}$$

$$T_{(\rho,b,e)}[\![f_i^n(r_1,\cdots,r_n)]\!] = \underline{f_i^n}(T_{(\rho,b,e)}[\![r_1]\!],\cdots,T_{(\rho,b,e)}[\![r_n]\!])$$

Formules de durée: La définition du fait qu'une trajectoire ρ sur intervalle de bornes b et e satisfait une formule de durée est:

$(\rho,b,e) \models vraie$

$(\rho,b,e) \models A_i^n(r_1,\cdots,r_n)$ si et seulement si $\underline{A_i^n}((\rho,b,e)[\![r_1]\!],\cdots,(\rho,b,e)[\![r_n]\!]) = vraie$

$(\rho,b,e) \models \neg\mathcal{D}$ si et seulement si $(\rho,b,e) \not\models \mathcal{D}$

$(\rho,b,e) \models \mathcal{D}_1 \vee \mathcal{D}_2$ si et seulement si $(\rho,b,e) \models \mathcal{D}_1$ ou $(\rho,b,e) \models \mathcal{D}_2$

$(\rho,b,e) \models \mathcal{D}_1;\mathcal{D}_2$ si et seulement si $\exists m \in [b,e] : (\rho,b,m) \models \mathcal{D}_1$ et $(\rho,m,e) \models \mathcal{D}_2$

$(\rho,b,e) \models (\forall x)\mathcal{D}$ si et seulement si $(\rho',b,e) \models \mathcal{D}$,

pour tout (ρ',b,e) x-équivalente à (ρ,b,e) .

Une formule de durée $\mathcal{D}$ est vraie sur la trajectoire ρ (que nous notons $\rho_{\mathcal{D}} \models_D \mathcal{D}$) si et seulement si $(\rho,b,e) \models \mathcal{D}$ pour tout intervalle $[b,e]$ de la trajectoire ρ. Plus encore nous disons que $\mathcal{D}$ est valide (que nous notons $\models_D \mathcal{D}$) si et seulement si $\rho_{\mathcal{D}} \models \mathcal{D}$ pour toute trajectoire ρ.
Nous utilisons aussi les abréviations suivantes:

$$\lceil P \rceil \triangleq (\int P = \ell) \wedge (\ell > 0)$$

$$\lceil\ \rceil \triangleq \ell = 0$$

Les exigences et le système de contrôle du *brûleur à gaz* décrits dans l'exemple 7.1 peuvent alors être exprimés en CoD de la manière suivante:
Req-1

$$\ell \geq 150 \Rightarrow 25 \int fuite \leq \ell$$

Des-1

$$\Box(\lceil fuite \rceil \Rightarrow \ell \leq 1)$$

Des-2

$$\Box(((\Diamond\lceil fuite \rceil);(\Diamond\lceil \neg fuite \rceil);(\Diamond\lceil fuite \rceil)) \Rightarrow \ell \geq 30)$$

7.4 La logique DIL

Dans cette partie, nous définissons une logique temporelle d'intervalles que nous notons DIL ("Duration Interval Logic"). Cette logique est essentiellement une extension du calcul de durées propositionnelles (CoD) [CHR91], cette extension est une sorte de quantification permettant d'introduire des variables qui mesurent la durée des formules d'états. Cette quantification a été introduite dans [BES93] et est similaire à la quantification de remise à zéro de [Alu91, HNSY94].

Nous noterons $P, Q, \cdots$ les éléments de l'ensemble des propositions atomiques $\mathcal{P}$. Nous utilisons $x, y, \cdots$ pour noter les éléments de l'ensemble des variables de durées Var. Nous utilisons les lettres $f, g, \cdots$ pour noter les éléments des contraintes générales linéaires sur Var. Ces contraintes sont appelées contraintes de durées. Puis finalement nous utilisons $\pi, \delta, \cdots$ pour noter des formules d'état.

Nous définissons la logique DIL par l'ensemble de formules définies par la grammaire suivante:

$$\varphi ::= \uparrow\pi \mid f \mid [x : \pi].\ \varphi \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi;\varphi$$

En plus des notations standards associées aux opérations booléennes, nous utilisons les abréviations suivantes qui sont identiques à celles qui ont été introduites dans la partie précédente

- $\Diamond\varphi$ pour noter $vraie;\varphi;vraie$ et
- $\Box\varphi$ pour noter $\neg\Diamond\neg\varphi$.

Plus encore, nous écrivons $[x_1, \cdots, x_n : \pi_1, \cdots, \pi_n].\ \varphi$, $[x_i : \pi_i]_{i=1}^n.\ \varphi$, ou $[\vec{x} : \vec{\pi}].\ \varphi$ pour $[x_1 : \pi_1]\cdots[x_n : \pi_n].\ \varphi$. Nous écrivons aussi $x.\ \varphi$ pour $[x : vraie].\ \varphi$ et $\ell \sim c$ pour $x.\ x \sim c$. Pour une formule d'état π, nous écrivons la formule $\lceil\pi\rceil$ à la place de $[x, y : vraie, \pi].\ x - y = 0 \wedge x > 0$.

Les formules de la logique DIL décrivent des propriétés d'intervalles sur les trajectoires. Un intervalle I satisfait $\uparrow\pi$, si π change sa valeur de faux à vraie exactement à

la borne gauche de I donc au début de I. Cet opérateur a été introduit dans [CL94] (Mean Value Calculus).
La formule $\Diamond\varphi$ veut dire qu'il existe un sous-intervalle qui satisfait φ, $\Box\varphi$ veut dire que tout sous-intervalle satisfait φ.
Dans la formule $\phi = [x : \pi].\ \varphi$, la variable de durée x est associée à la formule d'état π. Considérons que ϕ est interprétée dans un intervalle I et soit f une contrainte de durée contenant x qui apparait dans φ. Alors, lorsque f est interprétée dans un sous-intervalle I' of I, x représente la durée de π à partir de la borne gauche de I (début de l'intervalle I) à la borne droite de I' (fin de l'intervalle I'). Par exemple, la formule "$[x, y : \pi, \delta].\ (x < y; vraie)$" exprime le fait qu'il existe un préfixe de l'intervalle considéré tel que dans ce préfixe la durée de π est inférieure à la durée de δ. La formule $\lceil\pi\rceil$ est satisfaite par un intervalle I, si cet intervalle n'est pas réduit à un point et que π est vraie presque partout dans I, c'est à dire que π est vraie partout dans I sauf dans un nombre fini de points. Il est à noter que la variable x qui est associée à la formule *vraie* compte le temps écoulé depuis le point où elle a été introduite. Une telle variable est appelée *horloge.*

Dans la formule $\phi = [x : \pi].\ \varphi$, le constructeur "$[x : \pi]$" lie la variable x à φ. Sans pour autant que ce soit une restriction, nous pouvons supposer que toute variable est liée au plus une fois dans toute formule. Ce n'est pas une restriction car dans le cas où une variable est liée plusieurs fois dans une même formule, il suffira de renommer la variable à chaque nouvelle liaison, puisqu'une nouvelle liaison remet à zéro la variable en question. Toute variable apparaissant dans une formule est soit liée soit libre. Nous dénotons par $\mathcal{F}(\varphi)$ l'ensemble des variables libres dans φ. La formule φ est fermée si $\mathcal{F}(\varphi) = \emptyset$. Dorénavant lorsqu'il n'y aura pas d'ambiguïté nous appellerons tout simplement formule lorsque nous ferons allusion à des formules fermées.

Etant donnée une trajectoire ρ, nous définissons une relation de satisfaction $\models$ entre des intervalles dans ρ et des formules de DIL. Puisque les formules de DIL peuvent contenir des variables libres la relation de satisfaction est paramétrée par une *affectation de position* $\mathcal{I}$ associe à chaque variable de durée le temps auquel elle a été initialisée et une *affectation de formules d'états* γ qui associe à chaque variable x la formule d'état π dont la durée est mesurée par x. Soit F l'un des deux types d'affectations $\mathcal{I}$ ou γ. Alors nous notons par $F[x \mapsto v]$ la fonction qui associe à x l'objet v et coïncide avec F pour les autres variables. Soient φ une formule de DIL, ρ une trajectoire, et $a, b \in \mathbb{R}_{\geq 0}$ avec $b \geq a$. Nous notons par $(\rho, a, b) \models_{\mathcal{I},\gamma} \varphi$ le fait que l'intervalle de ρ avec les bornes a et b satisfait φ; lorsque φ est fermée, les fonctions $\mathcal{I}$ et γ sont omises. Nous définissons $(\rho, a, b) \models_{\mathcal{I},\gamma} \varphi$ de manière inductive sur la structure de φ.

$$
\begin{array}{lll}
(\rho,a,b) \models_{\mathcal{I},\gamma} \uparrow\pi & \text{ssi} & \exists \epsilon > 0.\ \forall t \in]a, a+\epsilon[.\ \tilde{\rho}_\pi(t) = 1, \text{ et} \\
 & & \text{soit que } a = 0 \text{ ou bien } \forall t \in]a-\epsilon, a[.\ \tilde{\rho}_\pi(t) = 0, \\
(\rho,a,b) \models_{\mathcal{I},\gamma} f & \text{ssi} & \nu \models f \quad \text{où } \nu \text{ satisfait } \forall x \in \mathcal{V}ar.\ \nu(x) = \int_{\mathcal{I}(x)}^{b} \tilde{\rho}_{\gamma(x)}(t)\, dt, \\
(\rho,a,b) \models_{\mathcal{I},\gamma} [x:\pi].\varphi & \text{ssi} & (\rho,a,b) \models_{\mathcal{I}',\gamma'} \varphi \quad \text{où } \mathcal{I}' = \mathcal{I}[x \mapsto a] \text{ et } \gamma' = \gamma[x \mapsto \pi], \\
(\rho,a,b) \models_{\mathcal{I},\gamma} \neg\varphi & \text{ssi} & (\rho,a,b) \not\models_{\mathcal{I},\gamma} \varphi \\
(\rho,a,b) \models_{\mathcal{I},\gamma} \varphi_1 \vee \varphi_2 & \text{ssi} & (\rho,a,b) \models_{\mathcal{I},\gamma} \varphi_1 \text{ ou } (\rho,a,b) \models_{\mathcal{I},\gamma} \varphi_2, \\
(\rho,a,b) \models_{\mathcal{I},\gamma} \varphi_1;\varphi_2 & \text{ssi} & \exists m \in [a,b].\ (\rho,a,m) \models_{\mathcal{I},\gamma} \varphi_1 \text{ et } (\rho,m,b) \models_{\mathcal{I},\gamma} \varphi_2.
\end{array}
$$

Nous définissons une relation de satisfaction entre une trajectoire et les formules fermées de DIL. Une trajectoire ρ *satisfait* une formula fermée φ si et seulement si $\forall t \geq 0.\ (\rho, 0, t) \models \varphi$. Nous notons par $[\![\varphi]\!]$ (resp. $[\![\varphi]\!]_{rc}$) l'ensemble des trajectoires (respectivement trajectoires continues à droite) satisfaisant φ. Une formule φ de DIL est dite *satisfaisable*, si il existe une trajectoire qui satisfait φ. Elle est dite *valide*, si toute trajectoire satisfait φ. Deux formules de DIL φ et φ' sont dites *équivalentes*, si $[\![\varphi]\!] = [\![\varphi']\!]$. Elles sont dites *congruentes*, si $\Box(\varphi \Leftrightarrow \varphi')$ est valide.

Nous définissons la sémantique de DIL de cette manière afin qu'elle coïncide avec celle de CoD. Cela nous permet d'appliquer les résultats que nous établissons pour DIL à la logique de CoD. En effet, nous pouvons adopter la vue que les formules de DIL sont des formules de CoD écrites de manière plus compacte. Nous discuterons avec plus de détails des relations entre les formules de CoD et celles de DIL.

Nous allons commencer par décrire informellement par des commentaires l'interprétation des contraintes de durées. Considérons la formule suivante.

$$\varphi = [x:\pi].\ (x \leq 1; [y:\delta].\ x - y < 0). \tag{7.1}$$

Un intervalle I satisfait φ si il peut être divisé en deux intervalles I_1 et I_2 tels que la durée de π dans l'intervalle I_1 est inférieure à 1 et que la durée de δ dans I_2 est supérieure à celle de π dans tout l'intervalle I. Donc, lorsque nous interprétons la formule $x - y < 0$ dans I_2, la variable x évalue la durée de π dans I et non pas dans I_2 seulement.
Par contre, si nous considérons la formule $\varphi' = [x:\pi].\ (x \leq 1; [x', y:\pi, \delta].\ x' - y < 0)$ à la place de φ, alors x' représente la durée de π dans l'intervalle I_2 uniquement. Il est à noter que φ' est équivalente à $([x:\pi].\ x \leq 1); ([x', y:\pi, \delta].\ x' - y < 0)$.

Maintenant nous discutons de l'expressivité de DIL et nous ferons une étude comparative avec le CoD linéaire. C'est à dire le CoD avec des formules de la forme $\sum_{i=1}^{n} k_i \cdot \int \pi_i \sim c$ où les k_i et c sont des constantes entières. Dans ce qui suivra, nous ferons uniquement référence au CoD linéaire que nous écrirons simplement CoD. Nous noterons aussi PCoD pour le CoD propositionnel.

Avant tout, nous pouvons remarquer que PCoD correspond exactement au fragment de DIL contenant les formules sans l'opérateur "$\uparrow$" et tel que, entre toute occur-

rence d'une variable x et sa liaison (c'est à dire "$[x : \pi]$"), il n'y a pas de séparateur ";", par exemple la formule φ' ci-dessus.
Vu que les formules de CoD de la forme $e(\int \pi_1, \cdots, \int \pi_n) \sim c$ peuvent être écrite dans DIL comme $[x_i : \pi_i]_{i=1}^n$. $e(x_1, \cdots, x_n) \sim c$, par conséquent, toute la logique CoD peut être traduite dans DIL du premier ordre (DIL étendu avec la quantification du premier ordre). D'autre part , nous pouvons montrer que toute formule de DIL sans l'opérateur $\uparrow$ peut être traduite dans CoD en utilisant la quantification du premier ordre; nous rappelons que CoD possède des quantification du premier ordre alors que DIL n'en contient pas. En fait, DIL est strictement plus expressive que PCoD puisque "$[x : \pi]$" est une sorte de quantification qui permet la comparaison des durées de formules d'états mesurées dans des intervalles différents. Par exemple, la formule φ ci-dessus (7.1), n'est pas exprimable en PCoD. Mais elle est équivalente, à la formule de CoD

$$\forall x.\, (x = \int \pi) \Rightarrow (\int \pi \leq 1; x - \int \delta < 0)$$

Si nous considérons la logique DIL$^-$ comme étant DIL moins l'opérateur $\uparrow$, nous obtenons donc les inclusions strictes suivantes:

Proposition 7.4.1 *PCoD* $\subset$ *DIL*$^-$ $\subset$ *CoD.*

Par contre il n'est pas certain que l'on ait $DIL \subset CoD$ car il n'est pas certain que nous arrivons à transcrire l'opérateur $\uparrow$ dans CoD. Les formules de la forme $\uparrow\pi$ ne sont pas dans CoD. Nous avons néanmoins inclus l'opérateur $\uparrow$, lequel a été introduit dans [CL94], pour permettre l'expression de plusieurs propriétés de manière naturelle. La principale observation est que la formule $[x : \pi].\varphi$ exprime la même propriété que $\forall x.(x = \int \pi \Rightarrow \varphi)$.
Donc DIL$^-$ est moins expressive que CoD mais plus expressive que le fragment propositionel de CoD. En particulier, les formules contenant des contraintes de durées des états dans différentes phases ne sont pas expressibles dans le CoD propositionel, mais expressibles dans DIL et CoD; en effet la formule suivante de DIL
$[x : P].(\phi; [y : Q].\phi' \wedge x - y \leq 0)$ exprime la même propriété que la formule suivante de CoD: $\forall x.x = \int P \Rightarrow (\phi; (\phi' \wedge x - \int Q \leq 0))$.

7.5 Conception de systèmes de contrôles et spécification des exigences

Nous définissons dans cette partie deux sous-ensembles de DIL, appelés CDF (Control Design Formulas en anglais) et RF (Requirement Formulas en anglais). Le fragment

CDF servira à décrire des systèmes de contrôle alors que le fragment RF servira à décrire les exigences requises par des systèmes hybrides. Nous montrons qu'ils subsument les fragments de CoD qui ont été identifiés dans [HHF+94, Rav94] pour la description de systèmes de contrôle et des exigences de systèmes hybrides.

7.5.1 Formules de contrôle et d'exigence

Aussi bien les formules de CDF que de RF que nous considérons sont des combinaisons booléennes de formules d'invariance positives de la forme $\Box(\varphi \Rightarrow \varphi')$ où φ et φ' satisfont des conditions spécifiques que nous définirons dans ce qui suit. Cette forme est comme nous pouvons le voir bien particulière, nous la retrouvons dans [HHF+94]. Cette forme est assez naturelle, puisqu'elle dit que sur tout intervalle dans lequel φ est vraie alors nous avons φ'; c'est une forme très intéressante aussi bien pour spécifier des systèmes de contrôle que des exigences.

Soit une spécification d'un système de contrôle décrite par la formule φ_d de CDF et une spécification d'exigences décrite par la formule φ_R de RF, dans la prochaine partie, nous montrons comment nous réduisons le problème de savoir si φ_d satisfait ou non φ_R au problème d'atteignabilité d'un système linéaire hybride bien particulier.

Pour ce, nous commençons par introduire le fragment positif DIL, appelé DIL$^+$. Les négations sur l'opérateur de séparation ";" ne sont pas permises. L'ensemble des formules de DIL$^+$ est défini par la grammaire suivante:

$$\varphi \ ::= \ \neg\uparrow\pi \mid \uparrow\pi \mid \lceil\pi\rceil \mid f \mid [x:\pi].\,\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi;\varphi$$

Nous définissons une *forme canonique* pour les formules de DIL$^+$. Elle correspond à l'ensemble des formules décrites par la grammaire suivante:

$$\begin{aligned}
\varphi &::= \bigvee \phi \\
\phi &::= [\vec{x}:\vec{\pi}].\,((\beta \wedge \xi);\phi) \mid [\vec{x}:\vec{\pi}].\,(\beta \wedge \xi) \\
\beta &::= \mu \mid \mu;\beta \\
\mu &::= \{\bigwedge \uparrow\pi \wedge\}\{\bigwedge \neg\uparrow\pi \wedge\}\lceil\pi\rceil \mid \mathit{vraie} \\
\xi &::= \bigwedge e \sim c, \sim \in \{<, >, \leq, \geq\}
\end{aligned}$$

Nous pouvons prouver qu'à toute formule de DIL$^+$ nous pouvons associer une formule sous la forme canonique qui lui soit congruente. La preuve peut être effectuée simplement à l'aide des règles qui régissent les opérations booléennes et l'opérateur de séparation ";". Principalement il faudra utiliser dans cette preuve que la conjonction de deux formules munies de séparateurs est équivalente à la disjonction de formules munies de séparateurs. Nous pouvons voir ce point avec plus de détails dans la preuve 7.1.

Proposition 7.5.1 *Toute formule de DIL$^+$ a une formule sous la forme canonique qui lui est congruente.*

Preuve 7.1 Nous pouvons transformer une formule de DIL$^+$ en une formule sous la forme canonique en appliquant la transformation des règles dont la correction est basée sur les lois standards pour les opérations booléennes et les lois exprimant la distributivité de l'opérateur "séparateur" ";" pour la disjonction et la conjonction. Pour la disjonction, nous avons une règle basée sur le fait que $(\phi_1 \vee \phi_2); \phi$ et $(\phi_1; \phi) \vee (\phi_2; \phi)$ sont congruentes. Nous avons aussi la règle qui dit que $\phi; (\phi_1 \vee \phi_2)$ et $(\phi; \phi_1) \vee (\phi; \phi_2)$ sont congruentes. Pour la conjonction, nous avons une règle qui transforme une formule $\phi_1 \wedge \phi_2$ en la formule de la forme $\bigvee \phi$.
Pour des raisons de clarté nous illustrons ici seulement le cas où on a ϕ_1 et ϕ_2 avec
$\phi_1 = [\vec{x}_1 : \vec{\pi}_1].\ ((\lceil \delta_1 \rceil \wedge \xi_1); [\vec{x}_2 : \vec{\pi}_2].\ (\lceil \delta_2 \rceil \wedge \xi_2))$ et
$\phi_2 = [\vec{y}_1 : \vec{\lambda}_1].\ ((\lceil \delta'_1 \rceil \wedge \xi'_1); [\vec{y}_2 : \vec{\lambda}_2].\ (\lceil \delta'_2 \rceil \wedge \xi'_2))$

Il n'est pas difficile de voir que $\phi_1 \wedge \phi_2$ est congruente à la disjonction des formules suivantes:
$[\vec{x}_1, \vec{y}_1 : \vec{\pi}_1, \vec{\lambda}_1].\ ((\lceil \delta_1 \wedge \delta'_1 \rceil \wedge \xi_1 \wedge \xi'_1); [\vec{x}_2, \vec{y}_2 : \vec{\pi}_2, \vec{\lambda}_2].\ (\lceil \delta_2 \wedge \delta'_2 \rceil \wedge \xi_2 \wedge \xi'_2))$
$[\vec{x}_1, \vec{y}_1 : \vec{\pi}_1, \vec{\lambda}_1].\ ((\lceil \delta_1 \wedge \delta'_1 \rceil \wedge \xi_1); [\vec{x}_2 : \vec{\pi}_2].\ ((\lceil \delta_2 \wedge \delta'_1 \rceil \wedge \xi'_1); ([\vec{y}_2 : \vec{\lambda}_2].\ (\lceil \delta_2 \wedge \delta'_2 \rceil \wedge \xi_2 \wedge \xi'_2))))$
$[\vec{x}_1, \vec{y}_1 : \vec{\pi}_1, \vec{\lambda}_1].\ ((\lceil \delta_1 \wedge \delta'_1 \rceil \wedge \xi'_1); [\vec{y}_2 : \vec{\lambda}_2].\ ((\lceil \delta_1 \wedge \delta'_2 \rceil \wedge \xi_1); ([\vec{x}_2 : \vec{\pi}_2].\ (\lceil \delta_2 \wedge \delta'_2 \rceil \wedge \xi_2 \wedge \xi'_2))))$

■

Maintenant, nous définissons des sous-classes de formules de DIL$^+$ en introduisant des conditions sur leur forme canonique. Pour cela, nous avons besoin de la notation suivante. Etant donnée une formule de la forme μ introduite ci-dessus, nous définissons $\Phi(\mu) =$ (si $\mu = \{\wedge \uparrow \pi \wedge\}\{\wedge \neg \uparrow \pi' \wedge\}\lceil \pi'' \rceil$ alors $\{\wedge \pi \wedge\}\pi''$ sinon *vraie*).

Soit ϕ une formule de DIL$^+$ sous la forme canonique donnée par

$$[\vec{x}_1 : \vec{\pi}_1].\ ((\mu_1 \wedge \xi_1); \cdots [\vec{x}_j : \vec{\pi}_j].\ ((\mu_j \wedge \xi_j); \cdots [\vec{x}_n : \vec{\pi}_n].\ (\mu_n \wedge \xi_n) \cdots)) \cdots) \quad (7.2)$$

La première sous-classe que nous considérons, consiste en celle que nous nommerons formules *queue-sans-contrainte* qui est définie comme suit:

Définition 7.2 *queue-sans-contrainte*
Les formules que nous appelons *queue-sans-contrainte* sont l'ensemble des formules de la forme (7.2) avec $\mu_n = vraie$ et $\xi_n = vraie$. □

Nous pouvons voir que pour toute formule queue-sans-contrainte, si elle est satisfaite par un intervalle $[a, b]$, alors elle est satisfaite par tout intervalle plus grand $[a, d]$.

Maintenant nous définissons la classe de formules *sans-π-alternation*, laquelle est un sous-ensemble de la classe de formules queue-sans-contrainte. Intuitivement, une formule ϕ est sous forme (7.2) est sans-π-alternation, si pour tout intervalle satisfaisant

$\Phi(\mu_1);\cdots;\Phi(\mu_{n-1})$, la formule $\uparrow\pi$ peut seulement être vraie au début de l'intervalle. La définition formelle est comme suit:

Définition 7.3 *sans-π-alternation*
Nous disons qu'une formule ϕ (7.2) queue-sans-contrainte est *sans-π-alternation* si $\forall j \in \{1,\cdots,n-1\}$, la conjonction $\pi \wedge \Phi(\mu_j)$ est soit équivalente à $fausse$ soit à $\Phi(\mu_j)$, plus encore $\pi \wedge \Phi(\mu_j) \neq fausse$ implique que $\forall k.\ 1 \leq k < j,\ \pi \wedge \Phi(\mu_k) \neq fausse$. □

La première partie de la définition de sans-π-alternation dans laquelle nous avons $\forall j \in \{1,\cdots,n-1\}$, la conjonction $\pi \wedge \Phi(\mu_j)$ est soit équivalente à $fausse$ soit à $\Phi(\mu_j)$, cela veut dire que pour chaque phase j soit que π ou bien que $\neg\pi$ est vraie durant toute la phase.
La seconde partie de la définition dit que $\pi \wedge \Phi(\mu_j) \neq fausse$ implique que $\forall k.\ 1 \leq k < j,\ \pi \wedge \Phi(\mu_k) \neq fausse$, cela veut dire qu'à partir de l'instant auquel π devient fausse elle restera fausse pour toujours. Donc si π est fausse au départ elle sera toujours fausse. Par contre si π est vraie au départ, soit que π reste toujours vraie soit qu'elle peut devenir fausse pour toujours et ce changement ne peut avoir lieu qu'en début d'une phase.

La prochaine sous-classe de formules de DIL$^+$ que nous définissons est la classe de formules *sans-enchevêtrement.* Considérons la formule ϕ de la forme (7.2), et un intervalle I qui satisfait ϕ. A partir de la sémantique de DIL, il y a n points dans l'intervalle I qui délimitent les sous intervalles où les formules $\mu_i \wedge \xi_i$ sont satisfaites. Intuitivement, la condition de sans-enchevêtrement implique que le choix de tels points est unique, sauf pour le dernier lorsque ϕ est queue-sans-contrainte. Nous obtenons alors la définition suivante:

Définition 7.4 Une formule ϕ de la forme (7.2) est dite *sans-enchevêtrement* si $\forall j \in \{1,\cdots,n-1\}$, une des conditions suivantes est vraie:

1. $\Phi(\mu_j) \wedge \Phi(\mu_{j+1}) = fausse$, ou
2. $j = n-1$ et ϕ est queue-sans-contrainte, ou
3. ξ_j est équivalent à $\xi \wedge (x = c)$, pour un ξ, une horloge x dans $\vec{x}_j$, et une constante c.

□

Finalement, nous introduisons une sous-classe de formules sans-enchevêtrement, c'est la classe des formules *fortement sans-enchevêtrement.*

Définition 7.5 Une formule ϕ sans-enchevêtrement de la forme (7.2) est dite *fortement sans-enchevêtrement*, si soit que ϕ n'est pas queue-sans-contrainte, ou bien que ξ_{n-1} est équivalente à vraie, ou ξ_{n-1} est équivalente à $\xi \wedge (x = c)$, pour un ξ, une horloge x dans $\vec{x}_j$ et une constante c. □

Une formule $\bigvee_i \phi_i$ sous la forme canonique est dite sans-enchevêtrement (respectivement fortement sans-enchevêtrement, queue-sans-contrainte, sans-π-alternation) si tout ϕ_i est sans-enchevêtrement (respectivememt fortement sans-enchevêtrement, queue-sans-contrainte, sans-π-alternation).

Nous pouvons maintenant définir deux classes de formules de DIL qui correspondent à la représentation des systèmes de contrôle et leurs exigences.

Formules de contrôle: Une *formule de contrôle* (que nous notons CDF) est une conjonction de formules de la forme

$$\Box((\uparrow\pi \wedge (\ell \sim c)) \Rightarrow \varphi)$$

où $\sim \in \{>, \geq\}$, et φ est une formule de DIL$^+$ sous la forme canonique qui est sans-enchevêtrement, et sans-π-alternation.

Formules d'exigences: une *formule d'exigences* (que nous notons RF) est une disjonction de conjonctions de formules de la forme

$$\Box(\varphi \Rightarrow \varphi')$$

où φ et φ' sont des formules de DIL$^+$ telles que φ' est sous la forme canonique et est fortement sans-enchevêtrement.

Cas particuliers: Nous appelons *horloges-contrainte-simple* toute contrainte simple qui ne contient que des horloges. Une contrainte de durée *bornée inférieurement* (que nous notons lbd-contrainte), est toute contrainte de la forme $\sum_{i=1}^{n} k_i x_i \sim d$, où $\sim \in \{>, \geq\}$, $k_i \in I\!N$, et $d \in I\!N$.

Nous appellerons alors *une formule CDF à temps pur*, toute formule de CDF telle que toutes ses contraintes de durées sont des horloges-contrainte-simple.

Plus encore, *une formule RF simple* est une conjonction de formules RF de la forme $\Box(\phi \Rightarrow \bigvee_{i=1}^{n} \phi_i)$ telle que toutes ses contraintes de durées apparaissant dans les formules $\phi, \phi_1, \cdots, \phi_n$ sont horloges-contrainte-simple, sauf au plus une, laquelle pourrait être une conjonction de lbd-contraintes et apparaître uniquement une fois dans l'une des formules $\phi_1, \cdots, \phi_n$.

7.5.2 Codage des spécifications de systèmes de contrôles et d'exigences décrites en CoD

Dans ce qui suit nous listons les formules de CoD qui ont été identifiées dans [HHF+94, Rav94] pour spécifier des systèmes de contrôle et des exigences. Puis nous montrons que ces propriétés sont exprimables dans CDF et RF respectivement.

Toutes les formules de CoD pour décrire des systèmes de contrôle et des exigences ont une des formes suivantes:

- **Entraîne vers:**
  $\phi \xrightarrow{c} \phi'$ signifiant $\Box(((\phi \wedge \ell = c); \ell > 0) \Rightarrow (\ell = c; \phi'; vraie))$.
  Ce qui a pour sens que si on a un préfixe ayant ϕ vraie durant c alors on aura ϕ' par la suite (ϕ' qui suit ϕ).
  Si par exemple nous considérons la formule suivante: $\lceil p \rceil \xrightarrow{2} \lceil \neg p \rceil$
  cette formule veut dire que p ne peut jamais être vraie plus de 2 unités de temps à la suite.

- **Transition à temps borné:**
  $\phi \xrightarrow{\leq c} \phi'$ signifiant $\forall y.\ \Box(((\phi \wedge \ell = y \wedge y \leq c); \ell > 0) \Rightarrow (\ell = y; \phi'; vraie))$.
  Tout intervalle $\leq c$ satisfaisant ϕ sera suivit par un interval satisfaisant ϕ'.
  Si nous prenons par exemple deux propositions atomiques p, q et que nous considérons la formule suivante: $\lceil p \rceil \xrightarrow{\leq c} \lceil q \rceil$
  Cette formule veut dire que durant les c unités de temps q sera vraie tant que p est vraie et q se poursuit au moins un peu au-dela p. Nous avons cette interprétation grace au fait que si un intervalle $[a, b]$ satisfait $\lceil r \rceil$ avec r une proposition atomique alors tout prefixe $[a, d]$ de l'intervalle $[a, c]$ satisfait $\lceil r \rceil$, ceci n'est pas vrai pour une formule générale ϕ.

- **Suivit par:**
  $\phi \rightarrow \phi'$ signifiant $\forall c \in \mathbb{R}.\ \phi \xrightarrow{c} \phi'$.
  Cette formule veut dire que n'importe quel intervalle satisfaisant ϕ est suivit par un intervalle satisfaisant ϕ'.

Séquencement: $\lceil \pi \rceil \rightarrow \lceil \pi \vee \bigvee_{i=1}^{n} \pi_i \rceil$.
Cette formule exige que le successeur de la phase π est toujours une des phases π_i, pour $i = 1, \cdots, n$. Si $n = 0$, cela voudrais dire que la phase π est stable à jamais (toujours stable).

Progression: $\lceil \pi \wedge \pi' \rceil \xrightarrow{c} \lceil \neg \pi \rceil$.
Cette formule spécifie une borne supérieure. La phase π est quittée lorsque π' est vraie durant c unités de temps.

Cette formule dit aussi que tout intervalle satisfaisant $\lceil\pi\rceil$ ne peut avoir un sous-intervalle de longueur supérieur à c qui satisfait π'.

Sélection: $\lceil\neg\pi\rceil; \lceil\pi \wedge \pi'\rceil \xrightarrow{\leq c} \lceil\pi \vee \bigvee_{i=1}^{n} \pi_i\rceil$.

Cette formule exprime le fait que si π' est vraie continuellement depuis la fin de la phase $\neg\pi$, alors lorsque π devient fausse avant les c unités de temps, une des phases π_i est active.
Par contre si π reste vraie au-dela de c unités de temps ou que π' se termine avant la fin de π on ne garantit rien. Lorsque $n = 0$ cette formule correspond à une phase de stabilité à temps borné qui est conditionnelle.

Synchronisation: $\lceil \bigvee_{i=1}^{n} \pi_i\rceil \xrightarrow{c} \lceil\pi'\rceil$.

Elle définit une borne inférieure; $\bigvee_{i=1}^{n} \pi_i$ entraîne que π' est vraie après les c unités de temps.

Cadrage: $\lceil\neg\pi\rceil; \lceil\pi \wedge \pi'\rceil \xrightarrow{\leq c} \lceil\pi'\rceil$. C'est une exigence de stabilité assurant que π' restera valide pour au moins c unités de temps si π est stable pour c unités de temps.

Contrainte de durée critique: $(\lceil\pi\rceil \wedge \int \pi' > d) \xrightarrow{\leq c} \lceil\neg\pi \vee \neg\pi'\rceil$.
La formule est congruente à $\Box(\lceil\pi\rceil \wedge \ell \leq c \Rightarrow \int \pi' \leq d)$ (cette congruence entre les deux formules et non triviale, elle sera énoncée et prouvée dans la prochaine proposition) et définit une contrainte de durée bornée: dans tout intervalle de longueur inférieure ou égale à c unités de temps durant lesquels π est continuellement vraie, la durée de π' n'excède pas d unités de temps.

Proposition 7.5.2 *Les formules* $\varphi_1 = (\lceil\pi\rceil \wedge \int \pi' > d) \xrightarrow{\leq c} \lceil\neg\pi \vee \neg\pi'\rceil$ *et* $\varphi_2 = \Box(\lceil\pi\rceil \wedge \ell \leq c \Rightarrow \int \pi' \leq d)$ *sont congruentes.*

Preuve 7.2 Nous allons montrer que $\varphi_1 \Leftrightarrow \varphi_2$:

- $\varphi_2 \Rightarrow \varphi_1$
  Nous commençons par ce sens car il est trivial; en effet supposons que φ_2 est vraie. Alors deux cas se présentent:
  Soit qu'on a $\neg\lceil\pi\rceil \vee \ell > c$ donc aussi bien pour $\neg\lceil\pi\rceil$ que pour $\ell > c$ on aura φ_1 qui sera trivialement vraie.
  Sinon on a $\lceil\pi\rceil \wedge \ell \leq c \wedge \int \pi' \leq d$ et là aussi à cause de $\int \pi' \leq d$ on aura φ_1 trivialement vraie.

- $\varphi_1 \Rightarrow \varphi_2$
  Ce sens est moins facile que le précédent, nous allons procéder par l'absurde. En effet nous supposons que nous avons φ_1 vraie et φ_2 fausse, nous montrerons qu'il y a une contradiction. On a par hypothèse φ_2 fausse, on a donc $\lceil\pi\rceil \wedge \ell \leq c \wedge \int \pi' > d$ sur un intervalle $[a, b]$.
  Soit $m = inf\{m' | \int_a^{m'} \pi' > d\}$, nous avons alors forcément $\int_a^m \pi = d$ car sinon il existerait $n < m$ tel que $\int_a^n \pi = d$ et donc il existe $n' \in]n, m[$ tel que $\int_a^{n'} \pi > d$ ce qui contredirais le fait que $m = inf\{m' | \int_a^{m'} \pi' > d\}$; nous avons alors forcément $\int_a^m \pi = d$. De plus φ_1 est vraie puisque $\lceil\pi\rceil$ (par hypothèse) est vraie sur $[a, b]$ donc on a forcément $\lceil\neg\pi'\rceil$ vraie sur $[m, b]$, donc $\forall r \in [m, b]$, $\int_a^r \pi' = \int_a^m \pi' = d$ ce qui contredit le fait que φ_2 est fausse.

■

Toutes les formules ci-dessus ont été proposées dans [HHF+94, Rav94] pour spécifier les systèmes de contrôle et leur exigences, sauf la dernière qui est utilisée uniquement pour spécifier les exigences. Donc, la spécification de systèmes de contrôle (respectivement de leurs exigences) est une conjonction de formules des cinq premières (respectivement toutes) classes ci-dessus. Nous pouvons montrer le fait suivant.

Proposition 7.5.3 *Pour toute spécification de système de contrôle en CoD (respectivement d'exigences en CoD) il existe une formule équivalente de CDF temps-pur (respectivement RF simple).*

Preuve 7.3 Dans ce qui suit nous montrons que toutes les formules identifiées dans [HHF+94, Rav94] pour spécifier la conception de contrôle et les exigences sont exprimables respectivement dans les fragments de CDF et RF. Toutes les formules qui ont été identifiées et faisant partie de CoD ont une des formes suivantes:

- $\phi \xrightarrow{c} \phi'$ signifiant $\Box(((\phi \wedge \ell = c); \ell > 0) \Rightarrow (\ell = c; \phi'; vraie))$
- $\phi \xrightarrow{\leq c} \phi'$ signifiant $\forall y.\ \Box(((\phi \wedge \ell = y \wedge y \leq c); \ell > 0) \Rightarrow (\ell = y; \phi'; vraie))$
- $\phi \rightarrow \phi'$ signifiant $\forall c \in I\!N.\ \phi \xrightarrow{c} \phi'$.

Nous allons lister ces formules de CoD et donner leur traductions vers DIL. Dans les formules traduites, nous écrivons $(\lceil\delta\rceil \wedge \ell \sim c); \phi$ à la place de $x.((\lceil\delta\rceil \wedge x \sim c); \phi)$. Il faut noter que toutes les formes que nous évoquons pour spécifier la concéption de systèmes de contrôle ainsi que les exigences ont été proposées dans [HHF+94, Rav94]; toutes ces formes permettent de spécifier aussi bien systèmes de contrôle que des exigences à l'exception de la dernière, laquelle est utilisée seulement pour spécifier des exigences.

Séquencement: $\lceil\pi\rceil \rightarrow \lceil\pi \vee \bigvee_{i=1}^{n} \pi_i\rceil$. Cette formule nécessite que le successeur de la phase π est toujours l'une des phases π_i, pour $i = 1, \cdots, n$. Si $n = 0$, veut dire que la phase π est stable à jamais. La formule de CDF correspondante est:

$$\Box(\uparrow\neg\pi \wedge \ell > 0 \Rightarrow (\lceil\bigvee_{i=1}^{n} \pi_i\rceil; vraie) \vee (\uparrow vraie \wedge \lceil\pi\rceil; vraie))$$

Progression: $\lceil\pi \wedge \pi'\rceil \xrightarrow{c} \lceil\neg\pi\rceil$.

La formule de CDF correspondante est:

$$\Box(\uparrow(\pi\wedge\pi')\wedge\ell > c \Rightarrow ((\lceil\pi\wedge\pi'\rceil\wedge\ell < c); \lceil\neg\pi\vee\neg\pi'\rceil; vraie) \vee ((\lceil\pi\wedge\pi'\rceil\wedge\ell = c); \lceil\neg\pi\rceil; vraie))$$

Sélection: $\lceil\neg\pi\rceil; \lceil\pi \wedge \pi'\rceil \xrightarrow{\leq c} \lceil\pi \vee \bigvee_{i=1}^{n} \pi_i\rceil$

La formule de CDF correspondante est:

$$\begin{aligned}\Box(\uparrow\pi \wedge \ell \geq c \implies\ & (\lceil\pi \wedge \neg\pi'\rceil; vraie) \vee \\ & ((\lceil\pi \wedge \pi'\rceil \wedge \ell < c); \lceil\neg(\pi \wedge \pi') \wedge (\pi \vee \bigvee_{i=1}^{n} \pi_i)\rceil; vraie) \vee \\ & (\lceil\pi \wedge \pi'\rceil \wedge \ell = c); vraie \vee \\ & (\uparrow vraie \wedge \lceil\pi\rceil; vraie))\end{aligned}$$

Synchronisation: $\lceil\bigvee_{i=1}^{n} \pi_i\rceil \xrightarrow{c} \lceil\pi'\rceil$.

La formule de CDF correspondante est:

$$\Box(\uparrow(\bigvee_{i=1}^{n} \pi_i)\wedge\ell > c \Longrightarrow ((\lceil\bigvee_{i=1}^{n} \pi_i\rceil\wedge\ell < c); \lceil\bigwedge_{i=1}^{n} \neg\pi_i\rceil; vraie) \vee ((\lceil\bigvee_{i=1}^{n} \pi_i\rceil\wedge\ell = c); \lceil\pi'\rceil; vraie))$$

Cadrage: $\lceil\neg\pi\rceil; \lceil\pi \wedge \pi'\rceil \xrightarrow{\leq c} \lceil\pi'\rceil$.

La formule de CDF correspondante est:

$$\begin{aligned}\Box(\uparrow\pi \wedge \ell \geq c \implies\ & (\lceil\pi \wedge \neg\pi'\rceil; vraie) \vee \\ & ((\lceil\pi \wedge \pi'\rceil \wedge \ell < c); \lceil\neg\pi \wedge \pi'\rceil; vraie) \vee \\ & (\lceil\pi \wedge \pi'\rceil \wedge \ell = c); vraie \vee \\ & (\uparrow true \wedge \lceil\pi\rceil; vraie))\end{aligned}$$

Contrainte de durée critique: $(\lceil\pi\rceil \wedge \int P > d) \xrightarrow{\leq c} \lceil\neg\pi \vee \neg P\rceil$.

Cette formule est équivalente à $\Box(\lceil\pi\rceil \wedge \ell \leq c \Rightarrow \int P \leq d)$.

Une traduction possible de cette formule dans RF est

$$\Box((\lceil\pi\rceil \wedge \ell \leq c) \Longrightarrow [x : P].\ x \leq d)$$

Cette formule n'est pas une formule simple de RF. Nous donnons alors une autre formule de RF qui est simple et qui exprime la même propriété (ensemble de trajectoires).

$$\Box((\lceil\pi\rceil \wedge \ell = c) \Longrightarrow [x : \neg P].\ \lceil vraie\rceil \wedge (x > c - d))$$

Dans les traductions ci-dessus, nous considérons uniquement le cas $c > 0$. Lorsque $c = 0$, toutes les formules des fragments identifiés faisant partie CoD dégénérent en *vraie*.

■

7.6 Traitement de l'exemple du brûleur à gaz

Nous rappelons que les exigences et le système de contrôle du *brûleur à gaz* sont exprimés en CoD de la manière suivante:
Req-1

$$\ell \geq 150 \Rightarrow 25 \int fuite \leq \ell$$

Des-1

$$\Box(\lceil fuite\rceil \Rightarrow \ell \leq 1)$$

Des-2

$$\Box(((\Diamond\lceil fuite\rceil);(\Diamond\lceil\neg fuite\rceil);(\Diamond\lceil fuite\rceil)) \Rightarrow \ell \geq 30)$$

Nous montrons que nous pouvons les décrire dans CDF et RF. En effet **Req-1** peut être décrite en RF par:

$$l \geq 150 \Rightarrow [x : fuite]\lceil vraie\rceil \wedge 25x \leq l$$

Mais cette formule n'est pas simple; par contre elle peut être transformée en une formule équivalente qui est simple. Cette formule est la suivante:
$\Box(l = 30 \Rightarrow ([z : \neg fuite]\lceil vraie\rceil \wedge z > 29))$

Les formules **Des-1** et **Des-2** sont décrites respectivement dans CDF par:

$$\Box(\lceil fuite\rceil \wedge l > 1 \Rightarrow \lceil fuite\rceil \wedge l \leq 1; \lceil\neg fuite\rceil; vraie)$$

$$\Box(\uparrow fuite \wedge l > 30 \Rightarrow (\lceil\neg fuite\rceil \wedge l = 30; vraie \vee vraie)$$

7.7 Le problème de vérification

Si D est l'ensemble des trajectoires du système (de contrôle) et R l'ensemble des trajectoires exprimant les exigences, le problème de vérification consiste à contrôler si $D \subseteq R$ est vraie ou non. Nous montrons dans cette partie que lorsque D et R sont représentées comme deux formules φ_D et φ_R respectivement dans CDF et RF, le problème de vérification si $[\![\varphi_D]\!] \subseteq [\![\varphi_R]\!]$ est réductible au problème du langage vide dans les automates à intégrateurs.

Dans CoD, aussi bien D que R sont exprimées en utilisant deux formules φ_D et φ_R, le problème considéré ici sera de prouver la validité de l'implication $\varphi_D \Rightarrow \varphi_R$. En général, dans CoD ou DIL une implication $\varphi \Rightarrow \varphi'$ est plus forte que l'inclusion $[\![\varphi]\!] \subseteq [\![\varphi']\!]$.
Nous pouvons voir cela en considérant par exemple la formule de CDF

$$\varphi_D = \Box(\uparrow\pi \wedge \ell > 1 \Rightarrow (\lceil \pi \wedge \pi' \rceil \wedge \ell = 1); vraie)$$

et la formule de RF

$$\varphi_R = \Box(\uparrow\pi \wedge \ell > 0 \Rightarrow \lceil \pi \wedge \pi' \rceil; vraie)$$

Alors, $[\![\varphi_D]\!]$, est l'ensemble des trajectoires telles qu'à chaque unité de temps si π change de fausse à vraie alors $\pi \wedge \pi'$ est stable pour au moins 1 unité de temps; $[\![\varphi_R]\!]$ est l'ensemble des trajectoires telles qu'à chaque unité de temps si π change de fausse à vraie alors $\pi \wedge \pi'$ est stable pour une durée non nulle. Il est clair que $[\![\varphi_D]\!] \subseteq [\![\varphi_R]\!]$. Puis d'autre part, soit ρ une trajectoire satisfaisant continuellement π et où $\neg\pi'$ est vraie partout sur l'intervalle $[0, 1[$. Alors, $(\rho, 0, 1) \models \varphi_D \wedge \neg\varphi_R$, et par conséquent, ρ ne satisfait pas l'implication $\varphi_D \Rightarrow \varphi_R$. En effet φ_D est vraie car la partie gauche de l'implication est toujours fausse vu que l'intervalle auquel nous nous intéressons n'est pas supérieur à 1, donc $\ell > 1$ est fausse. Par contre φ_R est fausse car à l'instant 0, π devient vraie et reste stable durant un temps non nul (donc la partie gauche de l'implication est vraie), mais la partie droite est fausse car $\lceil \pi \wedge \pi' \rceil$ est fausse vu que $\neg\pi'$ est vraie sur $[0, 1[$.

Puisque, l'inclusion de comportements est communément utilisée pour l'implémentation, nous nous intéressons dans ce chapitre au problème de la vérification que nous avons décrit ci-dessus, c'est à dire contrôler que $[\![\varphi_D]\!] \subseteq [\![\varphi_R]\!]$.

Nous présentons les étapes de la réduction de ce problème de vérification au problème du langage vide pour les automates à intégrateurs. Nous avons défini les langages reconnus par des automates à intégrateurs comme des ensembles de trajectoires continues à droite. Donc nous montrons que le problème de vérification $[\![\varphi_D]\!] \subseteq [\![\varphi_R]\!]$ peut être résolu en considérant uniquement les trajectoires continues à droite.

Par induction sur la structure des formules de DIL, nous pouvons montrer que l'interprétation d'une formule de DIL dans un intervalle d'une trajectoire ne dépend pas des points isolés. Donc, pour toute trajectoire ρ nous pouvons trouver une trajectoire continue à droite ρ' telle que l'interprétation de chaque formule de DIL dans chaque intervalle le long de ρ ou le long de ρ' donne le même résultat. Par conséquent, nous obtenons le résultat suivant:

Proposition 7.7.1 *Pour toutes formules φ et φ' de DIL, $[\![\varphi]\!] \subseteq [\![\varphi']\!]$ si et seulement si $[\![\varphi]\!]_{rc} \subseteq [\![\varphi']\!]_{rc}$.*

Alors, nous construisons deux automates à intégrateurs $\mathcal{H}_1$ et $\mathcal{H}_2$ tels que, $[\![\varphi_D]\!]_{rc} = \mathcal{T}(\mathcal{H}_1)$ et $\overline{[\![\varphi_R]\!]}_{rc} = \mathcal{T}(\mathcal{H}_2)$, où $\overline{[\![\varphi]\!]}_{rc}$ dénote l'ensemble des trajectoires continues à droites qui ne satisfont pas φ. Donc, nous avons $[\![\varphi_D]\!]_{rc} \subseteq [\![\varphi_R]\!]_{rc}$ si et seulement si $\mathcal{T}(\mathcal{H}_1) \cap \mathcal{T}(\mathcal{H}_2) = \emptyset$, lequel est équivalent au vide de l'automate produit de $\mathcal{H}_1$ et $\mathcal{H}_2$.

7.7.1 Formules de spécification de systèmes de contrôle

Nous présentons dans cette partie la construction d'un automate à intégrateur reconnaissant toutes les trajectoires continues à droite satisfaisant une formule de CDF. En fait, il suffit de considérer seulement les formules de CDF qui sont des conjonctions de formules de la forme

$$\Box(\uparrow\pi \wedge (\ell > c) \Rightarrow \psi) \tag{7.3}$$

Nous pouvons montrer que toute formule de CDF est équivalente à une autre formule de la forme(7.3); en effet dans le cas où nous avons la formule $\Box(\uparrow\pi \wedge (\ell \geq c) \Rightarrow \psi)$
nous pouvons la remplacer par la formule suivante qui est une conjonction de deux formules ayant la forme 7.3:
$\Box(\uparrow\pi \wedge (\ell > c) \Rightarrow \psi) \wedge$
$\Box(\uparrow\pi \wedge (\ell > c) \Rightarrow x.([\vec{x}_1 : \vec{\pi}_1].\ ((\mu_1 \wedge \xi_1); \cdots [\vec{x}_j : \vec{\pi}_j].\ ((\mu_j \wedge \xi_j); \cdots [\vec{x}_n : \vec{\pi}_n].\ (\mu_n \wedge \xi_n \wedge x \leq c)); vraie)$.

Nous commencerons par définir pour toute formule de CDF φ une formule équivalente dénotée φ^* qui servira à la construction d'un automate à intégrateurs reconnaissant $[\![\varphi]\!]_{rc}$.

Nous débuterons en considérant le cas simple d'une formule de CDF φ de la forme $\Box(\uparrow\pi \wedge (\ell > c) \Rightarrow \phi)$, où ϕ est donnée par:

$$[\vec{x}_1 : \vec{\pi}_1].\ (\underbrace{(\mu_1 \wedge \xi_1)}_{\text{Phase 1}}; \cdots [\vec{x}_n : \vec{\pi}_n].\ (\underbrace{(\mu_n \wedge \xi_n)}_{\text{Phase } n}; vraie) \cdots) \tag{7.4}$$

Soit ρ une trajectoire et $a \in \mathbb{R}_{\geq 0}$ un instant (un point du temps) où $\uparrow\pi$ est vraie. Par définition, si ρ satisfait la formule φ, alors ϕ est satisfaite par tout intervalle $[a, b]$ tel que $b > a + c$. Supposons que $c > 0$, puisque ϕ est sans-enchevêtrement, seulement les trois situations suivantes peuvent arriver:

1. La phase n commence (strictement) avant et se termine avant ou à $a + c$.
2. La phase n commence (strictement) avant et se termine (strictement) après $a+c$.
3. La phase n commence exactement à $a + c$.

En effet si l'on se souvient de la condition de sans-enchevêtrement, elle implique que pour un intervalle fixé $[a, b]$ où ϕ est vraie, il y a une séquence unique de points qui délimitent les phases 1 à $n - 1$. A partir de cela, il est clair que la phase $n - 1$ ne peut se prolonger au dela de $a + c$, puisque dans ce cas nous pouvons trouver un intervalle de longueur supérieure à c dont la fin est dans cette phase. Par conséquence, cet intervalle ne peut contenir la phase n, et par conséquent, il ne satisfait pas ϕ.
Dans le cas de $c = 0$, le même raisonnement permet de déduire que si $n > 1$, alors φ est équivalente à $fausse$.

Donc, étant donnée une formule sans-enchevêtrement ϕ (de la forme 7.4), nous définissons une formule $\phi^*_{>c}$ qui reflète les situations discutées ci-dessus.
Lorsque $c = 0$, $\phi^*_{>c} =$ si $n > 1$ alors $fausse$ sinon ϕ.
Pour $c > 0$, la formule $\phi^*_{>c}$ caractérise les trois situations évoquées ci-dessus et est définie comme suit:
Soit $\phi_t = t.\ \phi$ et supposons $n > 1$ (si $n = 0$ alors $\phi^*_{>c} = vraie$). Etant donnée une constante c, nous définissons la formule $\phi^*_{>c}$ comme une disjonction des trois formules ϕ_1, ϕ_2, et ϕ_3 qui reflètent les différentes situations possibles que nous avons énuméré ci-dessus. ϕ_1, ϕ_2, et ϕ_3 sont telles que:

1. ϕ_1 est obtenue à partir de ϕ_t en remplaçant ξ_n par $\xi_n \wedge (t \leq c)$,
2. ϕ_2 est obtenue à partir ϕ_t en remplaçant $[\vec{x}_n : \vec{\pi}_n].\ (\mu_n \wedge \xi_n; vraie)$ par $[\vec{x}_n : \vec{\pi}_n].\ (\mu_n \wedge (t = c); \mu_n \wedge \xi_n \wedge (t > c); vraie)$,
3. ϕ_3 est obtenue à partir de ϕ_t en remplaçant ξ_{n-1} par $\xi_{n-1} \wedge (t = c)$, et ξ_n par $\xi_n \wedge (t > c)$.
   (Si $n = 1$ la disjonction ϕ_3 est omise.)

Nous avons donc le résultat suivant:

Lemme 7.1 Soit ϕ une formule sans-enchevêtrement de la forme (7.4), ρ une trajectoire et $a \in \mathbb{R}_{\geq 0}$.
Alors, $\forall b > a + c.\ (\rho, a, b) \models \phi$ si et seulement si $\forall b > a + c.\ (\rho, a, b) \models \phi^*_{>c}$. ∎

Par conséquent, nous obtenons $[\![\Box(\uparrow\pi \wedge (\ell > c) \Rightarrow \phi)]\!]_{(rc)} = [\![\Box(\uparrow\pi \wedge (\ell > c) \Rightarrow \phi^*_{>c})]\!]_{(rc)}$. Nous considérons maintenant un cas plus général de formule de CDF $\varphi = \Box(\uparrow\pi \wedge (\ell > c) \Rightarrow \bigvee_{i=1}^n \phi_i)$. Si nous dénotons par φ^* la formule $\Box(\uparrow\pi \wedge (\ell > c) \Rightarrow \bigvee_{i=1}^n (\phi_i)^*_{>c})$. Cette définition se généralise directement pour la conjonctions de formules.

Nous prouvons ensuite que les formules φ et φ^* caractérisent le même ensemble de trajectoires. Il est facile de voir que chaque intervalle qui satisfait $\phi^*_{>c}$, satisfait aussi ϕ. Par conséquent $[\![\varphi^*]\!] \subseteq [\![\varphi]\!]$.
Dans ce qui suit nous donnons la preuve de l'inclusion $[\![\varphi]\!] \subseteq [\![\varphi^*]\!]$, cette inclusion n'est pas aussi triviale que la précédente.
Soient ρ une trajectoire, et $a \in I\!R_{\geq 0}$ telles que chaque intervalle $[a, b]$ de ρ avec $b > a + c$ satisfait la formule $\bigvee_{i=1}^n \phi_i$. nous montrons qu'en effet, il existe une formule $\phi \in \{\phi_1, \cdots, \phi_n\}$ qui est satisfaite par tout intervalle $[a, b]$ avec $b > a + c$.

Il est à noter que puisque nous avons une variabilité finie des trajectoires, pour tout intervalle fixé fini I d'une trajectoire ρ, il existe $\epsilon \in I\!R_{>0}$ tel que I peut être partitionné en intervalles de longueurs ϵ où toutes les fonctions caractéristiques de formules d'états par rapport à ρ sont constantes. Plus encore, puisque les contraintes de durées sont linéaires, nous pouvons montrer que pour toute contrainte ξ, une association de position $\mathcal{I}$, une formule d'association γ et $a \in I\!R_{\geq 0}$, il existe un $\epsilon' \in I\!R_{>0}$ tel que, pour tout $\theta, \theta' \in]0, \epsilon'[$, $(\rho, a, a + \theta) \models_{\mathcal{I},\gamma} \xi$ si et seulement si $(\rho, a, a + \theta') \models_{\mathcal{I},\gamma} \xi$.

Considérons un intervalle $[a, b]$ avec $b > a + c$ qui satisfait une formule $\phi \in \{\phi_1, \cdots, \phi_n\}$. Nous rappelons encore que puisque ϕ est sans-enchevêtrement, les points auxquels les variables dans $\vec{x}_1, \cdots, \vec{x}_n$ sont initialisées sont complètement déterminés par a. Il découle des observations décrites ci-dessus, qu'il existe $b > a + c$ tel que soit $\forall b' \in]a + c, b[$. $(\rho, a, b') \models \phi$ ou bien $\forall b' \in]a + c, b[$. $(\rho, a, b') \not\models \phi$.

Par conséquent si la disjonction $\bigvee_{i=1}^n \phi_i$ est satisfaite par tout intervalle $[a, b]$ avec $b > a + c$, il existe nécessairement $\phi \in \{\phi_1, \cdots, \phi_n\}$ et $d > a + c$, tels que tout intervalle $[a, b']$ avec $b' \in]a + c, d[$ satisfait ϕ. Puisque les ϕ_i sont queue-sans-contrainte, cela implique que tout intervalle $[a, b]$ de longueur $> c$ satisfait ϕ. En utilisant le lemme 7.1, nous en déduisant que tout intervalle $[a, b]$ de longueur $> c$ satisfait $\phi^*_{>c}$. D'où nous obtenons que $\Box(\uparrow\pi \wedge (\ell > c) \Rightarrow \bigvee_{i=1}^n \phi_i)$ et $\Box(\uparrow\pi \wedge (\ell > c) \Rightarrow \bigvee_{i=1}^n (\phi_i)^*_{>c})$ sont équivalentes. Finalement, nous pouvons voir que pour toute formule de DHÊ, $[\![\varphi \wedge \varphi']\!] = [\![\varphi]\!] \cap [\![\varphi']\!]$. Donc, nous obtenons le résultat suivant:

Lemme 7.2 Pour toute formule φ de CDF, $[\![\varphi]\!]_{(rc)} = [\![\varphi^*]\!]_{(rc)}$. ■

Considérons une formule de CDF avec $\varphi = \Box(\uparrow\pi \wedge (\ell > c) \Rightarrow \bigvee_{i=1}^n \phi_i)$,
$\varphi^* = \Box(\uparrow\pi \wedge (\ell > c) \Rightarrow \bigvee_{i=1}^m \phi'_i)$ et une trajectoire ρ. Par le Lemme 7.2, prouver que ρ satisfait φ est équivalent à prouver que ρ satisfait φ^*. Comme nous l'argumentons dans

la justification de ce lemme si ρ satisfait φ, pour tout point a où $\uparrow\pi$ est vraie, il existe une formule $\phi \in \{\phi_1, \cdots, \phi_n\}$ et un point $d > a + c$, tels que $\phi^*_{>c}$ est satisfaite par tout intervalle $[a, b]$ avec $b \in]a + c, d[$. Nous rappelons que $\phi^*_{>c}$ peut être une disjonction de trois formules (lorsque $c > 0$). Il faut noter dans ce cas, que ces formules sont en exclusion mutuelle. Donc, nous en déduisons qu'afin de vérifier qu'une trajectoire ρ satisfait φ, nous devons trouver pour tout point a où $\uparrow\pi$ est vraie, une formule $\phi' \in \{\phi'_1, \cdots, \phi'_m\}$ et $d > a + c$, telles que ϕ' est satisfaite par tout intervalle $[a, b]$ avec $b \in]a + c, d[$. Maintenant, pour une formule fixée ϕ' et un point d, nous devons contrôler la vérité de ϕ pour une infinité d'intervalles, tous les intervalles $[a, b]$ avec $b \in]a + c, d[$. La solution de ce problème est donnée par le prochain lemme qui énonce qu'il suffit de considérer seulement l'intervalle $[a, d]$ et de contrôler une propriété plus forte qui est décrite dans ce lemme.

Lemme 7.3 Soient φ une formule de CDF avec $\varphi^* = \Box(\uparrow\pi \wedge (\ell > c) \Rightarrow \bigvee_{i=1}^{m} \phi_i)$, et ρ une trajectoire continue à droite. Alors ρ satisfait φ^* si et seulement si pour tout $a, b \in I\!R_{\geq 0}$ tels que $(\rho, a, b) \models \uparrow\pi$, il existe $d \in I\!R_{\geq 0}$ avec $d > a + c$ et $\phi \in \{\phi_1, \cdots, \phi_m\}$ tels que

- si $\phi = t.\ [\vec{x}_1 : \vec{\pi}_1].\ (\cdots [\vec{x}_n : \vec{\pi}_n].\ (\mu_n \wedge \xi_n \wedge (t \leq c); vraie) \cdots)$, alors il existe $a_1, \cdots, a_{n+1} \in I\!R_{\geq 0}$ avec $a = a_1$, une association de positions $\mathcal{I}$ satisfaisant $\mathcal{I}(t) = a_1$, $\forall j \in \{1, \cdots, n\}\ \forall x \in \vec{x}_j.\ \mathcal{I}(x) = a_j$ et une association de formules d'états γ satisfaisant $\forall j \in \{1, \cdots, n\}.\ \gamma(\vec{x}_j) = \vec{\pi}_j$ tels que

  1. $\forall j \in \{1, \cdots, n\}.\ (\rho, a_j, a_{j+1}) \models_{\mathcal{I},\gamma} \mu_j \wedge \xi_j$,
  2. $(\rho, a_n, a_{n+1}) \models_{\mathcal{I},\gamma} t \leq c$,

- si $\phi = t.\ [\vec{x}_1 : \vec{\pi}_1].\ (\cdots [\vec{x}_n : \vec{\pi}_n].\ (\mu_n \wedge (t = c); \mu_n \wedge \xi_n \wedge (t > c); vraie) \cdots)$, alors il existe $a_1, \cdots, a_{n+1} \in I\!R_{\geq 0}$ avec $a = a_1$, $\mathcal{I}$ satisfaisant $\mathcal{I}(t) = a_1$, $\forall j \in \{1, \cdots, n\}.\ \forall x \in \vec{x}_j.\ \mathcal{I}(x) = a_j$ γ satisfaisant $\forall j \in \{1, \cdots, n\}.\ \gamma(\vec{x}_j) = \vec{\pi}_j$ tels que

  1. $\forall j \in \{1, \cdots, n-1\}.\ (\rho, a_j, a_{j+1}) \models_{\mathcal{I},\gamma} \mu_j \wedge \xi_j$,
  2. $(\rho, a_n, a_{n+1}) \models_{\mathcal{I},\gamma} \mu_n \wedge (t = c)$
  3. $\forall u \in]a_{n+1}, d[.\ (\rho, a_{n+1}, u) \models_{\mathcal{I},\gamma} \mu_n \wedge \xi_n$,

- si $\phi = t.\ [\vec{x}_1 : \vec{\pi}_1].\ (\cdots [\vec{x}_{n-1} : \vec{\pi}_{n-1}].\ (\mu_{n-1} \wedge \xi_{n-1} \wedge (t = c);$
  $[\vec{x}_n : \vec{\pi}_n].\ \mu_n \wedge \xi_n \wedge (t > c); vraie) \cdots)$, alors il existe $a_1, \cdots, a_n \in I\!R_{\geq 0}$ avec $a = a_1$, $\mathcal{I}$ satisfaisant $\mathcal{I}(t) = a_1$, $\forall j \in \{1, \cdots, n\}.\ \forall x \in \vec{x}_j.\ \mathcal{I}(x) = a_j$ et γ satisfaisant $\forall j \in \{1, \cdots, n\}$
  $\gamma(\vec{x}_j) = \vec{\pi}_j$ tels que

  1. $\forall j \in \{1, \cdots, n-1\}.\ (\rho, a_j, a_{j+1}) \models_{\mathcal{I},\gamma} \mu_j \wedge \xi_j$,

2. $(\rho, a_{n-1}, a_n) \models_{\mathcal{I},\gamma} t = c$ et
3. $\forall u \in]a_n, d[.\ (\rho, a_n, u) \models_{\mathcal{I},\gamma} \mu_n \wedge \xi_n$,

■

Puis dans ce qui va suivre nous montrons en utilisant le Lemme 7.3 que pour une formule donnée de CDF φ avec $\varphi^* = \Box(\uparrow\pi \wedge (\ell > c) \Rightarrow \bigvee_{i=1}^{m} \phi_i)$, nous pouvons construire un automate $\mathcal{H}$ qui reconnais $[\![\varphi]\!]_{rc}$. Pour des raisons de présentation, nous supposons que les ϕ_i ne contiennent pas de sous-formules de la forme $\uparrow\delta$. Le cas général n'est guère plus difficile, mais il n'est pas très pratique à présenter. Cet automate consiste en deux parties. La première, que l'on appelle *partie d'attente*, trouve le prochain point auquel $\uparrow\pi$ est vraie. Lorsqu'un tel point du temps a (instant) est détecté, l'exécution de $\mathcal{H}$ passe dans la seconde partie de l'automate, appelée la *partie recherche non déterministe*, où il trouve de manière non déterministe $\phi \in \{\phi_1, \cdots, \phi_m\}$ et $d > a + c$, tel qu'une des situations décrits par le Lemme 7.3 est appliquée à l'intervalle $[a, d]$ et la formule ϕ. La condition de sans-π-alternation assure que durant cette procédure, il n'y a pas d'autres points où l'on a $\uparrow\pi$ vraie. Lorsque cette procédure termine, cette exécution revient à la partie d'attente de $\mathcal{H}$, sauf si un nouveau point où $\uparrow\pi$ est vraie est immédiatement détecté, dans ce cas il réentre directement dans la partie recherche non déterministe. Les nœuds initiaux de $\mathcal{H}$ sont les nœuds $\neg\pi$-nœud de la partie d'attente et les nœuds cibles des transitions π-dangereuses de la partie recherche non déterministe. Tous les nœuds sont accepteurs, par conséquent $\mathcal{H}$ est un graphe à intégrateurs.

Afin de présenter dans les détails la construction de $\mathcal{H}$, nous avons besoin d'introduire quelques notions. Etant donnée une formule d'état π nous appelons un nœud q un π-nœud, si $\Pi(q) \models \pi$ et nous dirons qu'une transition est π-dangereuse si c'est une transition partant d'un nœud $\neg\pi$-nœud vers un nœud π-nœud.

La partie d'attente contient pour chaque état $s \in 2^{\mathcal{P}}$ un nœud étiqueté par s et un invariant égal à vraie. Toutes les transitions entre les nœuds sont permises sauf les transitions π-dangereuse. Les pentes d'évolution des variables dans cette partie de l'automate ne sont pas importantes.

Les transitions qui sont π-dangereuse vont de la partie d'attente vers la partie recherche non déterministe. La structure de la partie recherche non déterministe est induite par les formules ϕ_i apparaissant dans φ^*. Considérons par exemple la formule suivante:

$$t.\ [\vec{x}_1 : \vec{\pi}_1].\ \underbrace{((\lceil\delta_1\rceil \wedge \xi_1);}_{\text{Phase 1}} \cdots [\vec{x}_n : \vec{\pi}_n].\ \underbrace{(\lceil\delta_n\rceil \wedge (t = c);}_{\text{Phase } n} \underbrace{\lceil\delta_n\rceil \wedge \xi_n \wedge (t > c);}_{\text{Phase } n+1} \mathit{vraie}) \cdots$$

La partie recherche non déterministe associée à cette formule est représentée dans la figure 7.1. Le graphe qui la représente est construit de la manière suivante: nous associons à chaque phase i, avec $i \in \{1, \cdots, n\}$, un graphe complet où les nœuds sont étiquetés par tous les états possible satisfaisant la formule d'état correspondante. A la phase $n+1$, nous associons deux tels graphes étiquetés par soit $\delta_n \wedge \pi$ ou bien $\delta_n \wedge \neg\pi$. Tous les nœuds de ces deux graphes sont connectés.

Chaque nœud dans le graphe associé avec la phase i est en relation avec tous les nœuds de la phase $i+1$ par une transition qui est gardée par la contrainte correspondant à la phase i et qui remet à zéro les variables introduites pour la phase $i+1$. La pente d'évolution de chaque variable x qui est liée dans une formule d'état π, est 0 (respectivement 1) dans tout $\neg\pi$-nœud (respectivement π-nœud). Les nœuds dans les graphes correspondant à la phase $n+1$ ont ξ_n comme invariant; tous les autres nœuds ont $t < c$ comme invariant.

La partie recherche non déterministe est connectée avec la partie d'attente comme suit: chaque nœud du graphe étiqueté par $\delta_n \wedge \pi$ a une transition vers tous les nœuds de la partie d'attente de l'automate; les nœuds du graphe qui sont étiquetés par $\delta_n \wedge \neg\pi$ sont connectés aux nœuds $\neg\pi$-nœuds de la partie d'attente et tous les nœuds du graphe de la phase 1.

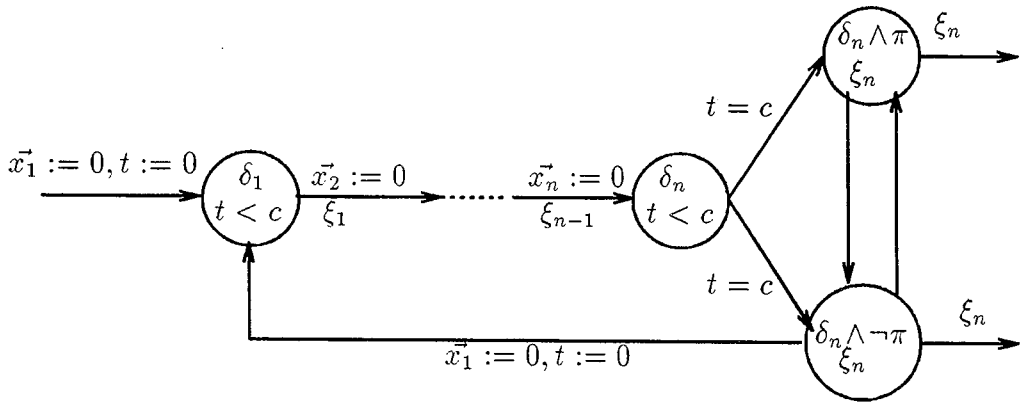


Figure 7.1: Recherche non déterministe

Nous avons expliqué la construction de la partie recherche non déterministe correspondant à une des formes possibles que ϕ peut prendre. D'autres formes peuvent être considérées de manière similaires. Nous obtenons alors le résultat suivant:

Théorème 7.1 Pour toute formule de φ de CDF, nous pouvons construire un graphe à intégrateur $\mathcal{H}$ tel que $\mathcal{T}(\mathcal{H}) = [\![\varphi]\!]_{rc}$. □

7.7.2 Formules d'exigence

Nous montrons dans cette partie que pour toute formule φ de RF, nous pouvons construire un automate à intégrateur reconnaissant toutes les trajectoires continues à droite qui ne satisfont pas φ.

Pour commencer il faut observer qu'en général, $[\![\phi_1 \vee \phi_2]\!]_{(rc)}$ est différente de $[\![\phi_1]\!]_{(rc)} \cup [\![\phi_2]\!]_{(rc)}$. Par exemple, $[\![\ell = 0]\!]_{(rc)} = [\![\lceil vraie \rceil]\!]_{(rc)} = \emptyset$ tandis que $[\![(\ell = 0) \vee \lceil vraie \rceil]\!]_{(rc)}$ est l'ensemble de toutes les trajectoires (continues à droites), en effet car dans le cas de la disjonction nous avons tous les intervalles (ponctuelle ou non). Par conséquent, $\overline{[\![\phi_1 \vee \phi_2]\!]}_{(rc)}$ est en général différente de $\overline{[\![\phi_1]\!]}_{(rc)} \cap \overline{[\![\phi_2]\!]}_{(rc)}$. Néanmoins, puisque pour toutes formules φ et φ' de DIL, $[\![(\Box\varphi) \vee (\Box\varphi')]\!]_{(rc)} = [\![\Box\varphi)]\!]_{(rc)} \cup [\![\Box\varphi)]\!]_{(rc)}$, nous pouvons prouver le fait suivant:

Lemme 7.4 Pour toute formule de DIL
$\varphi = \bigvee_{i=1}^{n} \bigwedge_{j=1}^{m_i} \Box\varphi_i^j$, $\overline{[\![\varphi]\!]}_{rc} = \bigcap_{i=1}^{n} \bigcup_{j=1}^{m_i} \overline{[\![\Box\varphi_i^j]\!]}_{rc}$. ■

A présent nous donnons le lemme qui est la clef de notre construction. Dans ce lemme nous énonçons que la classe des formules fortement sans-enchevêtrement est fermée par complémentation.

Lemme 7.5 pour toute formule φ de DIL$^+$ fortement sans-enchevêtrement, il existe une formule de DIL$^+$ fortement sans-enchevêtrement qui est congruente à $\neg\varphi$. ■

Preuve 7.4 Dans cette partie nous prouvons le lemme 7.5, qui représente l'observation clef qui permet la construction d'un automate à intégrateur reconnaissant l'ensemble des trajectoires continues à droite ne satisfaisant pas une formule φ de RF donnée.

Soit ϕ une formule fortement sans-enchevêtrement. Pour des raisons de clartés, nous supposons qu'il n'y a pas de formule de la forme $\uparrow\pi$ apparant dans ϕ. Le cas général est similaire. Pour prouver le lemme, il suffit de le montrer vrai pour ϕ, avec ϕ sous l'une des formes suivantes:

1. $[\vec{x}_1 : \vec{\pi}_1].\ ((\lceil\delta_1\rceil \wedge \xi_1); \cdots; [\vec{x}_n : \vec{\pi}_n].\ (\lceil\delta_n\rceil \wedge \xi_n) \cdots)$,
2. $[\vec{x}_1 : \vec{\pi}_1].\ ((\lceil\delta_1\rceil \wedge \xi_1); \cdots; [\vec{x}_n : \vec{\pi}_n].\ (\lceil\delta_n\rceil; vraie) \cdots)$,
3. $[\vec{x}_1 : \vec{\pi}_1].\ ((\lceil\delta_1\rceil \wedge \xi_1); \cdots; [\vec{x}_n : \vec{\pi}_n].\ ((\lceil\delta_n\rceil \wedge \xi_n \wedge (\ell = c)); vraie) \cdots)$.

Nous considérons trois cas.

1. Cas ϕ est de la forme $[\vec{x}_1 : \vec{\pi}_1].\ ((\lceil\delta_1\rceil \wedge \xi_1); \cdots ; [\vec{x}_n : \vec{\pi}_n].\ (\lceil\delta_n\rceil \wedge \xi_n)\cdots)$.

La formule $\psi_1 \vee \psi_2 \vee \psi_3$ de DIL$^+$, où les ψ_i's sont définis ci-dessous est congruente à $\neg\phi$.

$$\begin{aligned}
\psi_1 = \ & \bigvee_{i=1}^{n-1} (\lceil\delta_1\rceil; \cdots ; \lceil\delta_i\rceil; \lceil\neg\delta_i \wedge \neg\delta_{i+1}\rceil; vraie) \vee \\
& \lceil\neg\delta_1\rceil; vraie \vee \\
& \lceil\delta_1\rceil; \cdots ; \lceil\delta_n\rceil; \lceil\neg\delta_n\rceil; vraie \\
\psi_2 = \ & \bigvee_{i=1}^{n-1} [\vec{x}_1 : \vec{\pi}_1].\ (\lceil\delta_1\rceil; \cdots ; [\vec{x}_{i-1} : \vec{\pi}_{i-1}].\ (\lceil\delta_{i-1}\rceil; \\
& [\vec{x}_i : \vec{\pi}_i].\ ((\lceil\delta_i\rceil \wedge \neg\xi_i); \lceil\delta_{i+1}\rceil; vraie))\cdots) \\
\psi_3 = \ & [\vec{x}_1 : \vec{\pi}_1].\ (\lceil\delta_1\rceil; \cdots ; [\vec{x}_{n-1} : \vec{\pi}_{n-1}].\ (\lceil\delta_{n-1}\rceil; [\vec{x}_n : \vec{\pi}_n].\ (\lceil\delta_n\rceil \wedge \neg\xi_n)))
\end{aligned}$$

2. Cas ϕ est de la forme $[\vec{x}_1 : \vec{\pi}_1].\ ((\lceil\delta_1\rceil \wedge \xi_1); \cdots ; [\vec{x}_n : \vec{\pi}_n].\ (\lceil\delta_n\rceil; vraie)\cdots)$.

La formule $\psi_1' \vee \psi_2$ de DIL$^+$ où ψ_1' est défini comme ci-dessous et ψ_2 comme ci-dessus, est congruente à $\neg\phi$.

$$\begin{aligned}
\psi_1' = \ & \bigvee_{i=1}^{n-1} (\lceil\delta_1\rceil; \cdots ; \lceil\delta_i\rceil; \lceil\neg\delta_i \wedge \neg\delta_{i+1}\rceil; vraie) \vee \\
& \lceil\neg\delta_1\rceil; vraie
\end{aligned}$$

3. Cas ϕ est de la forme
$[\vec{x}_1 : \vec{\pi}_1].\ ((\lceil\delta_1\rceil \wedge \xi_1); \cdots ; [\vec{x}_n : \vec{\pi}_n].\ ((\lceil\delta_n\rceil \wedge \xi_n \wedge (\ell = c)); vraie)\cdots)$.

La formule $\psi_1' \vee \psi_2 \vee \psi_3' \vee \psi_4$ de DIL$^+$, où ψ_1' et ψ_2 sont comme ci-dessus, ψ_3' et ψ_4 sont définis comme ci-dessous, est congruente à $\neg\phi$.

$$\begin{aligned}
\psi_3' = \ & \lceil\delta_1\rceil; \cdots ; \lceil\delta_{n-1}\rceil; (\lceil\delta_n\rceil \wedge (\ell < c)); \lceil\neg\delta_n\rceil; vraie \\
\psi_4 = \ & [\vec{x}_1 : \vec{\pi}_1].\ (\lceil\delta_1\rceil; \cdots ; [\vec{x}_{n-1} : \vec{\pi}_{n-1}].\ (\lceil\delta_{n-1}\rceil; \\
& [\vec{x}_n : \vec{\pi}_n].\ (\lceil\delta_n\rceil \wedge (\ell = c) \wedge \neg\xi_n); vraie))
\end{aligned}$$

■

Notre construction est basée sur le fait qu'étant donnée une formule de DIL$^+$, nous pouvons définir un automate à intégrateurs qui pour toute trajectoire ρ devine de manière non déterministe le commencement d'un intervalle dans ρ qui satisfait φ. Lorsqu'un tel intervalle est trouvé, l'automate passe dans un nœud accepteur.
Nous obtenons alors le résultat suivant:

Proposition 7.7.2 *Pour toute formule φ de DIL$^+$, nous pouvons construire un AI (automate à intégrateur) $\mathcal{H}$ tel que $\exists a, b \in \mathbb{R}_{\geq 0}.\ (\rho, a, b) \models \varphi$.*

Pour illustrer comment l'automate $\mathcal{H}$ est construit, nous commençons par considérer une formule ϕ qui est définie comme

$$[\vec{x}_1 : \vec{\pi}_1].\ ((\lceil \delta_1 \rceil \wedge \xi_1); \cdots [\vec{x}_n : \vec{\pi}_n].\ (\lceil \delta_n \rceil \wedge \xi_n)) \cdots)$$

L'automate correspondant $\mathcal{H}$ est représenté dans la figure 7.2. Il a un nœud accepteur, marqué par un double cercle. Ses états initiaux sont marqués par des flèches verticales. Pour tous les nœuds de $\mathcal{H}$, nous avons $\Gamma(q) = vraie$ et pour toute variable x qui est liée à π, $\partial(q, x) = 1$ si $\Pi(q) \models \pi$ sinon $\partial(q, x) = 0$.

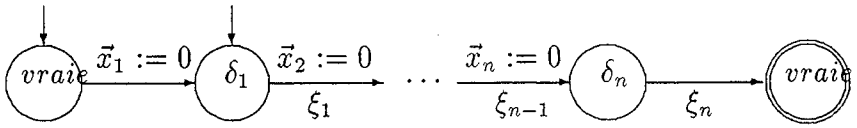


Figure 7.2: L'automate de l'ensemble des trajectoires ayant un intervalle satisfaisant ϕ

Soit $\varphi = \bigvee_{i=1}^{n} \bigwedge_{j=1}^{m_i} \Box(\varphi_{(i,j)} \Rightarrow \varphi'_{(i,j)})$ une formule de RF. Par le Lemme 7.4, $\overline{[\![\varphi]\!]}_{rc} = \bigcap_{i=1}^{n} \bigcup_{j=1}^{m_i} \mathcal{T}_i^j$, où les $\mathcal{T}_i^j$ sont les ensembles des trajectoires continues à droite ρ, telles que $\exists a, b \in \mathbb{R}_{\geq 0}.\ (\rho, a, b) \models \varphi_{(i,j)} \wedge \neg\varphi'_{(i,j)}$. Plus encore, puisque les $\varphi_{(i,j)}$ sont des formules de DIL$^+$ et par le lemme 7.5, les $\neg\varphi'_{(i,j)}$ sont congruentes aux formules de DIL$^+$. Donc toutes les formules $\varphi_{(i,j)} \wedge \neg\varphi'_{(i,j)}$ sont congruentes aux formules de DIL$^+$. Par conséquent, en utilisant la proposition 7.7.2, nous obtenons le résultat suivant:

Théorème 7.2 Pour toute formule φ de RF, nous pouvons construire un automate à intégrateur $\mathcal{H}$ tel que $\mathcal{T}(\mathcal{H}) = \overline{[\![\varphi]\!]}_{rc}$. □

7.7.3 Discussion

Les constructions que nous avons présenté précédemment sont basées sur certaines conditions que nous avons considéré dans les définitions des formules de CDF et de RF. Nous montrons dans ce qui suit que sans ces conditions, les constructions que nous avons présenté sont en général impossibles. En effet, nous exhibons des exemples de formules qui violent ces conditions et telles que pour ces formules nous avons besoin

d'automates ayant un nombre infini de variables. Les exemples que nous donnons sont inspirés par un exemple donné dans [AD94] pour montrer que la classe des langages temporisés réguliers n'est pas fermée par complémentation.

Commençons avec le cas des formules de CDF. Considérons la formule suivante:

$$\varphi_1 = \Box(\uparrow\pi \wedge (\ell > 1) \Rightarrow (\lceil vraie \rceil \wedge \ell = 1); \lceil \neg\pi \rceil; vraie).$$

Clairement, cette formule n'est pas dans CDF puisqu'elle n'est pas sans-π-alternation. Elle exprime la propriété que lorsque π change de fausse à vraie, alors après exactement une unité de temps, π est fausse. Un automate qui reconnait cette propriété doit être capable de garder la trace de tous les instants (points du temps) où $\uparrow\pi$ est vraie dans la dernière unité de temps. Puisque dans un intervalle de longueur 1 unité de temps, le nombre d'instants où $\uparrow\pi$ est vraie est non borné, un nombre infini d'horloges est alors nécessaire.

La prochaine formule que nous considérons est

$$\varphi_2 = \Box(\uparrow vraie \wedge (\ell > 1) \Rightarrow (\lceil vraie \rceil; x.\ (\lceil P \rceil; x = 1)) \vee (\lceil vraie \rceil; \lceil \neg P \rceil)).$$

Alors, φ_2 n'est ni sans-enchevêtrement ni queue-sans-contrainte. Une trajectoire ρ satisfait φ_2, si pour tout $b > 1$ avec P presque partout vraie dans un voisinage précédant b, il y a un intervalle commençant à $b - 1$ qui satisfait $\lceil P \rceil$. Afin de contrôler cette propriété l'automate doit enregistrer tous les intervalles (maximaux) dans la dernière unité de temps qui satisfont $\lceil P \rceil$. Puisque le nombre d'intervalles est non borné, donc un automate avec un nombre fini de variables ne peut pas reconnaître $[\![\varphi_2]\!]_{rc}$.

A partir de ces exemples, nous pouvons en déduire que pour les formules de spécification la condition de π-alternation est nécessaire en général, aussi bien que la combinaison de formules sans-enchevêtrement et de formules queue-sans-contrainte. Par contre, il n'est pas clair si chacune des ces dernières conditions est nécessaire par elle-même.

Considérons le cas des formule de RF. Considérons la formule

$$\varphi_3 = \Box(\ell \geq 2 \Rightarrow vraie; x.\ (\lceil \pi \rceil; x = 1; \lceil \pi \rceil; vraie)).$$

Cette formule n'est pas dans RF puisque la partie droite de son implication n'est pas sans-enchevêtrement. La formule φ_3 exprime la propriété que dans chaque intervalle de longueur supérieure ou égale à 2 il existe deux intervalles satisfaisant $\lceil \pi \rceil$ et qui soient séparés par une unité de temps. Donc pour accepter une trajectoire ρ qui falsifie φ_3, un automate doit trouver un intervalle $[a, b]$ avec $b - a \geq 2$ tel que pour chaque intervalle $[c, d] \subseteq [a, a + 1]$ satisfaisant $\lceil \pi \rceil$, l'intervalle $[c + 1, d + 1]$ satisfait $\lceil \neg\pi \rceil$. Cela implique qu'un tel automate est capable d'enregistrer tous les sous-intervalles

de $[a, a+1]$ qui satisfont $[\pi]$. Puisque le nombre de ces intervalles est non borné, il nécessite par conséquent un nombre infini d'horloges.

Donc la condition de sans-enchevêtrement dans la partie droite des implications dans les formules de RF est nécessaire. Pour la construction que nous présentons, nous avons imposé une forte condition sur ces formules, cette condition est dite fortement sans-enchevêtrement. Notre motivation est de donner un fragment de DIL^+ qui est fermé par complémentation. De cette manière nous pouvons alors réduire le problème de la construction d'un automate dont les trajectoires falsifient la formule de RF, au problème de la construction d'un automate de trajectoires ayant un intervalle qui satisfait une formule de DIL^+, lequel automate peut être construit facilement. Nous pouvons donner une construction sans passer par cette étape intermédiaire, elle permet de considérer des formules d'exigences où nous n'imposons que la condition de sans-enchevêtrement.

7.8 Résultats de décidabilité

Dans cette partie nous essayons de voir si les problèmes de vérification qui nous intéressent sont décidables ou non. Les problèmes sur lesquels notre intérêt s'est porté concernent la vérification de systèmes hybrides décrits à l'aide de formules de CDF par rapport à des spécifications définies par des formules de RF. Nous considérons aussi les problèmes de satisfaisabilité et de validité des formules de RF.

Nous avons montré dans la section précédente que le problème de vérification de formules de CDF par rapport à des formules de RF, peut être réduit au problème du langage vide d'un automate à intégrateurs. Les automates que nous considérons ont comme condition d'acceptance la 1-acceptance, par conséquent le problème du langage vide sur ces automates est équivalent au complément du problème d'atteignabilité des graphes à intégrateurs. Le problème d'atteignabilité est en général indécidable [Cer92, HK94, Kop95a], néanmoins, quelques sous-classes de graphes à intégrateurs ont été identifiées pour lesquelles ce problème est décidable [ACD93, ACH93, KPSY93, BER94d, BR95]. En utilisant certains de ces résultats, nous montrons que le problème de la vérification de formules de CDF par rapport à des formules simples de $\overline{\text{RF}}$ est décidable. Cela implique donc que le problème de la vérification de systèmes de contrôle de CoD par rapport à des exigences de CoD est décidable.

Plus encore, nous montrons que le problème de la validité de formules de RF est décidable alors que le problème de satisfaisabilité de telles formules n'est même pas récursivement énumerable.

Commençons par considérer le problème de la vérification. Tout d'abord par observer que pour toute formule φ à temps pure de CDF φ, le graphe à intégrateurs

reconnaîssant $[\![\varphi]\!]_{rc}$ obtenu par la construction de la section 7.7.1 est en fait un graphe temporisé.
Plus encore, pour toute formule simple φ de RF, la construction décrite dans la partie 7.7.2 donne un automate à intégrateur simple qui reconnaît les trajectoires falsifiant φ.
En effet, soit $\varphi = \bigwedge_{i=1}^{n} \Box(\psi_i \Rightarrow \psi_i')$ une formule simple de RF. L'automate que nous construisons est l'union d'automates, tels que chacun d'eux reconnaît l'ensemble des trajectoires pour lesquelles il existe un intervalle fini satisfaisant $\psi_i \wedge \neg\psi_i'$. Un examen des formules sous la forme canonique qui sont équivalente à $\psi_i \wedge \neg\psi_i'$ (voir en particulier la complémentation de formules fortement sans-enchevêtrement dans la preuve 7.4), montre que ces formules sont disjointes là où il y a au plus une contrainte qui n'est pas une horloges-contrainte-simple dans chacune de leurs disjonctions. Plus encore, cette contrainte est de la forme $0 \sim \sum_{i=1}^{n} k_i x_i \sim d$, avec $\sim \in \{<, \leq\}$, $k_i \in I\!N$ et $d \in I\!N$. Par conséquent, les automates correspondants obtenus par notre construction sont des automates à intégrateurs simples.

Nous pouvons voir aussi que le produit de graphes temporisés avec un automate à intégrateurs simple est un automate à intégrateurs simple. Donc le problème de la vérification de formules de CDF à temps pur et plus génerallement pour conceptions décrites à l'aide d'un graphe temporisé par rapport à des formules simples de RF, est réductible au problème d'atteignabilité pour un graphe à intégrateurs simple. Nous pouvons alors déduire à partir du résultat énoncé dans [ACH93] que ce problème d'atteignabilité est décidable. Donc, nous obtenons le résultat de décidabilité suivant.

Theorem 7.8.1 *Le problème de vérification pour les formules de CDF à temps pur (de manière plus générale pour les graphes temporisés) par rapport aux formules simples de RF, est décidable.*

D'après la proposition 7.5.3 et le theorem 7.8.1, nous obtenons le résultat de décidabilité suivant.

Corollaire 7.8.1 *Le problème de la vérification pour les formules de conception de systèmes de contrôle en CoD par rapport aux formules d'exigence décrites en CoD est décidable.*

Nous considérons les problèmes de validité et de satisfaisabilité des formules de RF. Puisqu'une formule est valide si et seulement si elle est satisfaite par toutes les trajectoires continues à droite et puisque l'ensemble de toutes les trajectoires continues à droite peut être défini par un graphe temporisé, nous obtenons alors comme conséquence du théorème 7.8.1 le résultat suivant.

Corollaire 7.8.2 *Le problème de validité de formules simples de RF et par conséquent des formules de spécification de propriétés d'exigences de CoD, est décidable.*

Nous pouvons prouver que le problème de validité de toutes les formules de RF est décidable par réduction à un problème de programmation linéaire. D'autre part, nous pouvons montrer que le problème de divergence (complémentaire du problème de l'arrêt) d'une machine à 2 compteurs déterministe est réductible au problème de satisfaisabilité de formules de RF Nous avons donc le résultat suivant:

Theorem 7.8.2 *Le problème de validité des formules de RF est décidable alors que leur problème de satisfaisabilité est indécidable (Π_1^0-hard).*

Preuve 7.5 Soit M une machine à deux compteurs déterministe avec deux compteurs C_1, C_2 et n instructions de programme numérotés de 0 à $n-1$. Nous supposons que les instructions numérotées par 0 et $n-1$ sont respectivement l'instruction de début et l'instruction de fin. Nous supposons aussi que l'instruction de début est exécutée une seule fois.
Une configuration de la machine est représentée par un triplet $\langle i, c_1, c_2\rangle$, où i est le nombre d'instructions qui vont être exécutées prochaînement, c_1 et c_2 sont respectivement les valeurs des compteurs C_1 et C_2. Une exécution de M est une séquence de configurations. Nous supposons sans pour autant que soit une restriction que lorsque M exécute son instruction fin, elle entre dans une boucle infini.

Nous définissons une formule φ de RF laquelle est satisfaisable si et seulement si la machine M est divergente, c'est à dire que la machine n'exécute jamais l'instruction $n-1$. Pour cela, nous représentons les exécutions de M comme des trajectoires partitionnées en intervalles délimités par la valeur vraie de la formule $\uparrow R$ pour une proposition spéciale R. Nous utilisons des formules d'état exclusives $\pi_1, \cdots, \pi_n$ pour indiquer l'instruction sous considération, nous utilisons aussi deux propositions spéciales Q_1 et Q_2 dont les durées code les valeurs des deux compteurs de M. Nous réalisons cela de la manière suivante:
Une configuration $\langle i, c_1, c_2\rangle$ est codée par un intervalle qui satisfait la formule

$$\underbrace{\uparrow R \wedge \lceil R \wedge \pi_i \rceil}_{\text{Phase 1}}; \underbrace{\lceil \neg R \wedge \bigwedge_{j=1}^{n} \neg \pi_j \rceil}_{\text{Phase 2}}; \uparrow R$$

et telle que c_1 (respectivement c_2) est égale à la durée de Q_1 (respectivement Q_2) dans la Phase 2.

Donc, la formule φ est la conjonction de cinq formules φ_{init}, φ_{excl}, φ_{inv}, φ_{prog} et φ_{div}, qui sont présentées ci-dessous.

La formule φ_{init} exprime le fait que la première instruction exécutée est l'instruction 0 et que initialement les compteurs sont à 0. Par conséquent, la formule φ_{init} est la conjonction de deux formules φ^1_{init} et φ^2_{init}, où φ^1_{init} est donnée par

$$\Box((\uparrow vraie \wedge \ell > 0) \Rightarrow (\lceil \pi_0 \wedge R \rceil; vraie))$$

et φ^2_{init} est donnée par

$$\Box(\lceil \pi_0 \wedge R \rceil; \lceil \neg R \rceil; \lceil R \rceil \Rightarrow \lceil \pi_0 \wedge R \rceil; [x_i : Q_i]^2_{i=1}. (\lceil \neg R \rceil \wedge x_1 = 0 \wedge x_2 = 0; \lceil R \rceil))$$

La formule φ_{excl} exprime le fait que les π_i sont mutuellement exclusives. C'est donnés par

$$\Box(\ell > 0 \Rightarrow \lceil \bigwedge_{i=1}^{n} (\neg \pi_i \vee \bigwedge_{j \neq i} \neg \pi_j) \rceil)$$

La formule φ_{inv} exprime la relation entre deux configurations consécutives. Elle consiste en une conjonction $\bigwedge_{i=1}^{n} \varphi_i$ où chaque φ_i décrit l'effet de l'exécution de l'instruction i. Par exemple, si l'instruction i est une incrémentation du compteur c_2 et un saut à l'instruction j, alors φ_i est donnée par

$$\begin{aligned} &\Box(\lceil R \wedge \pi_i \rceil; \lceil \neg R \rceil; \lceil R \rceil; \lceil \neg R \rceil; \lceil R \rceil \Rightarrow \\ &\qquad \lceil R \wedge \pi_i \rceil; [x_k : Q_k]^2_{k=1}. (\lceil \neg R \rceil; \lceil R \wedge \pi_j \rceil; \\ &[x'_k : Q_k]^2_{k=1}. (\lceil \neg R \rceil \wedge x_1 = 2x'_1 \wedge x_2 + 1 = 2x'_2; \lceil R \rceil))) \end{aligned} \tag{7.5}$$

La formule φ_{prog} exprime le fait que chaque intervalle codant une configuration est suivi par un autre intervalle du même type. Cette formule est donnée par

$$\Box(x.(\uparrow R \wedge \lceil R \rceil; \lceil \neg R \rceil; y.(\uparrow R \wedge 2y > x + 1)) \Rightarrow \lceil R \rceil; \lceil \neg R \rceil; \lceil R \rceil; \lceil \neg R \rceil; \lceil R \rceil; vraie)$$

L'idée ici est que si un intervalle $[a, b]$ code une configuration, alors afin de coder la prochaine configuration nous avons besoin d'au plus un intervalle de longueur $(b-a)+1$. Par conséquent, il suffit d'imposer que dans tout intervalle $[b, c]$ avec $c - b > (b-a)+1$ nous devons détecter le début d'une autre configuration.

Finalement, la formule φ_{div} exprime le fait que M n'exécutes jamais l'instruction $n-1$. Elle est donnée par

$$\Box(\ell > 0 \Rightarrow \lceil \neg \pi_{n-1} \rceil)$$

■

7.9 Conclusion

Nous avons montré dans ce chapitre que le problème de la vérification d'un système de contrôle décrit en CoD par rapport à des exigences décrites aussi en CoD est décidable. C'est le premier résultat positif de décidabilité concernant les fragments de CoD qui ne sont pas propositionnels. Nous avons obtenu ce résultat en établissant des liens entre ce problème et le problème d'analyse pour les automates hybrides. En effet, étant donnés un système de contrôle φ_1 décrit en CoD et des exigences φ_2 décrites en CoD, nous pouvons construire systématiquement un automate dont le langage est vide si et seulement si toute trajectoire satisfaisant φ_1 satisfait aussi φ_2. Ce résultat est le premier résultat de traduction automatique entre le CoD et les automates hybrides. Les seuls cas de traductions connus sont des cas d'exemples particuliers. Plus encore, les résultats positifs de décidabilité que nous avons obtenu sont les premiers résultats concernant les fragment de CoD à temps dense qui ne sont pas purement propositionnels.

Chapitre 8

Conclusion

Bilan

L'objectif de ce travail était, d'une part, la mise en évidence de sous-classes de systèmes hybrides se trouvant sur la frontière séparant les systèmes décidables de ceux qui sont indécidables, d'autre part la recherche de nouvelles techniques de vérification en tenant compte des particularités de ces systèmes. Notre principal apport a été de donner des résultats de décidabilité pour des sous-classes intéressantes de systèmes hybrides qui soient aussi bien spécifiées à l'aide d'automates hybrides qu'à l'aide de logiques de durées, ainsi que l'établissement de liens entre ces deux approches.

Pour l'approche basée sur les automates hybrides, la plupart des travaux existant ne considéraient que des systèmes ayant uniquement des variables continues, nous avons pour notre part commencé par l'addition à ces automates des structures de données discrètes (pile et compteurs), ce qui nous permet d'avoir une meilleur expressivité. Dès lors nous avons raisonné sur le nombre d'occurrences de certains événements, et par conséquent, nous pouvons modéliser des comportements qui ne sont pas forcément réguliers. En effet nous pouvons décrire des comportements sous-contextes.
De plus nous montrons que pour des sous-classes de tels systèmes hybrides le problème de la vérification de propriétés d'invariance contraintes est décidable.

Nous avons par la suite essayé d'introduire dans nos systèmes des variables de contrôles continues autres que les horloges. Nous avons pu voir qu'afin de rester dans un cadre décidable nous sommes obligés d'avoir des systèmes avec des restrictions très fortes; en effet pour un intégrateur de contrôle, nous imposons que les horloges soient en phases, puis dans le cas d'un intégrateur étendu de contrôle nous avons un système à deux dimension seulement (un intégrateur étendu et un intégrateur). Dans ce dernier cas (à deux dimensions), ces systèmes sont assez restreints mais ils ont un grand intérêt; en effet ce sont les seuls systèmes connus pour lesquels le problème d'atteignabilité est décidable sans pour autant qu'il n'existe de bisimulation à borne

finie. Ces systèmes sont à la frontière séparant les systèmes décidables de ceux qui sont indécidables.

Nous établissons aussi des liens entre deux approches de spécification et de vérification de systèmes hybrides, celle basée sur les automates hybrides et celle basée sur la logique de durées. Grace à ces liens il nous a été possible d'extraire un très large fragment de la logique de durées pour lequel le problème de la vérification est décidable, alors que jusqu'à là le seul fragment connu de CoD pour lequel ce problème est décidable est le PCoD (partie propositionnelle pure du CoD).

Rappel des principaux résultats de décidabilités

Nous donnons dans le tableau qui suit un rappel de nos principaux résultats ainsi que des résultats existant, de manière à avoir une vue très claire sur la séparation entre les sous-classes de systèmes pour lesquels le problème d'atteignabilité est décidable et les sous-classes de systèmes pour lesquels ce même problème est indécidable. Ce tableau comporte 4 colonnes, la première correspond au nom de la sous-classe, la seconde correspond aux différents types de variables qui interviennent dans ces sous-classes, la troisième correspond aux contraintes qui sont associées à ces systèmes et la quatrième contient le résultat de décidabilité de ces systèmes.
Lorsque le résultat est positif (systèmes décidables) la troisième colonne doit être vue comme énonçant des restrictions sinon comme énonçant un relachement. C'est-à-dire que si pour des systèmes indécidables on a des restrictions alors en les enlevant, ces systèmes restent indécidables.

Nom de la sous-classe	Variables	Restrictions	Résultat
Systèmes temporisés	Horloges	Aucune	Décidables [ACD93]
STP	Horloges + 1 pile	Aucune	Décidables [Rob95]
STPCM	Horloges + 1 pile +des compteurs monotones	Aucune	Décidables [Rob95]
STPCO	Horloges + 1 pile +des compteurs d'observations	Aucune	Décidables [Rob95]
STPCMO	Horloges + 1 pile +des compteurs monotones compteurs d'observations	Aucune	Décidables [Rob95]
STPVO	Horloges + 1 pile + 1 variable continue d'observation	systèmes larges	Décidables [Rob95]
1-SIP	Horloges + 1 intégrateur	systèmes larges et simples	Décidables [Rob95]
STPCMOV	Horloges + 1 pile +des compteurs monotones +des compteurs d'observations + 1 variable continue d'observation	systèmes larges	Décidables [Rob95]
STPCMOI	Horloges + 1 pile +des compteurs monotones +des compteurs d'observations + 1 intégrateur	systèmes larges et simples	Décidables [Rob95]
2-dim-GIE	1 intégrateur + 1 intégrateur étendu	Aucune	Décidables [Rob95]
3-dim-GIE	1 horloge + 2 intégrateurs étendus	larges	Indécidables [KPSY93]
4-dim-GIE	3 horloges + 1 intégrateur étendu	systèmes larges	Indécidables [Kop95a]
5-dim-STI	4 horloges + 1 intégrateur	systèmes larges	Indécidables [Kop95b]

Tableau récapitulatif.

Applications

Notre travail a aussi un intérêt pratique, puisqu'il permet de bénéficier des avantages des deux approches, aussi bien celle qui est basée sur les automates hybrides que celle qui est basée sur le CoD. Ces résultats permettent de profiter d'une part du CoD comme un langage de spécification de haut niveau, puis d'autre part de plusieurs techniques d'analyses existantes concernant les automates hybrides ainsi que la multiplicité des outils de vérification de tels automates [Yov93, Oli94, DOY94, ACH+95]. Il existe plusieurs systèmes de preuves permettant de vérifier les systèmes par rapport à des exigences décrits dans le CoD, il serait intéressant d'intégrer à ces systèmes de preuves nos procédures de décisions, afin de les rendre plus performants.
De plus il serait très intéressant d'utiliser la logique de durées comme un formalisme permettant d'exprimer des contraintes temporelles dans des documents multimédias. En effet, jusqu'à récemment il y a eu une utilisation croissante de formalismes fondés sur les grammaires pour la description et la manipulation de documents statiques. Ces formalismes ont facilités la production d'outils puissants d'assistance à la réalisation de tels documents. Ils ont également fourni une base pour la résolution des problèmes de transformation de documents. Malgré l'intérêt croissant dans le multimédia et la disponibilité d'un certain nombre de systèmes pour le développement de systèmes multimédias, il n'existait pas de formalismes pour la description de contraintes temporelles dans de tels systèmes. L'unique tentative a été celle de King [Kin94] qui en se basant sur la logique de Moszkowski [Mos85, All83], montre comment implanter les fonctionnalités requises dans la norme HyTime [10792, Erf94]. C'est dans ce sens qu'il serait pratique d'utiliser le fragment décidable de DIL que nous avons mis en évidence comme formalisme pour exprimer les contraintes temporelles dans des documents multimédias. Car le fragment que nous exhibons est non seulement plus expressif que celui utilisé dans [Kin94], mais de plus nous avons établis des procédures de décision pour ce fragment.

Perspectives

Plusieurs perspectives sont envisageables, la première d'entre elles est de savoir ce qui se passe du point de vu décidabilité en trois dimensions (3-dim-GIE) mais avec deux horloges et une variable linéaire, tout en sachant que pour le cas de trois horloges plus une un intégrateur étendu ce même problème a été montré indécidable [Kop95a] et que pour le cas d'une horloge et d'un intégrateur étendu nous avons montré que ce problème est décidable (voir le chapitre 6).

Puis dans le cas où nous utilisons la logique de durées comme un formalisme permettant d'exprimer des contraintes temporelles dans des documents multimédias, il serait intéressant d'avoir ce que l'on pourrait appeler par la notion d'incrémentalité. C'est-à-dire comment traiter le rajout d'une nouvelle contrainte entre les médias sans pour autant avoir à refaire tout le traitement relatif aux contraintes d'origines (pour lesquelles le traitement a été effectué). En d'autres termes, comment utiliser à bon profit le résultat d'une analyse déjà effectuée lors du rajout d'une ou de plusieurs contraintes. Ce problème est similaire à celui du traitement de la composition parallèle, qui consiste à savoir sous quelles contraintes il est possible de répercuter sur le système global le résultat des différentes analyses locales.

Il serait aussi très intéressant d'introduire des structures de données discrètes au niveau de la logique de durées; il faut commencer par introduire la notion de compteurs tels que chacun d'eux est associé à un état (un ensemble de propositions) et comptabilise le nombre d'occurrences de ce même état. Il faudrait aussi introduire la notion de pile au niveau de cette logique de durées, tout en adoptant une sémantique bien particulière. Nous pouvons par exemple adopter le fait que les opérations sur la pile s'effectuent de la manière suivante:
lorsqu'on a un passage de fausse à vraie d'une proposition particulière p alors pour chaque valeur de la tête de la pile nous la remplaçons par une chaîne du vocabulaire de la pile.
A l'aide d'une telle extension nous pourrons aussi établir des liens avec les systèmes que nous avons adoptés dans le chapitre 5 (systèmes hybrides munis de structures de données discrètes). Cette extension nous permet d'avoir un formalisme basé sur la logique de durées qui soit plus expressif.

Bibliographie

[10792] International Standard ISO/IEC 10744. Information Technology-Hypermedia/Time-based Language (HyTime). *ed., International Organization for Standardization, Geneva/New-York, first edition,*, 1992.

[ACD93] R. Alur, C. Courcoubetis, and D.L. Dill. Model checking in dense real time. *Information and Computation*, 104(1):2–34, 1993.

[ACH93] R. Alur, C. Courcoubetis, and T. A. Henzinger. Computing Accumulated Delays in Real-time Systems. In *CAV'93*. LNCS 697, 1993.

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The Algorithmic Analysis of Hybrid Systems. *TCS*, 138, 1995.

[ACHH93] R. Alur, C. Courcoubetis, T. Henzinger, and P-H. Ho. Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. In *Hybrid Systems*. LNCS 736, 1993.

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

[AL92] M. Abadi and L. Lamport. A Formal Approach to Hybrid Systems. In *REX workshop on Real-Time: Theory and Practice*. LNCS 600, 1992.

[All83] J.F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, Vol. 26 No. 11, 1983.

[Alu91] R. Alur. *Techniques of Automatic Verification of Real-Time Systems*. PhD thesis, Stanford Univ., 1991.

[AM95] L de Alfaro and Zohar Manna. Verification in Continuous Time by discrete Reasoning. In *AMAST'95*, 1995. IEEE.

[AMP95] E. Asarin, O. Maler, and A. Pnueli. Symbolic Controller Synthesis for Discrete and Timed Systems. In *Hybrid Systems II*, 1995. To appear.

[BEH94] A. Bouajjani, R. Echahed, and P. Habermehl. Verifying Infinite State Processes with Sequential and Parallel Composition. In *in Proc. 22nd Symp. on Principles of Programming Languages (POPL'95), San Francisco.* ACM, 1994.

[BEH95] A. Bouajjani, R. Echahed, and P. Habermehl. On the Verification Problem of Nonregular Properties for Nonregular Processes. In *in Proc. 10th Symp. on Logic In Computer Science (LICS'95), San Diego.* IEEE, 1995.

[BER93] A. Bouajjani, R. Echahed, and R. Robbana. Constrained Grammars. Technical report, VERIMAG, 1993.

[BER94a] A. Bouajjani, R. Echahed, and R. Robbana. Decidability Results for the Verification Problem of Pushdown Linear Hybrid Systems. Technical report, Verimag, 1994.

[BER94b] A. Bouajjani, R. Echahed, and R. Robbana. Verification of Context-Free Timed Systems using Linear Hybrid Observers. In *Proc. Computer Aided Verification (CAV'94).* LNCS 818, 1994.

[BER94c] A. Bouajjani, R. Echahed, and R. Robbana. Verification of Nonregular Temporal Properties for Context-Free Processes. In *CONCUR'94.* LNCS 836, 1994.

[BER94d] A. Bouajjani, R. Echahed, and R. Robbana. Verifying Invariance Properties of Timed Systems with Duration Variables. In *FTRTFT'94.* LNCS 863, 1994.

[BER95] A. Bouajjani, R. Echahed, and R. Robbana. On the Automatic Verification of Systems with Continuous Variables and Unbounded Discrete Data Structures. In *"Hybrid systems and autonomous control".* To appear, 1995.

[BES93] A. Bouajjani, R. Echahed, and J. Sifakis. On Model Checking for Real-Time Properties with Durations. In *8th Symp. on Logic in Computer Science (LICS'93).* IEEE, 1993.

[BKP86] H. Barringer, R. Kuiper, and A. Pnueli. A Really Abstract Concurrent Model and its Temporal Logic. In *POPL'86*, 1986.

[BLR95] A. Bouajjani, Y. Lakhnech, and R. Robbana. From Duration Calculus to Linear Hybrid Automata. In *CAV'95.* LNCS 939, 1995.

[BR95] A. Bouajjani and R. Robbana. Verifying ω-Regular Properties for Subclasses of Linear Hybrid Systems. In *CAV'95.* LNCS 939, 1995.

[Buc62] J.R. Buchi. On a Decision Method in Restricted Second Order Arithmetic. In *Intern. Cong. Logic, Method and Philos. Sci.* Stantford Univ. Press, 1962.

[Bur92] Jerry R. Burch. *Trace Algebra for Automatic Verification of Real-Time Concurent Systems.* PhD thesis, Carnegie Mellon univ., 1992.

[CE81] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Proceedings of Workshop on Logic of Programs.* Springer-Verlag, 1981. LNCS 131.

[Cer92] K. Cerans. Decidability of Bisimulation Equivalence for Parallel Timer Processes. In *Proc. Computer-Aided Verification (CAV'92).* LNCS 663, 1992.

[CES83] E.M. Clarke, E.A. Emerson, and E. Sistla. Automatic verification of finite state concurrent systems using temporal logic specification: a practical approach. In *10th ACM Symposium on Principles of Programming Languages (POPL83)*, 1983. Complete version published in ACM TOPLAS, 8(2):244–263, April 1986.

[CG77] R.S. Cohen and A.Y. Gold. Theory of ω-Languages. I: Characterizations of ω-Context-Free Languages. *Journal of Computer and System Sciences*, 15:169–184, 1977.

[CHR91] Z. Chaochen, C.A.R. Hoare, and A.P. Ravn. A Calculus of Durations. *Information Processing Letters*, 40:269–276, 1991.

[CHR93] Z. Chaochen, C.A.R. Hoare, and A.P. Ravn. An extended duration calculus for hybrid real time systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems.* Springer-Verlag, 1993. LNCS 736.

[CHS93] Z. Chaouchen, M.R. Hansen, and P. Sestoft. Decidability and Undecidability Results for Duration Calculus. In *STACS'93.* LNCS 665, 1993.

[CL94] Z. Chaochen and X. Li. A Mean -Value Duration Calculus. In *A Classical Mind, Essays in Honour of C.A.R. Hoare.* Prentice-Hall, 1994.

[CS91] R. Cleaveland and B. Steffen. A Linear-Time Model-Checking Algorithm for the Alternation-Free Modal Mu-Calculus. In *Proc. Computer-Aided Verification (CAV'91).* Springer-Verlag, 1991. LNCS 575.

[Dan48] G.B Dantzig. Programming in a Linear Structure. In *United States Air Force, Washington, D.C*, February 1948.

[DOY94] C. Daws, A. Olivero, and S. Yovine. Verifying ET-LOTOS programs with KRONOS. In *Proc. FORTE'94*, Berne, Switzerland, October 1994.

[DY95] C. Daws and S. Yovine. Tow examples of verification of multirate timed automata with KRONOS. In *Proc. 1995 Real-Time Systems Symposium, RTSS'95*, Pisa, Italy, December 1995. IEEE Computer Society Press.

[EGK87] E.Clarke, O. Grümberg, and R. Kurshan. A Synthesis of two Approaches for Verifying Finite State Concurrent Systems. In *CMU tech. rep.*, 1987.

[EH83] E. A. Emerson and J. Y. Halpern. 'Sometimes' and 'not never' revisited: On branching versus linear time. In *10th ACM Symposium on Principles of Programming Languages (POPL'83)*, 1983. also published in Journal of ACM , 33:151-178.

[EL86] E.A. Emerson and C.L. Lei. Efficient Model-Checking in Fragments of the Propositional μ-Calculus. In *First Symp. on Logic in Computer Science*, 1986.

[Eme90] E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 995–1072. Elsevier Science Publishers (North-Holland), 1990.

[Erf94] R. Erfle. Specification of temporal constraints in multime'dia documents using HyTime. In *Fifth International Conference on Electronic Publishing, Document Manipulation and Typography*, 1994.

[FGM+92] Jean-Claude Fernandez, Hubert Garavel, Laurent Mounier, Anne Rasse, Carlos Rodríguez, and Joseph Sifakis. A toolbox for the verification of lotos programs. In Lori A. Clarke, editor, *Proceedings of the 14th International Conference on Software Engineering ICSE'14 (Melbourne, Australia)*, pages 246–259, New-York, May 1992. ACM.

[Hal93] N. Halbwachs. Delay Analysis in Synchronous Programs. In *CAV'93*. LNCS 697, 1993.

[Har78] M.A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley Pub. Comp., 1978.

[Hen95] T. Henzinger. Hybrid Automata with Finite Bisimulations. In *ICALP'95*, 1995.

[HHF+94] J. He, C.A.R. Hoare, M. Fränzle, M. Müller-Olm, E.R. Olderog, M. Schenke, M.R. Hansen, A.P. Ravn, and H. Rishel. Provably Correct Systems. In *FTRTFT'94*. LNCS 863, 1994.

[HK94] T.A. Henzinger and P.W. Kopke. Undecidability Results for Hybrid Systems. *Draft*, October 1994. Presented in Workshop on Hybrid Systems and Autonomous Control, Cornell University, Ithaca.

[HKPV95] T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's Decidable about Hybrid Automata. In *STOC'95*, 1995.

[HMP92] T. Henzinger, Z. Manna, and A. Pnueli. What Good are Digital Clocks? In *19th. Internat. Coll. on Automata, Languages and Programming.* LNCS 623, 1992.

[HNSY94] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[Hoo93] J. Hooman. A compositional approach to the disign of hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems.* Springer-Verlag, 1993. LNCS 736.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley Pub. Comp., 1979.

[Kin94] P.R. King. Towards a Temporal Logic Based Formalism For Expressing Temporal Constraints in Multimedia Documents. Technical report, L.R.I, 1994.

[Kop95a] P. Kopke. Communication personnelle juin 1995. Unpublished Notes, 1995.

[Kop95b] P. Kopke. Communication personnelle mai 1995. Unpublished Notes, 1995.

[Koz83] D. Kozen. Results on the propositional μ-calculus. In *Theoretical Computer Science.* North-Holland, 1983.

[KPSY93] Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine. Integration Graphs: A Class of Decidable Hybrid Systems. In *Hybrid Systems.* LNCS 736, 1993.

[Lam93] L. Lamport. Hybrid Systems in TLA^+. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems.* Springer-Verlag, 1993. LNCS 736.

[Mil89] R. Milner. *Communication and concurrency.* Prentice-Hall, 1989.

[MMP92] O. Maler, Z. Manna, and A. Pnueli. A Formal Approach to Hybrid Systems. In *REX workshop on Real-Time: Theory and Practice.* LNCS 600, 1992.

[Mos85] B. Moszkowski. A Temporal Logic for Multilevel Reasoning about Hardware. *IEEE Computer*, 18(2):10–19, 1985.

[MP89] Z. Manna and A. Pnueli. The anchored version of the temporal framework. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Lecture Notes in Computer Science 354*. Springer-Verlag, 1989.

[MP93] Z. Manna and A. Pnueli. Verifying Hybrid Systems. In *Hybrid Systems*. Springer-Verlag, 1993. LNCS 736.

[Mul63] D.E. Muller. Infinite Sequences and Finite Machines. In *4th Symp. on Switching Circuit Theory and Logical Design*. IEEE, 1963.

[Nic92] Xavier Nicollin. *ATP : une algèbre pour la spécification et l'analyse des systèmes temporisés*. PhD thesis, Institut Nationale Polytechnique de Grenoble, 1992.

[NK93] A. Nerode and W. Kohn. Models for Hybrid Systems: Automata, topologies, controllability, observability. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*. Springer-Verlag, 1993. LNCS 736.

[NOSY93] X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. An Approach to the Description and Analysis of Hybrid Systems. In *Hybrid Systems*. LNCS 736, 1993.

[NSY92] X. Nicollin, J. Sifakis, and S. Yovine. From ATP to Timed Graphs and Hybrid Systems. In *REX workshop on Real-Time: Theory and Practice*. LNCS 600, 1992.

[Oli94] Alfredo Olivero. *Modélisation et Analyse de Systèmes Temporisés et Hybrides*. PhD thesis, Institut Nationale Polytechnique de Grenoble, 1994.

[OSY94] A. Olivero, J. Sifakis, and S. Yovine. Using abstractions for the verification of linear hybrid systems. In D. Dill, editor, *Proc. 6th Computer-Aided Verification*, pages 81–94, California, June 1994. Lecture Notes in Computer Science 818, Springer-Verlag.

[PHKV95] A. Puri, T. Henzinger, P. Kopke, and P. Varaiya. What's Decidable about Hybrid Automata. In *STOC'95*, 1995.

[Pnu77] A. Pnueli. The Temporal Logic of Programs. In *FOCS'77*. IEEE, 1977.

[PV94] A. Puri and P. Varaiya. Decidability of Hybrid Systems with Rectangular Differential Inclusions. In *CAV'94*. LNCS 818, 1994.

[QS82] J-P. Queille and J. Sifakis. Specification and Verification of Concurrent Systems in CESAR. In *International Symposium on Programming, LNCS 137.* Springer Verlag, 1982.

[Rab69] M.O. Rabin. Decidability of Second Order Theories and Automata on Infinite Trees. *Trans. Amer. Math. Soc.*, 141, 1969.

[Rav94] A. Ravn. Design of Embedded Real-time Computing Systems. Unpublished Notes, 1994.

[Rob95] Riadh. Robbana. *Spécification et vérification de systèmes hybrides.* PhD thesis, Ce document, 1995.

[RR93] S. Ramanathan and P. Venkat Rangan. Feedback Techniques for Intra-Media Continuity and Inter-Media Synchronization in Distributed Multimedia Systems. *The Computer Journal*, Vol. 36 No. 1, 1993.

[RRH93] A.P. Ravn, H. Rischel, and K.M. Hansen. Specifying and Verifying Requirements of Real-Time Systems. In *Transaction on Software Engeneering.* IEEE, 1993.

[SRK83] S.Aggrawal, R.P.Kurshan, and K.Sabnani. A calculus for protocol specification and validation. *Protocol Specification, Testing, and Verification*, 3:19–34, 1983.

[SS94] Jens Ulrik Skakkebaek and Natarajan Shankar. Towards a Duration Calculus Proof Assistant in PVS. In *FTRTFT'94.* LNCS 863, 1994.

[Tho89] W. Thomas. Computation Tree Logic and Regular ω-Languages. In . LNCS 354, 1989.

[Tho90] W. Thomas. Automata on Infinite Objects. In *Handbook of Theo. Comp. Sci.* Elsevier Sci. Pub., 1990.

[Var87] Moshe Vardi. Verification of concurrent programs- the automata-theoretic framework. In *2th Symp. on Logic in Computer Science (LICS'87).* IEEE, 1987.

[Var88] M.Y. Vardi. A Temporal Fixpoint Calculus. In *POPL'88*, 1988.

[Wol83] P. Wolper. Temporal Logic Can Be More Expressive. *Inform. and Cont.*, 56, 1983.

[WVS83] P. Wolper, M. Vardi, and A.P. Sistla. Reasoning about infinite computation paths. In *FOCS'83.* IEEE, 1983.

[Yov93] Sergio Yovine. *Méthodes et Outils pour la Vérification Symbolique de Systèmes Temporisés.* PhD thesis, Institut Nationale Polytechnique de Grenoble, 1993.

[YPS94] Houiqun Yu, Paritosh K. Pandya, and Yongqiang Sun. A Calculus for Hybrid Sampled Data Systems. In *FTRTFT'94*. LNCS 863, 1994.

[YWZP94] Xinyao Yu, Ji Wang, Chaochen Zhou, and Paritosh K. Pandya. Formal Design of Hybrid Syste'ms. In *FTRTFT'94*. LNCS 863, 1994.

[Zho94] Chaochen Zhou. Linear Duration Invariants. In *FTRTFT'94*. LNCS 863, 1994.

[ZZ94] Yuhua Zheng and Chaochen Zhou. A Formal Proof of the Deadline Driven Scheduller. In *FTRTFT'94*. LNCS 863, 1994.