ENSC-2009 n^o 185

THÈSE

présentée à l'École Normale Supérieure de Cachan

en vue de l'obtention du grade de

Docteur de l'École Normale Supérieure de Cachan

par : Florent BOUCHY

Spécialité : INFORMATIQUE

Logiques et modèles pour la vérification de systèmes infinis

Soutenue le 10 novembre 2009, devant un jury composé de :

– Alain FINKEL	directeur de thèse
– Serge HADDAD	examinateur
– Jérôme LEROUX	examinateur
– Laure PETRUCCI	rapporteuse
– Marc ZEITOUN	rapporteur

Résumé

Notre société actuelle dépend fortement de systèmes informatiques : appareils médicaux, transports en commun, voitures, téléphones portables, centrales nucléaires, ascenseurs, banques, industrie, exploration spatiale, etc. Ces systèmes prennent une place de plus en plus grande dans notre vie quotidienne, que ce soit de façon directe ou non. Les enjeux du bon fonctionnement de tels systèmes sont alors cruciaux, tant sur le plan économique qu'environnemental ou humain. La sophistication et l'omniprésence de ces systèmes font qu'ils deviennent de plus en plus complexes, à tel point que les techniques d'ingénierie actuelles sont généralement incapables de garantir le bon fonctionnement des systèmes qui nous entourent.

Le model-checking est une technique basée sur la modélisation de systèmes (notamment par des automates finis ou étendus avec des variables, des piles, des files, etc.), et sur la modélisation de propriétés du bon fonctionnement des systèmes par des logiques temporelles. La complexité des systèmes actuels amène à s'intéresser au model-checking de systèmes infinis, en particulier en ajoutant aux automates la capacité de manipuler des variables non-bornées afin de mieux modéliser la réalité. Les variables considérées sont de type réel, entier, ou mixte (i.e. les deux à la fois).

Afin de représenter des ensembles infinis de valeurs pour ces variables, on utilise des logiques arithmétiques du premier ordre. On propose un moyen de décomposer ces logiques réelles ou mixtes en logiques entières et en logiques décimales (i.e. réelles entre 0 et 1). On définit également une logique mixte décidable qui est plus expressive que la plupart des logiques du premier ordre connues.

On étudie des modèles de systèmes infinis temporisés, à compteurs, ou les deux à la fois. On s'intéresse alors à leurs problèmes d'accessibilité, i.e. savoir si l'on peut calculer, dans un modèle donné, l'ensemble de ses configurations accessibles. On introduit un nouveau modèle de systèmes à compteurs temporisés, pour lequel on généralise la construction du graphe des régions, ce qui permet de décider les problèmes d'accessibilité en fonction de l'expressivité des compteurs. Enfin, on complète l'étude du modèle des DCM introduit par Ibarra et San Pietro, en généralisant les gardes sur les transitions, en apportant de nouveaux résultats de décidabilité, et en montrant que toute formule de l'arithmétique linéaire mixte est définissable par une DCM à renversements bornés.

Mots-clés : Vérification formelle, Logique arithmétique, Systèmes à compteurs, Automates temporisés, Systèmes hybrides, Accessibilité.

Abstract

Our current society heavily relies on computer systems : medical devices, public transit, cars, cellphones, nuclear plants, elevators, banks, industry, spatial exploration, etc. These systems are becoming more and more important to our daily life, directly or not. The correct performance of such systems is a major stake on an economical level, as well as on both environmental and human levels. These systems are getting more and more complex due to their sophistication and omnipresence, so much that current engineering techniques are generally unable to guarantee that the systems surrounding us are operating correctly.

The model-checking technique relies on the modelling of such systems (namely using finite automata, possibly extended with variables, stacks, channels, etc.), and on the modelling of properties ensuring correct operation of systems (using temporal logics). The complexity of current systems suggests the application of model-checking techniques to infinite-state systems, in particular by endowing automata with the ability to use unbounded variables so that the models get closer to reality. Considered variables are either real-valued, integer-valued, or mixed (i.e. both at the same time).

In order to represent infinite sets of values for such variables, we use first-order arithmetical logics. We introduce a means to decomposing real-valued or mixed logics into integer-valued logics and decimal logics (i.e. real-valued between 0 and 1). We also define a decidable mixed logic which has a greater expression power than most known first-order logics.

We study models for infinite-state systems extended with time, counting, or both at the same time. We then look into their reachability problems, i.e. knowing whether for a given model, the set of its reachable configurations is computable. We define a new model for timed counter systems, for which we generalize the region graph construction, enabling to decide rechability problems according to the expression power of counters. Finally, we complete the study of DCM (a model introduced by Ibarra and San Pietro) by generalizing the guards allowed on transitions, by bringing new decidability results, and by showing that every formula of the mixed linear arithmetic is definable by a reversal-bounded DCM.

Keywords : Formal verification, Arithmetical logics, Counter systems, Timed Automata, Hybrid systems, Reachability.

Remerciements

Je remercie le LSV, qui est un labo où il fait extrêmement bon vivre, de m'avoir permis de me former avec tant de rigueur dans un cadre si confortable. Je ne connais aucun autre laboratoire dans lequel j'aurais terminé ma thèse tout en progressant autant : merci aux fondateurs du LSV d'y avoir instauré cet environnement de travail si spécial. Merci aux membres du LSV, passés et présents, et futurs (attention à ne pas laisser cette qualité s'effacer sous le poids de l'effectif toujours grandissant !).

Merci Alain pour ta confiance, tes conseils, ton encadrement, ta pluridisciplinarité, et pour tes fameuses techniques "manipulatrices" qui ont su me motiver et me remettre sur les rails quand j'en suis si souvent sorti ; tu as su me guider tout en me faisant travailler par moi-même, ce qui était indispensable pour ma formation mais pas facile de me pousser à faire.

Merci à Laure et Marc pour avoir relu cette thèse : vos commentaires et corrections ont été très appréciés. Merci également à Serge et Jérôme d'avoir participé à mon jury de soutenance. Merci à tous les membres du jury pour vos appréciations, vos encouragements, et vos questions intéressantes.

Un grand merci à tous mes co-auteurs, sans qui je n'aurais probablement pas réussi à me lancer sur la voie de la publication, essentielle dans la recherche : merci pour vos idées, vos conseils, et votre temps. Merci Alain de m'avoir appris comment présenter un résultat tout en sachant se mettre à la place du redoutable relecteur. Merci Jérôme pour ta disponibilité, ta sympathie, et ton niveau exemplaire parfois presque décourageant (oui oui tu es très fort ☺ ; je te rassure, c'est très motivant de travailler avec toi). Merci Arnaud de m'avoir montré la voie et de m'avoir accepté comme petit frère scientifique, même quand je mettais trois mouchys à réagir ☺. Thank you Pierluigi for all the work and effort you have put in our collaboration.

Merci à ceux qui ont partagé mon bureau en le rendant plus vivant et agréable (la liste serait trop longue !). Merci à Fabrice, Pierre-Alain, Jules, Delphine, et les deux Arnaud, pour les fois où vous avez réfléchi avec moi sur des questions de recherche, d'enseigne-

ment, ou de latexologie. Merci Jules pour tes goodies, notamment les paquets de mouchoirs rose-princesse et les autocollants de dinosaures (en particulier l'Ankylosaure, qui paraît si menaçant). Merci Delphine pour ta compagnie, surtout dans ces moments si forts au "restaurant universitaire" de l'école... (miam !). Merci Najla de croire que la logique de Presburger a un quelconque rapport avec le FAST food : ça donne enfin un sens à ma recherche ! Merci les coburals. Merci aussi aux académiciens de la bière, grâce à qui j'ai pu garder ma santé nerveuse, entretenir ma culture gastronomico-bièresque, et vider mon portefeuille tout en remplissant ma carte de fidélité et mon tour de taille sans commune mesure. Merci à tous d'avoir supporté mon humour à deux balles.

Un merci tout particulier à ma famille pour son soutien, surtout à ma maman pour toujours avoir été là et pour ce pot de soutenance extraordinaire et mémorable. Merci à mes ami(e)s, sans qui je ne serais pas qui je suis ; merci à celles et ceux qui m'ont soutenu et accompagné en France, et merci à celles et ceux qui m'ont attendu à Montréal durant ces trois longues années. Merci aussi à toutes les personnes, proches ou inconnues, qui ont accepté mes réponses évasives et succintes à toutes les questions du genre "mais en fait, tu fais quoi exactement ?". Et pour celles et ceux qui n'ont pas trouvé satisfaction dans mes réponses : je vous invite à lire ce tapuscrit ! (oui, ce mot existe bel et bien, malheureusement...)

Enfin, merci à toi, lecteur : tu es si rare !

Introduction

Enjeux de la vérification

Dans notre société actuelle, force est de constater que l'informatique est devenue omniprésente. En effet, notre vie quotidienne fait sans cesse appel à des bases de données, des appareils électroniques, des dispositifs de contrôle, des technologies de transport, des télécommunications, etc. Ces applications informatiques régissent le fonctionnement de ce que l'on appelle des *systèmes*, dont voici quelques exemples : téléphones et ordinateurs portables, automobiles, cartes bancaires, ascenseurs, logiciels, composants électroniques, centrales nucléaires, accélérateurs de particules, etc.

Ces systèmes, qu'ils soient logiciels ou matériels, suivent naturellement l'évolution technologique : ils bénéficient donc de plus en plus de mémoire, et sont de plus en plus rapides. Par conséquent, ils deviennent de plus en plus complexes, ce qui les rend de plus en plus difficiles à concevoir sans erreur ! Cela pose évidemment un problème majeur, étant données les conséquences désastreuses que peut avoir un dysfonctionnement, aussi infime puisse-t-il paraître.

En 1996, la fusée Ariane 5 a explosé 30 secondes après son décollage, suite à une erreur de programmation apparemment mineure. Entre 1985 et 1987, en Amérique du Nord, les appareils médicaux Therac-25 ont tué plusieurs personnes en les exposant à des doses trop élevées de radiations à cause d'une erreur de conception logicielle. Ces deux exemples illustrent la nécessité de vérifier que les systèmes soient sûrs, afin d'éviter les coûts potentiellement dramatiques d'une erreur, sur le plan financier mais aussi écologique ou humain.

On imagine facilement les conséquences d'une défaillance dans un métro, un avion, ou encore une centrale nucléaire. Il est donc crucial de garantir la fiabilité de tels systèmes, en certifiant qu'ils respectent certains standards de qualité. Les erreurs dans ces systèmes sont généralement dues à une conception imparfaite ; les concepteurs sont en effet des êtres humains, et leur *vérification* du bon fonctionnement d'un système est limitée, ne serait-ce qu'à cause de la complexité des systèmes en question.

De plus, les besoins d'un système sont généralement rédigés sous la forme d'un cahier des charges, dans une langue naturelle (français, anglais, etc.), ce qui est une source majeure d'ambiguïtés, d'imprécisions, voire d'omissions ou de contradictions ! C'est pourquoi la vérification de systèmes doit être effectuée de façon rigoureuse, systématique, et avec le moins d'intervention humaine possible. Pour ce faire, on utilise des outils issus des mathématiques et de l'informatique théorique, afin de prouver de façon formelle qu'un système fait bien ce que l'on attend de lui et qu'il ne comporte pas d'erreur.

On parle alors de vérification formelle.

Méthodes formelles

Il existe principalement trois techniques de vérification formelle, qui se placent dans ce que l'on appelle les "méthodes formelles" : le test, la démonstration automatique, et le model-checking (qui se traduit par "vérification de modèle"). Toutes ces techniques visent à prouver qu'un système fonctionne correctement : on modélise mathématiquement le système, d'une part, et le cahier des charges, d'autre part, puis on applique une de ces techniques (par exemple, le test) pour vérifier que les deux modèles sont bien compatibles. En d'autres termes, on vérifie que le modèle du système réel répond bien aux exigences présentes dans le cahier des charges.

Cependant, le test n'est pas toujours applicable ; et la démonstration automatique fait souvent appel à l'expertise humaine, qui est rapidement limitée par la complexité du problème. Le model-checking, lui, n'a pas ces inconvénients ; par contre, il utilise différents algorithmes (différentes approches) selon la complexité des types de modèles utilisés. La recherche en model-checking consiste principalement à trouver des algorithmes de plus en plus efficaces, sur des types de modèles de plus en plus puissants, afin de pouvoir vérifier les systèmes réels dans leur intégralité. C'est dans ce cadre-là que se placent les travaux présentés dans cette thèse.

Depuis les premières idées du model-checking il y a une trentaine d'années, de nombreuses applications industrielles ont prouvé l'efficacité de cette technique : citons notamment les microprocesseurs Intel et Motorola, la ligne de métro automatique de Paris [Bou99], quelques logiciels Microsoft, et des protocoles de communication. De plus, les chercheurs considérés comme les inventeurs du model-checking (E.M. Clarke, E.A. Emerson, et J. Sifakis) ont reçu en 2007 le prix Turing, qui est l'équivalent d'un prix Nobel pour l'Informatique.

Cette technique de vérification est en plein essor depuis sa création, tant sur le plan

de la recherche académique (voir [BBF+01]) que, plus récemment, dans le milieu industriel. Quoi qu'il en soit, certains domaines d'application nécessitent toujours de trouver de nouveaux algorithmes de model-checking, fonctionnant sur des nouveaux types de modèles de systèmes. La spécification du système, elle aussi, peut donner lieu à de nouveaux algorithmes : selon les propriétés du système inscrites dans le cahier des charges, de nouveaux modèles de spécification peuvent en effet s'avérer nécessaires.

De façon générale, on cherche constamment à ce que les modèles soient plus puissants, de sorte qu'ils décrivent plus fidèlement la réalité. Les modèles ne sont en effet que des représentations mathématiques de cette réalité, et ne la décrivent que de façon partielle. Le plus souvent, les systèmes peuvent avoir une infinité de comportements différents ; on ne peut donc pas assurer que tous ces comportements sont sûrs et correspondent à leur spécification.

De plus, cette infinité implique qu'un être humain est généralement incapable de penser à tous les cas de figure envisageables. C'est pourquoi on fait appel à des modèles que l'on appelle *systèmes infinis*, dont on cherche à vérifier le bon fonctionnement et l'absence d'erreurs.

Pour ce faire, l'algorithme de model-checking utilise une abstraction qui réduit l'infinité de comportements possibles à un nombre fini d'actions, de sorte qu'il devienne calculable en pratique. Cette abstraction est une simplification ; le fait qu'elle reste fidèle au système réel dépend alors fortement de la spécification des propriétés que le système doit satisfaire, ce qui donne lieu à différents algorithmes selon les cas de figure.

Logiques et modèles de systèmes pour la vérification

L'algorithme de model-checking dépend de deux composantes principales : un modèle du système (généralement, un *automate*), et un formalisme de spécification des propriétés que le système doit satisfaire (généralement, une *logique temporelle*). Afin d'augmenter leur pouvoir d'expression, les automates peuvent être étendus par des variables, notamment des *compteurs* et des *horloges*. Ces deux types de variables sont parmi les plus utilisés, par exemple pour des dispositifs de comptage, de la communication par messages, ou de l'empilement de données (pour les compteurs), ainsi que pour le temps-réel ou la synchronisation (pour les horloges).

Dans cette thèse, on définit de nouveaux modèles de systèmes prenant en compte de tels automates qui combinent compteurs et horloges. Le fait d'utiliser ne serait-ce qu'un seul compteur discret non-borné, ou une seule horloge continue, fait que le modèle du système (en l'occurrence, l'automate) peut se trouver dans une infinité de configu-

rations, d'où la difficulté de combiner ces deux sources d'infinitude. Couplées à des automates, diverses logiques sont utilisées de deux façons différentes.

Premièrement, les opérations sur les variables de l'automate sont décrites par une *logique arithmétique* (notamment la logique de Presburger pour les compteurs, et des contraintes linéaires pour les horloges), afin de gérer la partie infinie du comportement du système. Ce type de logique sert également à caractériser l'ensemble des comportements possibles du système.

Deuxièmement, des logiques temporelles (telles que LTL, CTL, MITL, etc.) servent à exprimer les propriétés du système que l'on veut vérifier. Ici, on se place dans le cadre du model-checking, mais on ne s'intéressera pas aux logiques temporelles.

Contributions de cette thèse

Les travaux présentés dans cette thèse portent principalement sur deux éléments importants dans la vérification de systèmes infinis : les **logiques arithmétiques** et les **modèles de systèmes**.

Les principaux résultats obtenus ont été publiés dans les trois articles suivants :

– Dans [BFL08], on définit un opérateur de décomposition de logiques arithmétiques. On applique cet opérateur sur trois logiques connues, et on en présente une implémentation. De nouveaux résultats, en cours de rédaction pour une revue, proposent deux nouvelles logiques définies à l'aide de cet opérateur, l'une de ces logiques étant plus expressive que les autres tout en restant décidable.

– Dans [BFS09], on définit un modèle de systèmes combinant compteurs et horloges, en proposant un moyen de l'analyser et en identifiant trois sous-classes décidables.

– Dans [BFSP09], on clarifie et on étend la définition d'un modèle de systèmes original, en motivant son étude. On donne de nouveaux résultats de décidabilité sur ce modèle, ainsi qu'une caractérisation logique d'une sous-classe analysable. Dans la version "revue" de cet article (pas encore publiée), on relie ce modèle à un modèle de systèmes à compteurs plus connu.

Dans le reste de cette section, on détaille la façon dont sont présentés ces résultats dans chacune des deux parties de cette thèse.

—•—

La première partie est consacrée à l'étude des logiques arithmétiques permettant de représenter des ensembles infinis de variables. Ces variables servent à modéliser des phénomènes discrets, notamment par des compteurs à valeurs entières, ainsi que des phénomènes continus, notamment par des horloges à valeurs réelles (ou rationnelles). Lorsqu'une logique permet de représenter à la fois des variables entières et des variables réelles, on dit que c'est une logique *mixte*.

Voici la liste des sections présentant les résultats obtenus dans la première partie de cette thèse :

– 1 – Dans la section 2.2, on définit un opérateur permettant de décomposer les logiques réelles ou mixtes en logiques entières d'une part, et en logiques décimales d'autre part. Notons que l'on utilise le terme "décimal" pour désigner les nombres dans le sous-ensemble réel $[0, 1[$.

– 2 – Dans la section 2.3, on utilise cet opérateur pour décomposer trois logiques mixtes connues, permettant de les caractériser.

– 3 – Dans la section 2.4, on définit deux nouvelles logiques mixtes grâce aux nouvelles perspectives engendrées par notre opérateur de décomposition. On montre que l'une de ces logiques, appelée RDM, est plus expressive que les autres, tout en restant décidable.

– 4 – Dans la section 3.4, on présente notre implémentation de l'opérateur de décomposition, dans un prototype appelé IDS.

Les résultats énoncés dans les points 1, 2, et 4 sont publiés dans [BFL08], tandis que ceux du point 3 sont en cours de rédaction pour une revue.

—•—

La deuxième partie de cette thèse est consacrée à l'étude des modèles de systèmes infinis. Ces systèmes ont la particularité de manipuler des variables soit entières, soit réelles, soit mixtes (i.e. les deux à la fois). Les modèles étudiés dans cette thèse sont principalement des systèmes à compteurs et des systèmes temporisés, et sont parfois les deux en même temps.

Cette deuxième partie peut être lue indépendamment de la première ; toutefois, elle nécessite que l'on ait pris connaissance des préliminaires sur les logiques (section 1.2).

Voici la liste des sections présentant les résultats obtenus dans la deuxième partie de cette thèse :

– 5 – Dans la section 4.5, on définit les systèmes à compteurs mixtes, et on montre que la théorie de l'accélération développée dans [Ler03] s'étend directement à ces systèmes.

– 6 – Dans la section 4.6, on compare les différents systèmes à compteurs étudiés dans cette thèse. Dans la section 5.3.3, on donne une comparaison partielle des différents systèmes temporisés étudiés dans cette thèse.

– 7 – Dans la section 5.4, on définit un nouveau modèle de systèmes à compteurs temporisés, appelé TCS. On montre comment analyser ces TCS, et on en exhibe trois sous-classes décidables.

– 8 – Dans les sections 6.1 et 6.2, on clarifie et on étend la définition d'un modèle de systèmes peu commun, en motivant son étude. On appelle ces systèmes DCM : ce sont des machines à choix non-déterministe d'incrément dense, qui se situent en quelque sorte "à cheval" entre les systèmes à compteurs et les systèmes temporisés.

– 9 – Dans la section 6.3, on donne de nouveaux résultats de décidabilité pour les DCM, et on donne une caractérisation logique d'une sous-classe des DCM dans la section 6.4.1.

– 10 – Dans la section 6.4.2, on compare les DCM à un modèle de systèmes à compteurs relationnels.

Les résultats présentés dans les points 5 et 6 sont mineurs et ne font pas l'objet de publications. Les résultats détaillés au point 7 sont publiés dans [BFS09], et ceux des points 8 et 9 sont publiés dans [BFSP09]. Le résultat énoncé dans le point 10 sera probablement publié dans une version "revue" de [BFSP09], préparée très prochainement.

Table des matières

Table des figures

Chapitre 1

Préliminaires

1.1 Notations

Nombres. Dans cette thèse, $\mathbb{R}$ représente l'ensemble des nombres réels, $\mathbb{R}_+$ l'ensemble des nombres réels non-négatifs, $\mathbb{Q}$ l'ensemble des nombres rationnels, $\mathbb{Q}_+$ l'ensemble des nombres rationnels non-négatifs, $\mathbb{Z}$ l'ensemble des nombres entiers relatifs, et $\mathbb{N}$ l'ensemble des nombres entiers naturels.

Ensembles et vecteurs. Les symboles en majuscules (eg. X) désignent en général des ensembles, et les minuscules (eg. x) désignent des éléments de ces ensembles. Les symboles en caractères gras (eg. $\mathbf{x}$) désignent des vecteurs, et les symboles indicés (eg. x_i) désignent des composantes de ces vecteurs. Parfois, pour faciliter la lecture, on écrira x au lieu de x_i (sans ambigüité).

Variables. Les variables sont représentées sous forme d'ensemble. Sauf précision contraire, $n \in \mathbb{N}$ est le nombre de variables (que ce soit pour une logique, ou pour les compteurs d'un système, par exemple). Une valuation des variables sera simplement une affectation d'une valeur numérique à chacune des variables, le plus souvent sous la forme d'un vecteur d'entiers ou de réels.

Intervalles. Un intervalle fermé entre deux nombres a et b est noté $[a, b]$, tandis qu'un intervalle ouvert entre ces deux nombres sera noté $]a, b[$. De même, si a est compris dans l'intervalle mais pas b, alors on notera $[a, b[$ (et réciproquement, $]a, b]$).

Suites. Une suite u, indicée par un ensemble I d'entiers, est notée $u = (u_i)_{i \in I}$. Quand I est clairement défini par le contexte, on notera la suite $(u_i)_i$. La suite $(u_i)_{i \in I}$ est finie (respectivement, infinie) si et seulement si I est fini (respectivement, infini).

Fonctions. Une fonction $f : D \longrightarrow E$ est une relation dans $D \times E$ telle que pour tout $d \in D$, il existe au plus un élément $e \in E$ vérifiant $(d, e) \in f$. On note $f(d) = e$ cet élément et E^D l'ensemble des fonctions de D vers E. Étant donnée une fonction $f : D \longrightarrow E$, on définit son domaine $\mathsf{dom}(f) = \{d \in D \mid \exists e \in E \text{ tel que } e = f(d)\}$. On dit qu'une fonction $f' : D' \longrightarrow E'$ étend une fonction $f : D \longrightarrow E$ si $D \subseteq D'$ et $E \subseteq E'$ et si pour tout $d \in D$, on a $f'(d) = f(d)$. La restriction d'une fonction $f : D \longrightarrow E$ à $D_0 \subseteq D$ est la fonction notée $f_{|D_0} : D_0 \longrightarrow Y$ définie par $\mathsf{dom}(f_{|D_0}) = D_0 \cap \mathsf{dom}(f)$ et pour tout $x \in \mathsf{dom}(f_{|D_0})$, on a $f_{|D_0}(x) = f(x)$. Une fonction $f : D \longrightarrow E$ est une fonction totale si $\mathsf{dom}(f) = D$; dans le cas contraire, on parle de fonction partielle. On dit qu'une fonction $f : D \longrightarrow E$ est :

- injective, si pour tout $d, d' \in D$, $f(d) = f(d')$ implique $d = d'$,
- surjective, si pour tout $e \in E$, il existe $d \in D$ tel que $f(d) = e$,
- bijective, si f est injective et surjective.

1.2 Logiques arithmétiques du premier ordre

Afin de représenter des ensembles de nombres ou de vecteurs de nombres, on utilise des logiques. Ces ensembles que l'on manipule étant souvent infinis ou difficilement descriptibles de manière simple et concise, on se base donc sur le formalisme des logiques et du calcul des prédicats. En effet, on cherche à représenter et étudier de tels ensembles de vecteurs numériques ; on les décrit alors comme les solutions de formules, et on s'intéresse à caractériser et classifier ces formules en grandes familles qu'on appelle des *logiques*. On compare alors ces logiques selon leur expressivité.

Les logiques étudiées dans cette thèse sont toujours *du premier ordre*, c'est-à-dire que les variables sont des éléments, mais pas des ensembles. Ces logiques sont également dites *arithmétiques*, puisque les variables sont des nombres, avec comme opération de base l'addition, et parfois la multiplication. On utilisera les relations classiques pour l'ordre et l'égalité, ainsi que d'autres prédicats, détaillés dans la première partie de cette thèse. On parlera parfois d'arithmétique au lieu de logique arithmétique, indifféremment.

Formellement, soit X un ensemble de n variables. Soit $\langle \mathcal{D}, P_1, \ldots, P_k \rangle$ une structure, où $\mathcal{D}$ est le domaine de valuation des variables (typiquement $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$, etc.) et $P_1, \ldots, P_k$ sont des prédicats (typiquement, $\leq$, $=$, $+$, etc.). On définit alors la *logique* sur cette structure $\langle \mathcal{D}, P_1, \ldots, P_k \rangle$, notée FO $(\mathcal{D}, P_1, \ldots, P_k)$, comme étant l'ensemble des formules que l'on peut écrire au moyen de variables, des prédicats $P_1, \ldots, P_k$, des connecteurs booléens (`ou`, `et`, `non`), et des quantificateurs (existentiel, universel).

Prenons un exemple, qui est à la base de la plupart des logiques étudiées dans cette thèse : la logique de Presburger. Les autres logiques étudiées ici ne seront pas autant

détaillées, mais les notions suivantes (variables libres, satisfiabilité, etc.) seront très similaires.

L'arithmétique de Presburger [Pre91], définie en 1929, est la logique du premier ordre sur les entiers, avec comme seuls prédicats l'addition et l'égalité ; on la note FO $(\mathbb{N}, +, =)$. Comme on le verra par la suite, on pourra exprimer l'ordre habituel à partir de ces prédicats, mais pas la multiplication. L'ensemble des formules de l'arithmétique de Presburger peut être décrit par la grammaire suivante, où t est un terme, ϕ est une formule, et x est une variable :

$$\begin{array}{rcl} t & ::= & 0 \mid 1 \mid x \mid t + t \\ \phi & ::= & t = t \mid \neg\phi \mid \phi \vee \phi \mid \exists x.\phi \end{array}$$

L'ensemble $\mathsf{var}(\phi)$ des variables libres d'une formule de Presburger ϕ est défini par induction de la façon suivante :

- $\mathsf{var}(0) = \mathsf{var}(1) = \emptyset$
- $\mathsf{var}(x) = \{x\}$
- $\mathsf{var}(t + t') = \mathsf{var}(t) \cup \mathsf{var}(t')$
- $\mathsf{var}(t = t') = \mathsf{var}(t) \cup \mathsf{var}(t')$
- $\mathsf{var}(\neg\phi) = \mathsf{var}(\phi)$
- $\mathsf{var}(\phi \vee \phi') = \mathsf{var}(\phi) \cup \mathsf{var}(\phi')$
- $\mathsf{var}(\exists x.\phi) = \mathsf{var}(\phi) \setminus \{x\}$

On note $\mathsf{Presb}(X)$, l'ensemble de formules de Presburger qui ont leurs variables libres dans X.

Soient une formule de Presburger ϕ et une fonction $f : \mathsf{var}(\phi) \longrightarrow \mathbb{N}$. On définit la fonction App_f qui, à chaque terme de ϕ, associe un entier de la façon suivante :

- $\mathsf{App}_f(0) = 0$
- $\mathsf{App}_f(1) = 1$
- $\mathsf{App}_f(x) = f(x)$
- $\mathsf{App}_f(t + t') = \mathsf{App}_f(t) + \mathsf{App}_f(t')$

Grâce à cette fonction App, on peut définir la relation de satisfiabilité $f \models \phi$ par :

- $f \models t = t'$ si et seulement si $\mathsf{App}_f(t) = \mathsf{App}_f(t')$
- $f \models \neg\phi$ si et seulement si $f \not\models \phi$
- $f \models \phi \vee \phi'$ si et seulement si $f \models \phi$ ou $f \models \phi'$ (ou $f \models \phi$ et $f \models \phi'$)
- $f \models \exists x.\phi$ si et seulement s'il existe $f' : \mathsf{var}(\phi) \longrightarrow \mathbb{N}$ étendant f et telle que $f' \models \phi$

Deux formules de Presburger ϕ et ϕ' sont équivalentes (noté $\phi \equiv \phi'$) si pour toute fonction $f : \mathsf{var}(\phi) \cup \mathsf{var}(\phi') \longrightarrow \mathbb{N}$, on a $f_{|\mathsf{var}(\phi)} \models \phi$ si et seulement si $f_{|\mathsf{var}(\phi')} \models \phi'$.

On interprète ainsi les formules de Presburger sur l'ensemble $\mathbb{N}$ des entiers naturels. On peut ensuite étendre la grammaire avec les autres opérateurs logiques classiques, comme le "`et`" (noté $\wedge$) qui se définit facilement par $\phi \wedge \phi' \equiv \neg(\neg\phi \vee \neg\phi')$, ou encore le quantificateur universel $\forall$ défini par $\forall x.\phi \equiv \neg\exists x.\neg\phi$. On note $true$ la valeur de vérité, qui peut être définie par $true \equiv \exists x.x = x$; on note sa négation $false$. De la même façon, on peut ajouter à la grammaire d'autres opérateurs arithmétiques classiques, comme $\neq$,$<$, $\leq$, $\geq$ ou $>$, qui peuvent aussi être définis dans l'arithmétique de Presburger. Par exemple, $t < t' \equiv \exists x.(t' = t + x) \wedge \neg(x = 0)$. Cependant, on ne peut pas simuler la multiplication dans FO $(\mathbb{N}, +, =)$, mais seulement la multiplication d'une variable par une constante : par exemple, $3.x \equiv x + x + x$.

Le problème auquel on s'intéresse pour toute logique est celui de sa satisfiabilité, c'est-à-dire savoir décider s'il existe, pour une formule donnée, une valuation des variables rendant cette formule vraie. Plus formellement, voici comment on pose le problème de satisfiabilité d'une logique $\mathscr{L} = \text{FO}(\mathcal{D}, P_1, \ldots, P_k)$:

Problème 1.1. *Problème de satisfiabilité dans $\mathscr{L}$:*
Données : *Une formule ϕ de $\mathscr{L}$.*
Question : *Existe-t-il une fonction $f : \mathsf{var}(\phi) \longrightarrow \mathcal{D}$ telle que $f \models \phi$?*

Remarque : On dira souvent, par abus de langage, qu'une logique $\mathscr{L}$ est décidable ; cela signifiera que le problème de satisfiabilité est décidable quelle que soit la formule ϕ de $\mathscr{L}$ considérée.

La logique de Presburger est souvent utilisée pour la vérification de systèmes à compteurs, principalement parce qu'elle est décidable [Pre91] et qu'elle est particulièrement adaptée au mécanisme des compteurs, comme on le verra par la suite (notamment dans le chapitre 4). En revanche, si l'on rajoute la possibilité de multiplier les variables entières, c'est-à-dire si l'on autorise des termes de la forme $t * t'$, alors le problème de satisfiabilité devient indécidable.

Voyons à présent la notion de définissabilité d'un ensemble dans une logique. Là encore, on détaille les notions sur l'exemple de la logique de Presburger, mais ces notions (interprétation, définissabilité, etc.) sont similaires à celles des autres logiques que nous verrons par la suite.

Soit $\mathbf{a} \in \mathbb{N}^n$ un vecteur. On définit la fonction $f_{\mathbf{a}} : X \longrightarrow \mathbb{N}$ telle que pour tout $i \in [1, n]$, $f_{\mathbf{a}}(x_i) = \mathbf{a}(i)$. Si ϕ est une formule dans $\mathsf{Presb}(X)$, son *interprétation* $[\![\phi]\!]_X$ est définie par :

$$[\![\phi]\!]_X = \{\mathbf{a} \in \mathbb{N}^n \mid f_{\mathbf{a}} \models \phi\}$$

En d'autres termes, l'interprétation d'une formule est l'ensemble des valuations de ses variables libres qui la rendent vraie. On écrit parfois $[\![\phi]\!]$ au lieu de $[\![\phi]\!]_X$ lorsqu'il n'y a pas de confusion possible sur les variables libres X considérées. On peut écrire, informellement, que ϕ est vraie pour **a** si **a** $\models \phi$.

Définition 1.2 (Ensemble définissable dans Presburger)**.** *Soit $n \in \mathbb{N}$. Un ensemble $E \subseteq \mathbb{N}^n$ est définissable dans Presburger si et seulement s'il existe des variables $X = \{x_1, \ldots, x_n\}$ et une formule de Presburger $\phi \in \mathsf{Presb}(X)$ telles que $E = [\![\phi]\!]_X$.*

Une propriété intéressante de la logique de Presburger est qu'elle permet de définir tous les ensembles semi-linéaires. On rappelle qu'un ensemble semi-linéaire est une union finie d'ensembles linéaires, et qu'un ensemble $E \subseteq \mathbb{N}^n$ est linéaire s'il existe $k+1$ vecteurs $\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_k \in \mathbb{N}^n$ tels que $E = \{\mathbf{v} \in \mathbb{N}^n \mid \mathbf{v} = \mathbf{v}_0 + \lambda_1.\mathbf{v}_1 + \ldots + \lambda_k.\mathbf{v}_k$ avec $\lambda_i \in \mathbb{N}$ pour tout $i \in [1,k]\}$.

Théorème 1.3. *[GS66] Soit $E \subseteq \mathbb{N}^n$. E est définissable dans Presburger si et seulement si E est semi-linéaire.*

Exemple 1.4. *L'ensemble E_1 des entiers pairs est définissable dans Presburger. En effet, si l'on considère la formule $\phi_1 \equiv \exists y.x = y + y$, on a bien $E_1 = [\![\phi_1]\!]$.*

La proposition suivante fournit un exemple d'ensemble qui n'est pas définissable dans Presburger :

Lemme 1.5. *L'ensemble $E_2 = \{2^n \mid n \in \mathbb{N}\}$ des puissances de 2 n'est pas définissable dans Presburger.*

Démonstration. Raisonnons par l'absurde : supposons que l'ensemble $E_2 = \{2^n \mid n \in \mathbb{N}\}$ soit définissable dans Presburger. D'après le théorème 1.3, E_2 serait semi-linéaire, donc une union finie d'ensembles linéaires. Comme E_2 est infini, on en déduit qu'il existe au moins un ensemble linéaire infini $L \subseteq E_2$. Puisque L est linéaire, il existe $a_0, \ldots, a_k \in \mathbb{N}$ tel que $L = \{a_0 + \lambda_1.a_1 + \ldots + \lambda_k.a_k \mid \forall i \in [1,k], \lambda_i \in \mathbb{N}\}$. Prenons $a_1 = \max(\{a_i \mid i \in [1,k]\})$. Comme L est infini, on a forcément $a_1 > 0$, et de plus, il existe $n \in \mathbb{N}$ tel que $2^n \in L$ et $a_1 < 2^n$. Par définition de L, $2^n + a_1 \in L$ et donc il existe m tel que $2^n + a_1 = 2^m$ avec $n+1 \leq m$. Comme $a_1 > 0$, on en déduit $2^n \leq a_1$, ce qui contredit $a_1 < 2^n$. □

Étant donné un ensemble $X = \{x_1, \ldots, x_n\}$ de variables, on note $X' = \{x'_1, \ldots, x'_n\}$ l'ensemble des variables "primées" obtenu à partir de X et tel que $X \cap X' = \emptyset$, et $\mathsf{Presb}(X, X')$ l'ensemble des formules de Presburger ϕ telles que $\mathsf{var}(\phi) \subseteq X \cup X'$. Pour deux fonctions $f : X \longrightarrow \mathbb{N}$ et $f' : X' \to \mathbb{N}$ ainsi qu'une formule $\phi \in \mathsf{Presb}(X, X')$, on note $(f, f') \models \phi$ le fait que la fonction $g : X \cup X' \longrightarrow \mathbb{N}$, définie par $g_{|X} = f$ et $g_{|X'} = f'$, vérifie $g \models \phi$. On note ainsi $[\![\phi]\!]_{X,X'} = \{(\mathbf{a}, \mathbf{a}') \in \mathbb{N}^n \times \mathbb{N}^n \mid$

$(f_{\mathbf{a}}, f_{\mathbf{a}'}) \models \phi\}$.

Précisons la définition 1.2 dans le cas particulier des relations binaires :

Définition 1.6 (Relation binaire définissable dans Presburger)**.** *Une relation binaire $R \subseteq \mathbb{N}^n \times \mathbb{N}^n$ est définissable dans Presburger si et seulement s'il existe une formule de Presburger $\phi \in \mathsf{Presb}(X, X')$ telle que $R = [\![\phi]\!]_{X,X'}$, avec $X = \{x_1, \ldots, x_n\}$ et $X' = \{x'_1, \ldots, x'_n\}$.*

Cette définition permet alors de passer à l'étape suivante dans la vérification de systèmes : on va en effet décrire les changements d'état d'un système au moyen de formules issues d'une logique $\mathscr{L}(X, X')$, X symbolisant la valeur des variables avant ce changement, et X', après. Souvent, on identifiera la relation binaire à la formule qui la définit ; il y a en effet une différence de type, mais elle est sufisamment minime pour que l'on puisse la négliger et ainsi gagner en lisibilité. De façon générale, on identifiera parfois une logique avec les ensembles qu'elle peut représenter.

1.3 Modèles de systèmes et accessibilité

Les modèles de systèmes décrits dans cette thèse concernent pour la plupart des systèmes à compteurs et/ou des systèmes temporisés. Un système est généralement décrit par deux modèles : un pour la syntaxe, et l'autre pour la sémantique.

La syntaxe décrit la structure de contrôle du système, sous forme d'un graphe à états et transitions, comme une machine à compteurs, un automate temporisé, ou un réseau de Petri, par exemple. Graphiquement, les états sont représentés par des cercles, et les transitions par des flèches reliant ces cercles. Un état modélise l'aspect statique dans lequel le système se trouve, tandis qu'une transition modélise la dynamique du système, en le faisant passer d'un état à un autre. À la base, ce modèle est celui d'un automate fini ; mais leur expressivité étant très limitée, on y ajoute des variables, comme des compteurs, des horloges, des piles, des files, etc. Dans cette thèse, on étudie divers modèles de systèmes, tous basés sur cette approche. La plupart du temps, on aura un ensemble fini d'états de contrôle, un ensemble fini de transitions reliant ces états de contrôle, ainsi qu'un ensemble de variables prenant leurs valeurs dans un ensemble donné.

La sémantique, elle, est définie par un *système de transitions*. En effet, les transitions définissent la dynamique du système, dont on peut analyser tous les cas possibles : c'est ce qu'on appelle le *comportement* du système. Un système de transitions sert donc à décrire tous ces comportements : il indique, à tout moment lors du fonctionnement du système, dans quel état il se trouve et quelles sont les valeurs des variables. C'est ce qu'on appelle une *configuration* : un état de contrôle et une valeur pour chaque variable.

Les comportements possibles du système sont donc des suites de configurations reliées par des transitions : une telle suite est appelée une *exécution* du système. Lorsque l'on ajoute des variables discrètes non-bornées (ou des variables continues) au modèle, alors l'ensemble de ses configurations devient infini !

Formellement, voici comment on définit un système générique et son système de transitions associé. Notons que cette définition très générale sera précisée pour chaque modèle dans la deuxième partie de cette thèse ; l'objectif ici est de définir les problèmes qui nous intéresseront sur de tels systèmes.

Soit X un ensemble de n variables. Une variable prend ses valeurs dans un ensemble $\mathcal{D}$ qui dépend du modèle de système considéré : typiquement, $\mathcal{D}$ peut être égal à $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}_+$, ou $\mathbb{R}_+$. On désigne par $\mathbf{v} \in \mathcal{D}^n$ une valuation des variables.

Soit $\mathscr{L}$ une logique donnée, par exemple Presburger, ou $\mathrm{FO}(\mathbb{R}, \mathbb{Z}, +, <)$, etc. Une formule $F(X, X')$ définie sur $\mathscr{L}$, avec $2n$ variables libres, est interprétée comme la transformation de X en X'. Une telle formule est donc une relation binaire, et définit les valeurs des variables *avant* et *après* l'exécution d'une transition étiquetée par $F(X, X')$. Souvent, on fait l'amalgame entre une variable x_i et sa valeur, lorsque le contexte est clair (et de même pour un ensemble de variables).

Définition 1.7. *Un $\mathscr{L}$-système à n variables est un quadruplet $\mathcal{S} = \langle Q, n, \mathscr{L}, T\rangle$ où :*
- *Q est un ensemble fini d'états de contrôle*
- *n est le nombre de variables*
- *$\mathscr{L}$ est la logique du système, donnant l'ensemble de valuation des variables et les formules permettant de les manipuler*
- *$T \subseteq Q \times \mathscr{L} \times Q$ est un ensemble fini de transitions étiquetées par des formules de $\mathscr{L}$*

Remarque : Le plus souvent, n et $\mathscr{L}$ seront omis, puisqu'ils seront clairs dans le contexte ; on parlera alors d'un système $\mathcal{S} = \langle Q, T\rangle$.

On appelle *structure de contrôle* d'un système $\langle Q, T\rangle$ le graphe orienté induit par Q et T : l'ensemble des sommets est Q, et l'ensemble des arcs est T, dans lequel on omet les étiquettes (on peut aussi laisser les étiquettes, mais ne pas les interpréter).

Visuellement, on représente habituellement un système en se basant sur sa structure de contrôle. L'exemple sur la figure 1.1 représente un système $\mathcal{S} = \langle Q, T\rangle$ où $Q = \{q_1, q_2\}$ et $T = \{(q_1, \phi_1, q_2), (q_2, \phi_2, q_2)\}$.

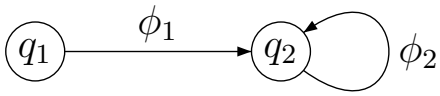


FIG. 1.1: Un exemple de système

Une configuration de $\mathcal{S}$ est un couple $(q, \mathbf{v})$ tel que $q \in Q$ et $\mathbf{v} \in \mathcal{D}^n$ est une valuation des variables. Le comportement du système, "navigant" entre ces configurations, est donné par un système de transitions.

Définition 1.8. *Un système de transitions d'un système $\mathcal{S} = \langle Q, T \rangle$ est un couple $TS(\mathcal{S}) = \langle C, \rightarrow \rangle$ où :*

- *$C = Q \times \mathcal{D}^n$ est l'ensemble des configurations de $\mathcal{S}$*
- *$\rightarrow \subseteq C \times T \times C$ est une relation telle que pour toutes $(q, \mathbf{v}), (q', \mathbf{v}') \in C$ et pour toute $t \in T$, on a $(q, \mathbf{v}) \xrightarrow{t} (q', \mathbf{v}')$ si et seulement si $t = (q, \phi, q')$ et $(\mathbf{v}, \mathbf{v}') \models \phi$*

Remarque : La notation standard $(q, \mathbf{v}) \xrightarrow{t} (q', \mathbf{v}')$ signifie $\Big((q, \mathbf{v}), t, (q', \mathbf{v}')\Big) \in \rightarrow$, et sera utilisée fréquemment tout au long de cette thèse, pour divers systèmes de transitions.

Remarque : On définit parfois la relation $\rightarrow$ dans $C \times \mathscr{L} \times C$ au lieu de $C \times T \times C$. En effet, $T \subseteq Q \times \mathscr{L} \times Q$: la seule information disparaissant avec cette définition sont les états de contrôle, mais ils sont déjà contenus dans les configurations, puisque $C = Q \times \mathcal{D}^n$.

Le système de transitions peut être vu comme un graphe orienté, ayant pour sommets les configurations, et pour arcs des instances de la relation de transition. Afin de simuler un comportement précis du système, on s'intéresse à un chemin dans ce graphe : un tel chemin est en fait une exécution du système. Une exécution peut alors être finie ou infinie ; parfois, on exclura un cas ou l'autre, selon le type de comportement auquel on s'intéresse.

En considérant une configuration de départ, les exécutions possibles qui en partent ne passent pas forcément par toutes les configurations ; et le plus souvent, beaucoup de configurations (voire une infinité !) ne font pas partie des exécutions considérées. On dit qu'une configuration $c \in C$ est *accessible* (ou *atteignable*) si et seulement s'il existe un chemin dans le graphe de transitions, qui débute par la configuration initiale donnée, et qui passe au moins une fois par c. L'ensemble des configurations accessibles depuis une

configuration initiale donnée est appelé l'*ensemble d'accessibilité* du système. Dans les systèmes étudiés dans cette thèse, l'ensemble des configurations est presque toujours infini : l'ensemble des configurations accessibles, quant à lui, peut être fini ou infini (cela dépend du système et de la configuration de départ).

Formellement, une *exécution finie* de $\mathcal{S}$ est une suite $(c_i)_{i\in\{0,\dots,l\}}$, $l \in \mathbb{N}$, telle qu'il existe des transitions $t_i \in T$ telles que $c_i \xrightarrow{t_i} c_{i+1}$ pour tout $i \in [0, l-1]$. Similairement, une *exécution infinie* de $\mathcal{S}$ est une suite $(c_i)_{i\geq 0}$ telle qu'il existe des transitions $t_i \in T$ telles que $c_i \xrightarrow{t_i} c_{i+1}$ pour tout $i \geq 0$.

Soit $c_0 \in C$ une configuration initiale donnée. Une configuration $c \in C$ est dite *accessible* s'il existe une exécution de la forme $(c_0, \dots, c)$. L'ensemble d'accessibilité de $\mathcal{S}$ relativement à c_0, noté $\mathsf{Reach}(\mathcal{S}, c_0)$, est défini comme l'ensemble des configurations accessibles dans $\mathcal{S}$ depuis c_0. Plus précisément, cet ensemble est obtenu en prenant la fermeture réflexive transitive de la relation de transition : $\mathsf{Reach}(\mathcal{S}, c_0) = \{c \mid c_0 \to^* c\}$. Par abus de langage, on parlera de l'ensemble d'accessibilité de $\mathcal{S}$, noté $\mathsf{Reach}(\mathcal{S})$, sans préciser la configuration initiale (qui sera implicite mais devra néanmoins être définie). Parfois, Reach sera noté Post^* : cela vient du fait que Post désigne l'image d'une configuration par une transition, et que l'on s'intéresse à sa fermeture réflexive transitive.

Pour poursuivre l'exemple de la figure 1.1, on représente sur la figure 1.2 le début de son système de transitions, en partant de la configuration initiale $(q_1, \mathbf{v}_0)$. Notons que pour l'instant, on ne précise pas de quel type sont les variables ni quelles sont les formules étiquetant les transitions : on ne manipule que des noms, pas des valeurs. Certaines des valuations suivantes peuvent être égales ; dans ce cas, si l'état de contrôle est le même, alors on fusionnera les configurations égales sur le graphe, et on pourra donc avoir des circuits. On a ici les valuations $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \dots$ telles que $(\mathbf{v}_0, \mathbf{v}_1) \models \phi_1$, $(\mathbf{v}_1, \mathbf{v}_2) \models \phi_2$, $(\mathbf{v}_2, \mathbf{v}_3) \models \phi_2$, etc. (le système peut a priori boucler indéfiniment sur q_2, tant qu'il produit des valuations de variables $\mathbf{v}_i$ telles qu'il existe $\mathbf{v}_j$ telles que $(\mathbf{v}_i, \mathbf{v}_j) \models \phi_2$). Par convention, on ne représente que les configurations accessibles depuis la configuration initiale donnée.

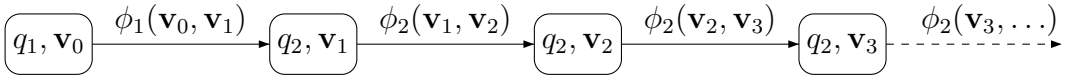


FIG. 1.2: Début du système de transitions de l'exemple de la figure 1.1

Notons qu'il est possible de définir un ensemble de configurations initiales, plutôt qu'une seule : il suffira d'utiliser des états de contrôle intermédiaires, ayant comme

source une seule et même configuration initiale fictive (que l'on rajoute). Au moyen de transitions, en partant de cette configuration fictive, on pourra donner aux variables les valeurs souhaitées dans les états de contrôle désirés, de sorte qu'il existe au moins une exécution passant par chacune des configurations initiales requises.

Un des problèmes principaux en vérification par model-checking consiste à déterminer si, pour un système donné, on peut calculer son ensemble d'accessibilité. Par "calculer" un tel ensemble infini, on entend qu'il existe un algorithme construisant une formule dont l'ensemble des solutions est précisément l'ensemble que l'on cherche à calculer. On dispose ainsi d'une représentation finie (la formule) pour un ensemble infini.

Plus précisément, il existe deux principales formes de ce problème d'accessibilité : un problème de décision, et un problème de calcul dans une logique donnée. La première variante, généralement plus simple à résoudre, consiste à décider si une configuration donnée est accessible depuis une configuration initiale. La deuxième variante, elle, consiste à calculer l'ensemble de tous les couples de configurations (c_1, c_2) telles c_2 est accessible depuis c_1, en construisant une formule dans une logique donnée.

Formellement, voici les énoncés de ces deux problèmes :

Problème 1.9. *Problème d'accessibilité (d'une configuration) :*
Données : *Un système $\mathcal{S}$ et deux configurations c_0 et c de $\mathcal{S}$.*
Question : *Est-ce que c est accessible depuis c_0 dans $\mathcal{S}$?*

Il existe des variantes de ce problème, notamment l'accessibilité d'un état de contrôle (et non pas d'une configuration complète) ; dans ce cas, on quantifie existentiellement sur les valeurs de variables dans la configuration à atteindre. Par la suite, lorsqu'une telle variante sera utilisée, l'énoncé formel sera détaillé.

Le deuxième problème, lui, nécessite de calculer *toutes* les configurations accessibles depuis *toutes* les configurations initiales possibles : on cherche donc à calculer la *relation d'accessibilité* $\rightarrow^*$, c'est-à-dire trouver une formule dont les solutions sont exactement l'ensemble des couples de configurations (c_1, c_2) tels que $c_1 \rightarrow^* c_2$.

Problème 1.10. *Problème d'accessibilité binaire :*
Données : *Un système $\mathcal{S}$ et une logique $\mathscr{L}$.*
Résultat : *Une formule ϕ de $\mathscr{L}$ telle que $[\![\phi]\!] = \rightarrow^*$.*

On s'intéresse généralement à la décidabilité du premier problème. Pour le deuxième problème, plus difficile, on s'intéresse à plusieurs variantes, selon les cas. Une première variante (1) est de savoir s'il existe une telle formule ϕ de $\mathscr{L}$, avant de chercher à la calculer. Une autre variante (2), étant données une formule ϕ et une relation $\rightarrow^*$, consiste à décider si $[\![\phi]\!] = \rightarrow^*$. Dans tous les cas, ce problème 1.10 est très difficile : par

exemple, pour les réseaux de Petri (voir section 4.2) et la logique de Presburger, on ne sait résoudre ni le problème d'accessibilité binaire, ni ses variantes (1) et (2) ! Pour le cas des automates binaires (voir section 3.1) et de la logique de Presburger, Jérôme Leroux a montré dans [Ler05] que ce problème est décidable (ce qui nécessite une preuve de 130 pages !).

Ces problèmes sont donc très difficiles à résoudre, et généralement indécidables sur la plupart des systèmes ; on s'intéressera alors à des sous-classes de systèmes pour lesquelles on obtient la décidabilité d'un de ces problèmes. Une autre variante du problème 1.10 consiste à ne pas se donner $\mathscr{L}$, mais plutôt à se demander si une telle logique décidable existe ; en pratique, on commence habituellement par chercher à calculer la formule ϕ, en utilisant le moins de prédicats possibles, puis on cherche la logique minimale requise pour pouvoir définir ϕ.

Remarque : Souvent, on dira par abus de langage qu'un système est décidable, pour dire en fait qu'il appartient à une classe de systèmes dont on sait que le problème d'accessibilité est décidable (de même que l'on dira qu'une logique est décidable pour parler de sa satisfaisabilité). De façon générale, lorsqu'on parle du problème d'accessibilité, on parle du problème 1.9.

Dans le cadre du model-checking, on s'intéresse souvent à de nombreuses variantes de ces problèmes et définitions. Le problème du model-checking fait lui-même appel à ces problèmes d'accessibilité : c'est pourquoi il est nécessaire d'étudier en profondeur l'accessibilité d'un modèle de systèmes, avant de s'intéresser à son model-checking. Ensuite, on peut définir des variantes, comme les conditions d'acceptation d'une exécution, ou l'étiquetage d'états ou de transitions par des propositions atomiques. De manière générale, pour étudier les modèles de systèmes plus en profondeur, on étiquette les transitions par des symboles d'un alphabet donné ; on caractérise alors le modèle par le langage (obtenu en ne gardant que les symboles) que génèrent ses exécutions. Cette thèse ne traite pas de ces variantes-là, mais pose plutôt les bases théoriques concernant les modèles décrits ici, principalement en termes d'accessibilité.

La finalité envisagée pour les modèles étudiés dans cette thèse est la vérification par model-checking. Ce type de vérification distingue globalement deux familles de propriétés que le système doit satisfaire : des propriétés de *sûreté*, et des propriétés de *vivacité*. Les propriétés de sûreté sont généralement plus faciles à vérifier, puisqu'elles nécessitent de garantir qu'une configuration n'appartient pas à l'ensemble d'accessibilité du système. Les propriétés de vivacité, elles, nécessitent de garantir que les exécutions passeront toujours (au sens "infiniment souvent") par certaines configurations ; il y a plusieurs cas de figure, selon les configurations à visiter, et selon que

l'on s'intéresse à au moins une exécution (quantification existentielle) ou à toutes les exécutions (quantification universelle). De façon générale, dans cette thèse, on se placera dans l'optique de la vérification de propriétés de sûreté.

Première partie

Logiques arithmétiques du premier ordre

Logiques arithmétiques du premier ordre : introduction

Les logiques étudiées dans cette thèse ont pour objectif de représenter des ensembles infinis de vecteurs numériques. On cherche donc à manipuler des formules qui puissent représenter le plus d'ensembles possibles, tout en restant dans des logiques décidables. En effet, l'objectif est de vérifier des modèles de systèmes, dont les valeurs de variables sont représentées par de telles formules ; il faut alors pouvoir calculer effectivement ces formules, pour pouvoir ensuite espérer que le model-checking reste faisable, déjà d'un point de vue théorique.

Cette partie est découpée en deux chapitres. Le premier présente des principales logiques réelles, entières, et mixtes, sur lesquelles on se base, puis se focalise sur l'étude des logiques mixtes, notamment à l'aide d'un opérateur séparant un ensemble réel donné en ensembles d'entiers et de décimaux. Le deuxième montre comment représenter efficacement des formules, en les encodant au moyen de quatre familles de mécanismes appelées "représentations symboliques", dans l'optique de les implémenter.

Chapitre 2

Décomposition de logiques mixtes

Ce chapitre se focalise sur les logiques mixtes, et propose un nouvel opérateur décomposant des ensembles de vecteurs réels en une combinaison de vecteurs d'entiers et de vecteurs de réels dans l'intervalle $[0, 1[$. Après avoir appliqué cet opérateur sur trois logiques mixtes utilisées en vérification, on propose une nouvelle logique les unifiant tout en restant décidable. Les travaux présentés dans les section 2.2 et 2.3 ont été publiés dans [BFL08]. Les travaux de la section 2.4, eux, n'ont pas encore été publiés. Avant tout cela, on commence par présenter les quelques logiques connues qui seront abordées dans cette thèse.

Notations. Dans ce chapitre, on note $\mathbb{D}$ l'intervalle $[0, 1[$, que l'on appelle l'ensemble des nombres décimaux, ou simplement, les *décimaux*[1]. Tout $d \in \mathbb{D}$ est appelé (nombre) *décimal*, et tout $D \subseteq \mathbb{D}$ est appelé *ensemble décimal*. De plus, les variables sont manipulées ici comme des vecteurs $\boldsymbol{x} = (x_1, \ldots, x_n)$, plutôt que comme des ensembles ; on fera trivialement le lien entre les deux, puisqu'un vecteur peut être vu comme un ensemble muni d'une relation d'ordre sur ses éléments. Pour un ensemble E, on note $\mathcal{P}(E)$ l'ensemble des parties de E.

2.1 Quelques logiques

Cette section présente les logiques décidables qui seront utilisées par la suite et dont les variables sont soit uniquement réelles (section 2.1.1), soit uniquement entières (section 2.1.2), ou les deux à la fois (section 2.1.3).

[1] On aurait pu choisir le mot *fractionnel* plutôt que *décimal*, mais il contient l'idée trompeuse que les décimaux devraient être rationnels (ce qui n'est pas le cas).

2.1.1 Logiques réelles

Dans cette section, les logiques sont de la forme FO $(\mathbb{R}, Pred_1, \ldots, Pred_k)$, et les prédicats $Pred_1, \ldots, Pred_k$ ne permettent pas de savoir si une variable a une valeur entière ou non. Les variables prennent donc leurs valeurs dans $\mathbb{R}$, et si par moments elles sont dans $\mathbb{N}$ ou $\mathbb{Z}$, on n'a aucun moyen de le savoir (les sections 2.1.3, 2.2, 2.3, et 2.4 sont consacrées au cas contraire).

Cette restriction à des variables "purement réelles" permet de considérer directement l'arithmétique réelle, notée FO $(\mathbb{R}, +, \times, \leq)$. En effet, on sait depuis Tarski [Tar48] que cette logique est décidable, bien qu'elle dispose de la multiplication. On s'intéressera également à la logique additive sur les réels, FO $(\mathbb{R}, +, \leq)$, qui est naturellement décidable ; on peut la considérer comme le "pendant réel" de la logique de Presburger.

Le fait que la multiplication réelle soit décidable peut surprendre, puisque la multiplication entière est connue pour rendre une logique additive indécidable. En fait, on peut ajouter à l'arithmétique réelle un prédicat unaire $P_2(x) \equiv \exists i \in \mathbb{Z}(x = 2^i)$ donnant les puissances de deux, tout en restant décidable. C'est ce qu'a montré Lou van den Dries en 1985 [vdD85], en donnant un argument (basé sur la théorie des modèles) pour la décidabilité de FO $(\mathbb{R}, +, \times, \leq, P_2)$. Cependant, la complexité n'est pas connue, puisqu'il n'a pas donné de procédure de décision ; c'est pourquoi Avigad et Yin ont donné dans [AY07] une procédure de décision, qui est primitive récursive (quoique non-élémentaire).

Bien qu'elles restent décidables, les logiques que l'on manipule peuvent atteindre des complexités peu encourageantes pour une utilisation pratique, comme c'est le cas pour FO $(\mathbb{R}, +, \times, \leq, P_2)$. Toutefois, cette logique peut encore être généralisée et rester décidable, et ce, à faible coût : plutôt que de se restreindre aux puissances entières de 2, on peut considérer les puissances entières de n'importe quelle constante entière. La deuxième logique réelle à laquelle nous nous intéressons, après l'arithmétique réelle, est donc FO $(\mathbb{R}, +, \times, \leq, P_b)$, où $b \in \mathbb{Z}$.

2.1.2 Logiques entières

Dans cette section, on s'intéresse aux logiques dont les variables prennent uniquement des valeurs entières. Pour un survey couvrant la plupart de ces logiques entières (et bien d'autres), voir [Bès02].

La première logique, à la base de toutes les logiques entières considérées dans cette thèse, est la logique de Presburger FO $(\mathbb{N}, +, =)$. Une logique qui lui est équivalente

est FO $(\mathbb{Z}, +, \leq)$, et correspond davantage au formalisme manipulé ici. En effet, la relation d'ordre $\leq$ peut s'exprimer dans Presburger : $x_1 \leq x_2 \equiv \exists x_3.(x_2 = x_1 + x_3)$. Réciproquement, l'égalité peut s'exprimer dans FO $(\mathbb{Z}, +, \leq)$, puisque $x_1 = x_2 \equiv x_1 \leq x_2 \wedge x_2 \leq x_1$. Enfin, on peut exprimer les entiers négatifs dans Presburger, bien qu'une variable ne puisse pas directement prendre une valeur négative ; on peut en effet encoder un entier négatif comme la différence de deux entiers positifs (voir la remarque suivante). Par la suite, on parlera alors de Presburger pour désigner indifféremment les logiques FO $(\mathbb{N}, +, =)$ ou FO $(\mathbb{Z}, +, \leq)$.

Remarque : De manière générale, on peut faire l'amalgame entre une logique sur les entiers naturels FO $(\mathbb{N}, \ldots)$ et cette même logique étendue aux entiers relatifs FO $(\mathbb{Z}, \ldots)$. En effet, on supposera que les parties de $\mathbb{Z}^n$ sont encodées dans les parties de $\mathbb{N}^{2n}$ simplement en prenant la différence de deux variables $x^+, x^- \in \mathbb{N}$ pour exprimer chaque variable $x \in \mathbb{Z}$. On considère alors la fonction affine $f_{\mathbb{N}\to\mathbb{Z}} : \mathbb{N}^{2n} \longrightarrow \mathbb{Z}^n$ telle que définie dans [Ler03], c'est-à-dire par $f_{\mathbb{N}\to\mathbb{Z}}(x_1^+, x_1^-, \ldots, x_n^+, x_n^-) = (x_1^+ - x_1^-, \ldots, x_n^+ - x_n^-)$. Les représentations obtenues à partir de cet encodage nécessiteront deux fois plus de variables, mais on s'intéresse plus à l'expressivité de ces représentations qu'à leur concision.

La deuxième logique entière qui nous intéresse ici est une extension de Presburger avec un prédicat donnant les puissances entières dans une base $b \geq 2$. Cette logique, notée FO $(\mathbb{Z}, +, \leq, V_b)$, a été montrée décidable dans [BHMV94], pour tout $b \geq 2$. Le prédicat V_b est défini comme la fonction $V_b : \mathbb{Z} \setminus \{0\} \longrightarrow \mathbb{Z}$ telle que $V_b(x) = b^j$, où $j \in \mathbb{N}$ est le plus grand entier tel que $b^{-j}x \in \mathbb{Z}$, avec la convention $V_b(0) = 1$. En d'autres termes, $V_b(x)$ donne la plus grande puissance entière positive de b divisant x.

Ce prédicat V_b a été étudié sous différents aspects. Un résultat intéressant est que l'on ne peut pas combiner n'importe comment ce prédicat dans deux bases différentes. En effet, soient deux bases entières b_1 et b_2. Deux entiers p, q sont dits *multiplicativement dépendants* s'il existe $k, l \geq 1$ tels que $p^k = q^l$. Si b_1 et b_2 sont multiplicativement dépendantes, alors FO $(\mathbb{Z}, +, \leq, V_{b_1}, V_{b_2})$ est décidable [BHMV94]. Par contre, si b_1 et b_2 sont multiplicativement indépendantes, alors FO $(\mathbb{Z}, +, \leq, V_{b_1}, V_{b_2})$ est indécidable [Vil92b, Vil92a].

D'autres variations de FO $(\mathbb{Z}, +, \leq, V_b)$ ont été étudiées : par exemple, la logique FO $(\mathbb{Z}, +, \leq, P_b)$, où $P_b(x)$ est le prédicat vrai si et seulement si x est une puissance de b (tel que défini dans la section précédente). Cette logique est décidable, et même incluse dans FO $(\mathbb{Z}, +, \leq, V_b)$. En effet, le prédicat $P_b(x)$ est exprimable dans FO $(\mathbb{Z}, +, \leq, V_b)$, car $P_b(x) \equiv V_b(x) = x$; par contre, on sait que V_b n'est pas définissable dans la logique FO $(\mathbb{Z}, +, \leq, P_b)$ grâce à [Sem79, BHMV94].

Une autre variation est FO $\left(\mathbb{Z}, +, \leq, b^{\mathbb{Z}}\right)$ (parfois notée FO $(\mathbb{Z}, +, \leq, b^x)$), elle aussi décidable [Sem83, CP86], mais incomparable avec FO $(\mathbb{Z}, +, \leq, V_b)$ [BHMV94]. Le prédicat $b^{\mathbb{Z}}$ est en fait la fonction exponentielle, donnant les puissances entières de b : on peut alors exprimer $b^x = y$, ce qui est plus expressif que $P_b(y)$, puisque l'on dispose alors du x tel que $b^x = y$. La combinaison de $b^{\mathbb{Z}}$ avec V_b, cependant, donne la logique indécidable FO $\left(\mathbb{Z}, +, \leq, V_b, b^{\mathbb{Z}}\right)$ [CP86].

2.1.3 Logiques mixtes : sur les réels et les entiers

Cette section présente les logiques *mixtes*, c'est-à-dire à la fois réelles et entières. Cette mixité des variables peut paraître étrange, puisque l'on a défini les logiques sur des structures $\langle \mathcal{D}, Pred_1, \ldots, Pred_k \rangle$, dont les variables ne prennent leur valeur que sur $\mathcal{D}$, qui est soit réel, soit entier. Les logiques mixtes sont en fait définies sur ces mêmes structures $\langle \mathcal{D}, Pred_1, \ldots, Pred_k \rangle$, avec la particularité que $\mathcal{D} = \mathbb{R}$ et $\exists i \in [1, k](Pred_i := \mathcal{Z})$, le prédicat $\mathcal{Z}(x)$ étant vrai si et seulement si $x \in \mathbb{Z}$.

On dispose donc de variables réelles, et on peut tester si toute valeur est entière ou non, ce qui revient en quelque sorte à avoir deux types de variables différents (en exploitant le fait que les entiers sont un sous-ensemble des réels). On notera alors ces logiques mixtes FO $(\mathbb{R}, \mathbb{Z}, Pred_1, \ldots, Pred_k)$, ce qui signifiera en réalité que l'un des prédicats est $\mathcal{Z}$ et que les variables sont dans $\mathbb{R}$.

La logique mixte la plus souvent rencontrée en vérification est probablement la logique additive mixte, notée FO $(\mathbb{R}, \mathbb{Z}, +, \leq)$. On peut la voir sous deux angles différents : comme Presburger étendue aux réels, ou comme l'arithmétique réelle avec entiers dans laquelle on enlève la multiplication afin de rester décidable. Elle est parfois appelée "arithmétique linéaire", ce qui montre bien qu'il ne manque que la multiplication à cette logique pour avoir toute l'arithmétique.

Cette logique peut admettre de nouveaux prédicats et rester décidable, comme c'est le cas notamment pour FO $(\mathbb{R}, \mathbb{Z}, +, \leq, X_b)$: cette logique sera étudiée en détail dans la section 2.3.3. Intuitivement, le prédicat $X_b(x, u, a)$ est vrai si et seulement si le bit en position indiquée par u vaut a, dans la décomposition de x en base b. La logique FO $(\mathbb{R}, \mathbb{Z}, +, \leq, X_b)$ est en quelque sorte la version "mixte" de la logique FO $(\mathbb{Z}, +, \leq, V_b)$, combinée avec FO $(\mathbb{R}, +, \leq, W_b)$: on verra dans la section 2.2 comment cette combinaison s'effectue. Les prédicats V_b et W_b sont similaires : V_b donne les puissances entières (comme expliqué dans la section précédente), et W_b donne les puissances négatives (i.e. dans l'intervalle $[0, 1[$), le tout dans une base entière $b \geq 2$. En fait, X_b est exprimable

en fonction de V_b et W_b, comme on le verra dans la section 2.3.3.

2.2 Un opérateur décomposant $\mathbb{R}$ en $\mathbb{Z} \uplus \mathbb{D}$

Dans cette section, on introduit un opérateur permettant de combiner entiers et décimaux pour représenter des réels ; on peut également le voir comme une décomposition des réels en une combinaison d'entiers et de décimaux. On commence par donner une motivation à la définition d'un tel opérateur. On montre ensuite qu'extraire les entiers contenus dans les réels peut générer une formule plus concise que si elle n'était composée que de réels. On définit ensuite formellement cet opérateur combinant des ensembles de vecteurs d'entiers et de décimaux, dans la section 2.2.2.

2.2.1 Motivation

Afin de motiver la définition de l'opérateur, on illustre son intérêt à travers un exemple. Cet exemple fait appel à des notions que l'on verra plus en détail par la suite : les automates temporisés (section 5.2) et les DBM (section 3.2.1).

Automates temporisés et DBM

Les automates temporisés constituent probablement le modèle le plus utilisé pour les systèmes dont le comportement dépend de l'écoulement du temps. Tels que définis dans [AD94], et comme détaillé dans la section 5.2, les automates temporisés sont essentiellement des automates finis munis de variables réelles, appelées horloges. Ces horloges modélisent le comportement temps-réel du système, en s'écoulant toutes à une même vitesse constante. Chaque horloge peut être comparée à une constante entière et éventuellement remise à 0 ; les seules autres actions permises dans les formules étiquetant un automate temporisé sont des contraintes diagonales, qui consistent à comparer la différence de deux horloges à une constante entière. Puisque les valeurs d'horloges ne sont pas bornées, l'espace des configurations d'un automate temporisé est infini ; c'est pourquoi on utilise une abstraction finie de cet espace, en utilisant le mécanisme des régions (voir le paragraphe page 106 et la section 5.4.2). En pratique, l'ensemble des régions n'est pas facilement calculable à cause de sa taille ; c'est pourquoi les régions sont encore abstraites en zones, ce qui constitue le mécanisme implémenté dans la plupart des outils de vérification de systèmes temporisés [BLL+95, LPY97, BDM+98, LL98].

Techniquement, dans ces outils, les zones sont représentées par des matrices à différences bornées, appelées DBM (pour Difference Bound Matrices) [BM83, Dil89]. Une DBM est une matrice carrée représentant les contraintes entre les n horloges, qui constituent une zone. Ici, on considère une DBM comme un couple $(\boldsymbol{c}, \prec)$, où $\boldsymbol{c} = (c_{i,j})_{0 \leq i,j \leq n}$, $\prec = (\prec_{i,j})_{0 \leq i,j \leq n}$, $c_{i,j} \in \mathbb{Z} \cup \{+\infty\}$, et $\prec_{i,j} \in \{\leq, <\}$. Chaque couple $(c_{i,j}, \prec_{i,j})$ est un élément de la matrice carrée, définissant un ensemble $R_{\boldsymbol{c},\prec}$ comme ceci :

$$R_{\boldsymbol{c},\prec} = \{\boldsymbol{r} \in \mathbb{R}^n \mid \bigwedge_{0 \leq i,j \leq n} r_i - r_j \prec_{i,j} c_{i,j}\}$$

Afin de représenter les contraintes non-diagonales (i.e. avec une seule horloge), on utilise une horloge fictive r_0 qui vaut toujours 0 (ce qui explique qu'il y ait $n+1$ éléments, et non pas n). Un élément $(c_{i,j}, \prec_{i,j})$ signifie que $r_i - r_j \prec_{i,j} c_{i,j}$, où r_i, r_j sont des horloges. Ainsi, chaque élément d'une DBM représente une contrainte diagonale, c'est-à-dire une différence bornée. Enfin, les termes $c_{i,j}$ ne constituant pas de véritable contrainte sont symbolisés par $c_{i,j} = +\infty$.

Extensions des DBM

Dans l'exemple suivant tiré de [BBFL03], l'automate temporisé possède deux horloges x et y, et un seul état de contrôle. Le comportement de l'automate est très simple : y est remise à 0 dès qu'elle atteint 1, alors que x progresse continuellement. Dans la configuration initiale, les horloges valent toutes les deux 0. De plus, un invariant dans l'état de contrôle s'assure que l'horloge y ne dépasse jamais 1.

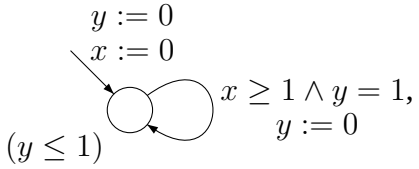
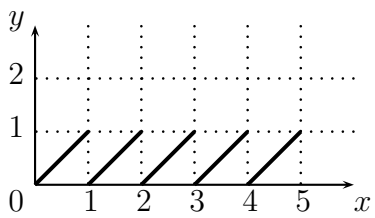


FIG. 2.1: Un automate temporisé

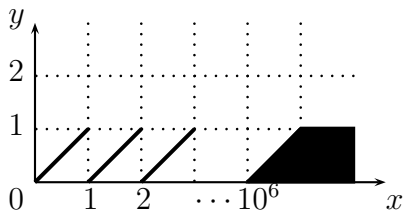
Le diagramme d'horloges associé à cet automate montre explicitement ce comportement :



On considère ici une analyse en avant classique [BLR05], en calculant les configurations $(état\ de\ contrôle, valuation\ d'horloges)$ accessibles depuis la configuration initiale (dans laquelle $x = y = 0$). Ensuite, on construit les zones correspondantes, chaque zone étant représentée par une DBM ; ici, on obtient un ensemble infini (quoique dénombrable) de DBM. Notons que dans cet exemple, $\prec$ est toujours $\leq$, et n'apparaît donc pas dans les matrices :

$$\left\{ \begin{array}{c|ccc} & 0 & x & y \\ 0 & 0 & -i & 0 \\ x & i+1 & 0 & i \\ y & 1 & -i & 0 \end{array} \right\}_{i \geq 0}$$

Afin de pouvoir calculer l'ensemble des configurations accessibles, il existe diverses techniques d'abstraction donnant un nombre fini de zones. L'abstraction la plus souvent utilisée en pratique est basée sur les constantes maximales : la valuation d'une horloge est fixée à ∞ dès qu'elle dépasse la plus grande constante à laquelle elle sera comparée. Sur l'exemple, si on ajoute une transition gardée par $x \geq 10^6$, menant à un autre état de contrôle, alors le diagramme d'horloges devient comme ceci :



Plus formellement, cette abstraction génère l'ensemble de DBM suivant :

$$\left\{ \left\{ \begin{array}{c|ccc} & 0 & x & y \\ 0 & 0 & -i & 0 \\ x & i+1 & 0 & i \\ y & 1 & -i & 0 \end{array} \right\}_{0 \leq i \leq 10^6}, \begin{array}{c|ccc} & 0 & x & y \\ 0 & 0 & \infty & 0 \\ x & \infty & 0 & \infty \\ y & 1 & -10^6 & 0 \end{array} \right\}$$

Cet ensemble de DBM est fini, mais reste très gros : $10^6 + 2$ matrices doivent être calculées et mémorisées, ce qui semble disproportionné, d'autant plus pour un exemple qui semble relativement simple. Dans [BBFL03], une abstraction de zones un peu plus élaborée est proposée : les constantes maximales de chaque horloge ne sont plus globales, mais dépendent de l'état de contrôle. Une autre abstraction est proposée dans [BBL06], en distinguant les constantes maximales qui sont des bornes supérieures de celles qui sont des bornes inférieures. Ces abstraction de zones sont les seules existantes

(à notre connaissance) ; dans chacune d'elles, le nombre de DBM dépend toujours fortement de la valeur des constantes maximales.

Écrire ici un tel nombre de DBM, infini (ou simplement très grand), aurait été impossible (ou trop fastidieux) si on n'avait pas utilisé une représentation paramétrique de ces DBM. C'est en fait un idée déjà utilisée par les CPDBM (Constrained Parametric DBM) [AAB00], qui sont la structure de données implémentée dans l'outil TREX [ABS01]. Les CPDBM sont en effet une version plus expressive des DBM, étendues de deux façons. Premièrement, on obtient les PDBM, dans lesquelles les constantes $c_{i,j}$ deviennent des paramètres $t_{i,j}$. Ces termes arithmétiques t sont définis par la grammaire $t ::= 0 \mid 1 \mid x \mid t - t \mid t + t \mid t \times t$, où x appartient à un ensemble de variables réelles. Deuxièmement, une PDBM devient une CPDBM lorsque ses termes sont contraints par des formules ϕ, du premier ordre, mais sans quantificateurs. Ces formules sont définies par $\phi ::= t \leq t \mid \neg\phi \mid \phi \vee \phi \mid Is\_int(t)$ (où le prédicat $Is\_int(t)$ est vrai si et seulement si t est un entier, et correspond donc au prédicat $\mathcal{Z}$ défini à la section 2.1.3). Chacun des deux ensembles de matrices ci-dessus est en fait une CPDBM.

N.B. : Dans le reste de cette section 2.2, le symbole $\times$ représente le produit cartésien (et non pas la multiplication).

On considère à présent un autre moyen de représenter l'ensemble des valeurs d'horloges accessibles. Sur le deuxième diagramme d'horloges montrant l'abstraction de zones (page 42), on remarque trois motifs : ⧸, ◢, et ■. On peut définir chaque motif comme suit : ⧸$= \{(x, y) \in [0, 1]^2 \mid x = y\}$, ◢$= \{(x, y) \in [0, 1]^2 \mid x \geq y\}$, et ■$= \{(x, y) \in [0, 1]^2\}$. Si l'on veut représenter le même ensemble que les zones abstraites précédemment, mais sans DBM, on peut exprimer la périodicité de chaque motif avec des entiers. Formellement, il suffit de prendre l'union de trois sommes d'ensembles en dimension 2 :

$$\begin{aligned} &\Big(\{0, \ldots, 10^6 - 1\} \times \{0\} + ⧸\Big) \\ \bigcup &\Big(\{10^6\} \times \{0\} + ◢\Big) \\ \bigcup &\Big(\{10^6 + 1, \ldots, \infty\} \times \{0\} + ■\Big) \end{aligned}$$

Cette dernière représentation d'ensemble d'accessibilité est plus concise que les DBM. En effet, représenter les zones par des DBM implique de mémoriser un nombre de matrices potentiellement très grand, dépendant de la constante maximale des horloges (un million, dans l'exemple). Cependant, en introduisant des entiers pour exprimer la périodicité, on peut réduire cette représentation à trois combinaisons d'intervalles. De plus, on peut même se passer de l'abstraction par constante maximale, et ainsi obtenir

une représentation exacte pour le même coût. Les CPDBM ont également ces avantages, mais sont généralement indécidables à cause de la multiplication. Pour une étude de l'expressivité des CPDBM, voir [DFvB05][2].

Précisons à présent la nouvelle représentation : on prend des unions finies de réels, chaque nombre réel étant décomposé en sommes d'entiers et de réels plus petits (les décimaux). Ces entiers et décimaux peuvent alors être définis en utilisant la quantification, l'addition, et les opérations booléennes.

En fait, cette nouvelle approche revient à représenter des ensembles de nombres réels en extrayant leurs composantes entières ; une caractéristique intéressante est que le fait d'ajouter des entiers à des ensembles de réels simplifie leur représentation et leur manipulation ! On pourrait penser qu'ajouter des entiers à une logique réelle du premier ordre la rendrait indécidable, mais la section 2.3.2 prouve le contraire. Tout d'abord, formalisons cette nouvelle représentation.

2.2.2 Composition d'entiers et de réels

Soient $\mathfrak{Z} \subseteq \mathcal{P}(\mathbb{Z}^n)$ et $\mathfrak{D} \subseteq \mathcal{P}(\mathbb{D}^n)$; on suppose que les vecteurs sont de taille n, tout comme les vecteurs de variables. On note $\mathfrak{Z} \uplus \mathfrak{D}$ la classe des vecteurs réels $R \subseteq \mathbb{R}^n$ telle que R puisse s'écrire $R = \bigcup_{i=1}^{p}(Z_i + D_i)$, avec $(Z_i, D_i) \in \mathfrak{Z} \times \mathfrak{D}$ et $p \geq 1$. Notons que le symbole $\uplus$ ne désigne pas l'union disjointe, comme c'est parfois le cas dans la littérature ; il n'y a pas de confusion possible, puisqu'aucune union disjointe n'est utilisée dans cette thèse.

Voici quelques exemples d'ensembles basiques auxquels on pourrait faire appel, écrits sous la forme d'unions finies de sommes d'entiers et de décimaux :

Exemple 2.1. *L'ensemble vide $\emptyset$ peut s'écrire $\emptyset + \emptyset$. L'ensemble $\mathbb{R}^n$ peut s'écrire $\mathbb{Z}^n + \mathbb{D}^n$. L'ensemble $\mathbb{Z}^n$ peut s'écrire $\mathbb{Z}^n + \{\mathbf{0}\}$.*

Exemple 2.2. *L'ensemble $R_= = \{\boldsymbol{r} \in \mathbb{R}^2 \mid r_1 = r_2\}$ peut s'écrire $\{\boldsymbol{z} \in \mathbb{Z}^2 \mid z_1 = z_2\} + \{\boldsymbol{d} \in \mathbb{D}^2 \mid d_1 = d_2\}$*

Exemple 2.3. *L'ensemble $R_\leq = \{\boldsymbol{r} \in \mathbb{R}^2 \mid r_1 \leq r_2\}$ peut s'écrire :*

$$\{\boldsymbol{z} \in \mathbb{Z}^2 \mid z_1 \leq z_2\} + \{\boldsymbol{d} \in \mathbb{D}^2 \mid d_1 \leq d_2\}$$
$$\bigcup \{\boldsymbol{z} \in \mathbb{Z}^2 \mid z_1 < z_2\} + \{\boldsymbol{d} \in \mathbb{D}^2 \mid d_1 > d_2\}$$

[2]Une précision : dans [DFvB05], les "Constrained PDBM" restreintes aux entiers sont appelées "Extended PDBM".

Exemple 2.4. *L'ensemble* $R_+ = \{\boldsymbol{r} \in \mathbb{R}^3 \mid r_1 + r_2 = r_3\}$ *peut s'écrire* $\bigcup_{c\in\{0,1\}}\{\boldsymbol{z} \in \mathbb{Z}^3 \mid z_1 + z_2 + c = z_3\} + \{\boldsymbol{d} \in \mathbb{D}^3 \mid d_1 + d_2 = d_3 + c\}$*, où* c *représente la retenue (*carry*, en anglais).*

Les limites de cette représentation peuvent être remarquées avec le contre-exemple suivant. Considérons l'ensemble $R_1 = \bigcup_{j=1}^{\infty}\Big(\{j\} + \left\{\frac{1}{j+1}\right\}\Big)$; notons que l'on utilise $j+1$ (et pas simplement j) afin d'éviter le cas où la partie décimale est $\frac{1}{j} = 1$ pour $j = 1$ (ce qui ne serait plus un décimal, car $1 \notin [0, 1[$). Cette représentation ne fonctionne apparemment pas pour un tel ensemble ; en effet, bien que R_1 soit une union de sommes d'entiers et de décimaux, cette union semble intrinsèquement infinie. On insiste sur la finitude de l'union dans cette représentation, principalement parce que l'un des objectifs est de l'implémenter, comme on le verra dans la section 3.4.

On s'intéresse maintenant à la stabilité de cette représentation. On prouve[3] que si $\mathfrak{Z} \subseteq \bigcup_{n\in\mathbb{N}} \mathcal{P}(\mathbb{Z}^n)$ et $\mathfrak{D} \subseteq \bigcup_{n\in\mathbb{N}} \mathcal{P}(\mathbb{D}^n)$ sont stables par les opérations classiques du premier ordre, alors la classe $\mathfrak{Z} \uplus \mathfrak{D} = \bigcup_{n\in\mathbb{N}} \mathfrak{Z}_n \uplus \mathfrak{D}_n$, où $\mathfrak{Z}_n = \mathfrak{Z} \cap \mathcal{P}(\mathbb{Z}^n)$ et $\mathfrak{D}_n = \mathfrak{D} \cap \mathcal{P}(\mathbb{D}^n)$, est elle aussi stable par ces opérations. Les opérations considérées ici sont les combinaisons booléennes (union, intersection, différence), le produit cartésien, la quantification, et la permutation. On utilise les définitions suivantes pour ces deux dernières opérations. La quantification est effectuée en projetant des variables du vecteur considéré : $\forall R \subseteq \mathbb{R}^n$, $\exists_i R = \{(r_1, \ldots, r_{i-1}, r_{i+1}, \ldots, r_n) \mid \exists r_i\ (r_1, \ldots, r_{i-1}, r_i, r_{i+1}, \ldots, r_n) \in R\}$. Une permutation π est simplement une fonction modifiant l'ordre des variables dans un vecteur : $\forall R \subseteq \mathbb{R}^n,\ \pi R = \{(r_{\pi(1)}, \ldots, r_{\pi(n)}) \mid (r_1, \ldots, r_n) \in R\}$. On utilise alors une définition générale pour la stabilité :

Définition 2.5. *Une classe* $\mathfrak{R} \subseteq \bigcup_{n\in\mathbb{N}} \mathcal{P}(\mathbb{R}^n)$ *est dite* stable *si elle est close par les opérations booléennes, le produit cartésien, la quantification, et la permutation.*

On remarque que dans cette représentation, prendre l'union de deux ensembles est trivial, puisqu'ils sont déjà des unions de parties entières et décimales. On remarque également que $(Z_1 + D_1) \cap (Z_2 + D_2) = (Z_1 \cap Z_2) + (D_1 \cap D_2)$ pour tout $Z_1, Z_2 \subseteq \mathbb{Z}^n$ et pour tout $D_1, D_2 \subseteq \mathbb{D}^n$; ainsi, la stabilité par union $\mathfrak{Z}_n \uplus \mathfrak{D}_n$ garantit la stabilité par intersection. On obtient la stabilité par différence grâce à l'égalité suivante : $(Z_1 + D_1) \setminus (Z_2 + D_2) = ((Z_1 \setminus Z_2) + D_1) \cup (Z_1 + (D_1 \setminus D_2))$. La stabilité par produit cartésien est donnée par $(Z_1 + D_1) \times (Z_2 + D_2) = (Z_1 \times Z_2) + (D_1 \times D_2)$. La stabilité par projection vient de $\exists_i R = (\exists_i Z) + (\exists_i D)$, où $R = Z + D$. Enfin, la stabilité par permutation est obtenue grâce à $\pi(Z + D) = (\pi Z) + (\pi D)$.

On a alors prouvé la proposition suivante, qui sera utilisée plus tard (notamment dans les théorèmes 2.9 et 2.13) :

[3] On utilise ici des *unions* de parties (et non plus simplement des parties) pour une raison purement technique. On a en effet besoin de modifier la dimension n des vecteurs lors de l'opération de projection.

Proposition 2.6 (Stabilité)**.** *La classe $\mathfrak{Z} \uplus \mathfrak{D}$ est stable si $\mathfrak{Z}$ et $\mathfrak{D}$ sont stables.*

On montre enfin que l'opérateur de décomposition préserve la décidabilité. En effet, les algorithmes permettant de calculer les opérations requises pour la preuve de la proposition 2.6 sont très simples, comme le montrent les égalités précédentes : ces opérations se distribuent sur l'opérateur $\uplus$. On rappelle qu'une classe (d'ensembles) est *récursive* si chacune de ses parties est représentable dans une logique décidable. Ainsi, si les ensembles dans $\mathfrak{Z}$ et dans $\mathfrak{D}$ sont récursifs, alors les ensembles dans $\mathfrak{Z} \uplus \mathfrak{D}$ sont eux aussi récursifs :

Proposition 2.7. *La classe $\mathfrak{Z} \uplus \mathfrak{D}$ est récursive si $\mathfrak{Z}$ et $\mathfrak{D}$ sont récursifs.*

Démonstration. Il suffit de ramener toute formule décrivant un ensemble dans $\mathfrak{Z} \uplus \mathfrak{D}$ à une union finie de sommes d'ensembles dans $\mathfrak{Z}$ et $\mathfrak{D}$. Pour ce faire on distribue toutes les opérations sur l'opérateur $\uplus$, comme indiqué précédemment : pour tous ensembles $E_1, E_2 \in \mathfrak{Z} \uplus \mathfrak{D}$, on utilise les encodages donnés à la page 46 pour décomposer $E_1 \cup E_2$, $E_1 \cap E_2$, $E_1 \setminus E_2$, $\exists_i E_1$, et πE_1 en formules entières et décimales. De plus, on utilise les encodages de l'exemple 2.1 pour décomposer l'ensemble vide (et son dual). □

2.3 Applications de l'opérateur de décomposition

Dans cette section, on applique successivement l'opérateur de décomposition sur trois logiques : la logique des CPDBM et ses variantes (section 2.3.1), la logique additive mixte (section 2.3.2), et la logique additive mixte étendue avec le prédicat X_b (section 2.3.3).

2.3.1 Décomposition de représentations basées sur les DBM

Une première application de l'opérateur de décomposition porte sur les extensions des DBM, qui nous permet de les caractériser. On note $\bigcup \mathrm{DBM}_{\mathbb{D}}$ pour désigner les unions finies de DBM-ensembles qui sont inclus dans $\mathbb{D}^n$. On remarque que $\bigcup \mathrm{DBM}_{\mathbb{D}}$ est stable par les opérations du premier ordre, grâce à l'élimination de quantificateurs de Fourier-Motzkin [Fou26, Mot51].

On définit une CP-DBM$_{\mathscr{L}}$ comme une DBM dans laquelle le vecteur $\boldsymbol{c}$ n'est plus une constante, mais un vecteur de paramètres contraints par une formule $\phi(\boldsymbol{c})$ définie dans une logique $\mathscr{L}$. Plus précisément, une CP-DBM$_{\mathscr{L}}$ est un couple $(\phi, \prec)$ qui représente un ensemble $R_{\phi,\prec}$ tel que :

$$R_{\phi,\prec} = \bigcup_{\boldsymbol{c} \models \phi} R_{\boldsymbol{c},\prec}$$

Telles qu'introduites dans [AAB00], les CPDBM correspondent aux CP-DBM$_{\mathscr{L}}$ dans lesquelles $\mathscr{L}$ est l'arithmétique du premier ordre, mais sans quantificateurs ; en particulier, la multiplication est autorisée dans cette logique. On étudie ici une autre variation des DBM : les CP-DBM$_+$, qui sont des CP-DBM$_{\mathscr{L}}$ pour lesquelles $\mathscr{L}$ est la logique Presburger FO $(\mathbb{Z}, +, \leq)$, qui est décidable. Les CP-DBM$_+$ sont en quelque sorte des CPDBM avec quantificateurs mais sans multiplication. On note $\bigcup$CP-DBM$_+$ pour désigner les unions finies de $R_{\boldsymbol{\phi},\prec}$, c'est-à-dire les unions finies d'ensembles représentables par CP-DBM$_+$.

On montre que les unions finies d'ensembles représentables par CP-DBM$_+$ sont en fait une combinaison d'ensembles définissables dans Presburger et dans les unions finies de DBM-ensembles inclus dans $\mathbb{D}^n$:

Proposition 2.8. $\bigcup$*CP-DBM*$_+ =$ *FO* $(\mathbb{Z}, +, \leq) \uplus \bigcup$*DBM*$_{\mathbb{D}}$

Démonstration. Prouvons d'abord l'inclusion $\supseteq$. Prenons une DBM $(\boldsymbol{c}, \prec)$ représentant un ensemble $D \subseteq \mathbb{D}^n$, ainsi qu'une formule de Presburger $\psi(\boldsymbol{x})$ représentant un ensemble $Z \subseteq \mathbb{Z}^n$; on prouve alors que $Z + D$ est un ensemble dans $\bigcup$CP-DBM$_+$. Remarquons que $\boldsymbol{r} \in Z + D$ si et seulement s'il existe $\boldsymbol{z} \in Z$ tel que $\boldsymbol{r} - \boldsymbol{z} \in D$. La condition $\boldsymbol{r} - \boldsymbol{z} \in D$ est équivalente à $\bigwedge_{0 \leq i,j \leq n} r_i - r_j \prec_{i,j} c_{i,j} + z_i - z_j$. Considérons la formule de Presburger $\psi(\boldsymbol{p}) := \exists \boldsymbol{z} \in \mathbb{Z}^n \; p_{i,j} = c_{i,j} + z_i - z_j$ et remarquons que $R_{\psi,\prec} = Z + D$. On vient donc de prouver l'inclusion $\supseteq$.

Pour l'inclusion $\subseteq$, considérons un ensemble représentable par CP-DBM$_+$, noté $R_{\boldsymbol{\phi},\prec}$. Soit $Z_{\boldsymbol{d}} = \mathbb{Z}^n \cap (R_{\boldsymbol{\phi},\prec} - \boldsymbol{d})$, indicé par $\boldsymbol{d} \in \mathbb{D}^n$. Remarquons que $Z_{\boldsymbol{d}}$ est en fait l'ensemble de vecteurs suivant :

$$Z_{\boldsymbol{d}} = \bigcup_{\boldsymbol{c} \models \phi} \left\{ \boldsymbol{z} \in \mathbb{Z}^n | \bigwedge_{0 \leq i,j \leq n} z_i - z_j \prec_{i,j} c_{i,j} + (d_j - d_i) \right\}$$

Puisque $d_j - d_i \in]-1, 1[$ et $z_i - z_j, c_{i,j} \in \mathbb{Z}$, on en déduit que $z_i - z_j \prec_{i,j} c_{i,j} + (d_j - d_i)$ est équivalent à $z_i - z_j \leq c_{i,j}$ si $d_i - d_j \prec_{i,j} 0$ et à $z_i - z_j \leq c_{i,j} - 1$ sinon. Étant donnée la matrice $\boldsymbol{m} = (m_{i,j})_{0 \leq i,j \leq n}$ telle que $m_{i,j} \in \{0, 1\}$ pour tout $0 \leq i, j \leq n$, on écrit $I_{\boldsymbol{m}}$ et $D_{\boldsymbol{m}}$ pour désigner les ensembles suivants :

$$I_{\boldsymbol{m}} = \{\boldsymbol{z} \in \mathbb{Z}^n \mid \exists \boldsymbol{c} \; \phi(\boldsymbol{c}) \wedge \bigwedge_{0 \leq i,j \leq n} z_i - z_j \leq c_{i,j} - m_{i,j}\}$$

$$D_{\boldsymbol{m}} = \{\boldsymbol{d} \in \mathbb{D}^n \mid \bigwedge_{0 \leq i,j \leq n} (d_i - d_j \prec_{i,j} 0 \iff m_{i,j} = 0)\}$$

Notons que $D_{\boldsymbol{m}}$ est un ensemble représentable par DBM, et que $Z_{\boldsymbol{d}} = I_{\boldsymbol{m}}$ pour tout $\boldsymbol{d} \in D_{\boldsymbol{m}}$. Or $\bigcup_{\boldsymbol{m}} D_{\boldsymbol{m}} = \mathbb{D}^n$, et donc $R_{\boldsymbol{\phi},\prec} = \bigcup_{\boldsymbol{d} \in \mathbb{D}^n} Z_{\boldsymbol{d}} + \{\boldsymbol{d}\} = \bigcup_{\boldsymbol{m}} I_{\boldsymbol{m}} + D_{\boldsymbol{m}}$. On a alors prouvé que $R_{\boldsymbol{\phi},\prec}$ est définissable dans FO $(\mathbb{Z}, +, \leq) \uplus \bigcup$DBM$_{\mathbb{D}}$. □

2.3.2 Décomposition de la logique additive mixte

On sait que l'arithmétique mixte est indécidable, mais qu'elle devient décidable si l'on enlève la multiplication. On obtient donc la logique additive mixte, dont la décidabilité a été suggérée par Büchi, puis prouvée dans [BRW98] en utilisant des automates binaires (voir section 3.1) et dans [Wei99] en utilisant une élimination de quantificateurs. Cette logique additive mixte est en fait une extension de Presburger aux réels (ou une restriction de l'artithmétique mixte sans la multiplication).

Cette logique FO $(\mathbb{R}, \mathbb{Z}, +, \leq)$ peut encoder des contraintes linéaires complexes qui combinent des variables entières et réelles. Dans cette section, on prouve que les ensembles définissables dans cette logique se décomposent en unions finies d'ensembles $Z+D$, où chaque Z est définissable dans FO $(\mathbb{Z}, +, \leq)$ et chaque D est définissable dans FO $(\mathbb{D}, +, \leq)$. Ce résultat montre que des contraintes linéaires complexes, combinant des variables entières et réelles, peuvent être décomposées en contraintes linéaires sur les entiers et en contraintes linéaires sur les réels (et même décimaux). Plus précisément, on prouve la décomposition suivante :

Théorème 2.9. *FO* $(\mathbb{R}, \mathbb{Z}, +, \leq) =$ *FO* $(\mathbb{Z}, +, \leq) \uplus$ *FO* $(\mathbb{D}, +, \leq)$

Démonstration. Observons tout d'abord que tout ensemble définissable dans la logique FO $(\mathbb{Z}, +, \leq) \uplus$ FO $(\mathbb{D}, +, \leq)$ est également définissable dans FO $(\mathbb{R}, \mathbb{Z}, +, \leq)$. De plus, les ensembles $\mathbb{R}$ et $\mathbb{Z}$, la fonction $+ : \mathbb{R}^2 \longrightarrow \mathbb{R}$, et le prédicat $\leq$, sont définissables dans FO $(\mathbb{Z}, +, \leq) \uplus$ FO $(\mathbb{D}, +, \leq)$, comme le montrent les exemples 2.1, 2.2, 2.3, et 2.4. De la stabilité par opérations du premier ordre, on obtient l'inclusion FO $(\mathbb{R}, \mathbb{Z}, +, \leq) \subseteq$ FO $(\mathbb{Z}, +, \leq) \uplus$ FO $(\mathbb{D}, +, \leq)$. On en déduit alors l'égalité. □

On rappelle que les ensembles définissables dans la logique de Presburger FO $(\mathbb{Z}, +, \leq)$ peuvent sont exactement les *ensembles linéaires* [GS66]. En fait, un ensemble $Z \subseteq \mathbb{Z}^n$ est définissable dans Presburger si et seulement s'il est égal à une union finie d'ensembles linéaires $\boldsymbol{b} + P^*$, où $\boldsymbol{b} \in \mathbb{Z}^n$, P est un sous-ensemble fini de $\mathbb{Z}^n$, et P^* décrit l'ensemble des sommes finies $\sum_{i=1}^{k} p_i$ avec $p_1, \ldots, p_k \in P$ et $k \in \mathbb{N}$. Cette caractérisation géométrique peut être étendue à la classe des ensembles définissables dans FO $(\mathbb{Z}, +, \leq) \uplus$ FO $(\mathbb{D}, +, \leq)$, en introduisant la classe des *ensembles polyédriques convexes* (voir notamment les travaux de Nicolas Halbwachs, par exemple [HMG06]). Un ensemble $C \subseteq \mathbb{R}^n$ est dit *polyédrique convexe* si C peut être défini par une conjonction finie de formules $\langle \alpha, \boldsymbol{x} \rangle \prec c$, où $\alpha \in \mathbb{Z}^n$, $\prec \in \{\leq, <\}$, et $c \in \mathbb{Z}$. En fait, une *élimination de quantificateurs de Fourier-Motzkin* [Fou26, Mot51] prouve qu'un ensemble $C \subseteq \mathbb{R}^n$ est définissable dans FO $(\mathbb{R}, +, \leq)$ si et seulement s'il est égal à une union finie d'ensembles polyédriques convexes. Dans [FL08], les auteurs ont montré la caractérisation géométrique suivante :

Proposition 2.10. *[FL08] Un ensemble $R \subseteq \mathbb{R}^n$ est définissable dans FO $(\mathbb{R}, \mathbb{Z}, +, \leq)$ si et seulement s'il est égal à une union finie d'ensembles de la forme $C+P^*$, où $C \subseteq \mathbb{R}^n$ est un ensemble polyédrique convexe et P est un sous-ensemble fini de $\mathbb{Z}^n$.*

2.3.3 Au-delà de la logique additive mixte : les RVA

On vient de montrer que la décomposition peut s'appliquer à FO $(\mathbb{R}, \mathbb{Z}, +, \leq)$ et à une de ses sous-logiques. On prouve maintenant qu'elle s'applique également à des logiques plus expressives. On prend le cas particulier des RVA (Real Vector Automata)[4] [BRW98], qui est à notre connaissance la représentation décidable d'ensembles de vecteurs réels et entiers la plus expressive, parmi celles qui ont été implémentées. Les RVA sont utilisés dans l'outil LASH [BJW01, BJW05], comme on le verra dans la section 3.1. Ici, on montre que la classe des ensembles représentables par RVA est décomposable grâce à notre opérateur "$\uplus$", en reprenant la présentation utilisée dans [Ler03].

Soit un entier $b \geq 2$, appelé *base de décomposition*. On note $\Sigma_b = \{0, \ldots, b-1\}$ l'ensemble fini des *bits* et $S_b = \{0, b-1\}$ l'ensemble des *bits de signe*. Un mot infini $\sigma = \boldsymbol{s}\boldsymbol{a}_1 \ldots \boldsymbol{a}_k \star \boldsymbol{a}_{k+1}\boldsymbol{a}_{k+2} \ldots$ sur l'alphabet $\Sigma_b^n \cup \{\star\}$ est dit *b-correct* si $\boldsymbol{s} \in S_b^n$ et $\boldsymbol{a}_i \in \Sigma_b^n$ pour tout $i \geq 1$. On appelle $\rho_b : \Sigma_b^n \cup \{\star\} \longrightarrow \mathbb{R}^n$ le *décodage* d'un mot b*-correct*. Un mot b*-correct* σ est alors appelé la *décomposition "bit de poids fort en premier"* du vecteur réel $\rho_b(\sigma) \in \mathbb{R}^n$:

$$\rho_b(\sigma) = b^k \left(\frac{\boldsymbol{s}}{1-b} + \sum_{i \geq 1} b^{-i} \boldsymbol{a}_i \right)$$

Remarques : Notons que le symbole $\star$ correspond simplement au séparateur décimal habituel, tout comme une virgule (ou un point dans la norme anglo-saxonne). De plus, le terme correct pour ce qu'on appelle ici un bit est en fait "digit" (chiffre, en anglais), puisqu'on ne se restreint pas à une décomposition en base 2 (bit = *bi*nary digi*t*, en anglais).

Définition 2.11. *[BRW98] Un* RVA *en base b est un automate de Büchi A sur l'alphabet $\Sigma_b^n \cup \{\star\}$ tel que le langage $\mathsf{L}(A)$ reconnu par A ne contient que des mots b-corrects.*

L'ensemble $[\![A]\!]$ représenté par A est défini de la façon suivante : $[\![A]\!] = \{\rho_b(\sigma) \mid \sigma \in \mathsf{L}(A)\}$. Un ensemble $R \subseteq \mathbb{R}^n$ est dit *b-reconnaissable* s'il existe un RVA A en base b tel que $R = [\![A]\!]$.

[4] Les RVA ont été définis pour la première fois dans [BBR97], mais l'approche utilisée dans [BRW98] correspond davantage à la nôtre.

D'après [BRW98], la classe des ensembles b-reconnaissables est caractérisée logiquement par FO $(\mathbb{R}, \mathbb{Z}, +, \leq, X_b)$, où X_b est un prédicat supplémentaire. Ce prédicat X_b, défini sur $\mathbb{R}^3$, est tel que $X_b(x, u, a)$ est vrai si et seulement s'il existe une décomposition "bit de poids fort en premier" $\sigma = sa_1 \dots a_k \star a_{k+1} \dots$ pour x et un entier $i \in \mathbb{N}$ tel que $a_i = a$ et $u = b^i$. Intuitivement, $X_b(x, u, a)$ est vrai si et seulement si u est une puissance entière (positive ou négative) de b et s'il existe une décomposition de x dans laquelle le bit en position $\frac{u}{b}$ vaut a.

Théorème 2.12. *[BRW98] Un ensemble $R \subseteq \mathbb{R}^n$ est b-reconnaissable si et seulement s'il est définissable dans FO $(\mathbb{R}, \mathbb{Z}, +, \leq, X_b)$.*

Afin de pouvoir décomposer FO $(\mathbb{R}, \mathbb{Z}, +, \leq, X_b)$ avec notre opérateur, on montre que le prédicat X_b peut s'exprimer au moyen de deux autres prédicats V_b et W_b, définis comme des fonctions de valuation :

- $V_b : \mathbb{Z} \setminus \{0\} \longrightarrow \mathbb{Z}$ est la *fonction de valuation entière* introduite dans [BHMV94] et définie par $V_b(z) = b^j$, où $j \in \mathbb{N}$ est le plus grand entier tel que $b^{-j}z \in \mathbb{Z}$.

- $W_b : \mathbb{D} \setminus \{0\} \longrightarrow \mathbb{D}$ est la *fonction de valuation décimale* définie par $W_b(d) = b^{-j}$, où $j \in \mathbb{N}$ est le plus petit[5] entier tel que $b^j d \notin \mathbb{D}$.

Intuitivement, pour un $z \in \mathbb{Z}$, $V_b(z)$ se "place" sur le bit de poids faible (i.e. en position b^0) et "remonte" le long des bits vers la gauche (i.e. se place en position b^i en faisant croître i), en s'arrêtant sur le premier bit non-nul.
De manière tout à fait symétrique, pour un $d \in \mathbb{D}$, $W_b(d)$ se place sur le bit "décimal" de poids fort (i.e. en position b^{-1}) et descend le long des bits vers la droite (i.e. se place en position b^{-i} en faisant croître i), en s'arrêtant sur le premier bit non-nul.

En exprimant X_b dans FO $(\mathbb{R}, \mathbb{Z}, +, \leq, V_b, W_b)$ et V_b, W_b dans FO $(\mathbb{R}, \mathbb{Z}, +, \leq, X_b)$, on en déduit que FO $(\mathbb{R}, \mathbb{Z}, +, \leq, X_b) =$ FO $(\mathbb{R}, \mathbb{Z}, +, \leq, V_b, W_b)$. Voici les encodages de ces prédicats ; notons que l'on peut multiplier une variable par la valeur de bit $a \in \{0, 1, \dots, b-1\}$ puisque la base b est connue et fixée.

$$\begin{aligned} X_b(x, u, a) \equiv\ & \exists w \in \mathbb{D}, \exists y \in \mathbb{Z}, \exists z \in \mathbb{R}, x = w + y + z + au \wedge z < u \\ & \wedge \Big(\Big(u \geq 1 \wedge V_b(y) \geq bu \Big) \vee \Big(u < 1 \wedge W_b(w) \geq bu \Big) \Big) \end{aligned}$$

L'idée de cet encodage de $X_b(x, u, a)$ est de remarquer que la décomposition du réel x est la somme de deux réels y et z tels que la décomposition de y n'a que des 0 à droite

[5] Dans [BFL08], j est défini par erreur comme le plus petit entier avec $W_b(d) = b^j$ et non pas avec $W_b(d) = b^{-j}$ (dans ce cas il faudrait prendre j comme le plus *grand* entier négatif).

de la position indiquée par u, et symétriquement que z n'a que des 0 à gauche de cette position indiquée par u. Deux cas de figure se présentent alors : soit cette position est à gauche du séparateur $\star$, donc on utilise la valuation entière V_b, soit elle est à droite, et on utilise alors la valuation décimale W_b. On sépare alors y de sa partie décimale w, parce que V_b n'est défini que sur $\mathbb{Z}$ et W_b n'est défini que sur $\mathbb{D}$.

$$V_b(x) = y \;\equiv\; \exists a \in \mathbb{R}, X_b(x, y, a) \wedge a \neq 0 \wedge \forall z \in \mathbb{N}\Big(z < y \implies X_b(x, z, 0)\Big)$$

L'idée de cet encodage de $V_b(x) = y$ est simplement de considérer que $V_b(x)$ ne fait que donner la valeur y indiquant la position du bit non-nul le plus à droite dans la décomposition de l'entier x. On indique alors que le bit à cette position est non-nul, et que tous ceux qui sont à droite de cette position sont nuls.

$$W_b(x) = y \;\equiv\; \exists a \in \mathbb{R}, X_b(x, y, a) \wedge a \neq 0 \wedge \forall z \in \mathbb{D}\Big((bz < 1 \wedge z > y) \implies X_b(x, z, 0)\Big)$$

Similairement, l'idée de cet encodage de $W_b(x) = y$ est de considérer que $W_b(x)$ ne fait que donner la valeur y indiquant la position du bit non-nul le plus à gauche dans la décomposition du décimal x. On indique alors que le bit à cette position est non-nul, et que tous ceux qui sont à gauche de cette position et à droite de la virgule sont nuls.

À l'aide de ces trois encodages pour X_b, V_b, W_b, et grâce à la proposition 2.6 et au théorème 2.9, on obtient le théorème suivant :

Théorème 2.13. *FO* $(\mathbb{R}, \mathbb{Z}, +, \leq, X_b) =$ *FO* $(\mathbb{Z}, +, \leq, V_b) \uplus$ *FO* $(\mathbb{D}, +, \leq, W_b)$

De plus, il est clair que la logique FO $(\mathbb{Z}, +, \leq, V_b) \uplus$ FO $(\mathbb{D}, +, \leq, W_b)$ étend la logique FO $(\mathbb{Z}, +, \leq) \uplus$ FO $(\mathbb{D}, +, \leq)$. Cependant, même si le prédicat W_b est crucial dans la caractérisation logique des ensembles b-reconnaissables, ce prédicat n'est pas utilisé en pratique. En fait, pour pouvoir obtenir des algorithmes efficaces manipulant des automates de Büchi (notamment pour la minimisation et la déterminisation), on ne considère que les ensembles $R \subseteq \mathbb{R}^n$ pouvant être représentés par *RVA faibles* [BJW01]. On rappelle qu'un automate de Büchi A est dit *faible* si chacune de ses composantes fortement connexes S satisfait $S \subseteq F$ ou $S \cap F = \emptyset$, où F est l'ensemble des états acceptants.

Malheureusement, la classe des ensembles $R \subseteq \mathbb{R}^n$ représentables par RVA faibles n'est pas caractérisée par une logique, puisqu'elle n'est pas stable par les opérations du premier ordre, à cause de la projection (pour une explication détaillée, voir le Lemme 2 et le paragraphe qui le suit dans [BBL09]). Notons enfin que les RVA faibles sont utilisés dans l'outil LIRA [BDEK07], dont les bancs d'essais montrent des temps de calcul efficaces pour des ensembles définis dans la logique FO $(\mathbb{R}, \mathbb{Z}, +, \leq)$.

Remarque : Les trois prédicats X_b, V_b, et W_b ne donnent pas directement l'indice i d'un bit, mais la puissance b^i, bien que l'indice eût été généralement plus facile à manipuler. La raison principale est que si l'on disposait de l'indice, on pourrait exprimer les puissances, ce qui rendrait les logiques indécidables. Le paragraphe suivant prouve cette affirmation.

Supposons que l'on définisse X'_b par $X'_b(x, i, a) \equiv X_b(x, b^i, a)$ et V'_b par $V'_b(x) = i \equiv V_b(x) = b^i$. On a montré que V_b est exprimable dans FO $(\mathbb{R}, \mathbb{Z}, +, \leq, X_b)$ avec un encodage fonctionnant également pour exprimer V'_b dans FO $(\mathbb{R}, \mathbb{Z}, +, \leq, X'_b)$, de manière presque identique (il suffit de changer $z < y$ en $z > y$, puisque l'on numérote les indices dans le sens inverse des puissances de b). De plus, d'après [CP86], les prédicats V_b et b^x sont incomparables et rendent la logique FO $(\mathbb{N}, +, \leq, V_2, 2^x)$ indécidable (comme on l'a rappelé dans la section 2.1.2). On obtient alors l'égalité FO $(\mathbb{R}, \mathbb{Z}, +, \leq, X'_b) =$ FO $(\mathbb{R}, \mathbb{Z}, +, \leq, X'_b, V'_b)$. Or, de l'indécidabilité de la logique FO $(\mathbb{N}, +, \leq, V_2, 2^x)$, on peut déduire que FO $(\mathbb{N}, +, \leq, V'_2)$ est indécidable (en utilisant l'encodage du deuxième paragraphe de la page 5 de [CP86]), et donc la logique FO $(\mathbb{R}, \mathbb{Z}, +, \leq, X'_b, V'_b)$ est elle aussi indécidable. Suite à l'égalité précédente, FO $(\mathbb{R}, \mathbb{Z}, +, \leq, X'_b)$ est donc indécidable. Or la logique FO $(\mathbb{R}, \mathbb{Z}, +, \leq, X_b)$ a été montrée décidable dans [BRW98].

Par conséquent, les prédicats X_b et V_b (et similairement, W_b) ne peuvent pas être définis pour donner les indices i ; il doivent en effet donner directement la puissance b^i, à partir de laquelle on ne peut pas retrouver i (du moins, pas sans la multiplication, qui mènerait de toute façon à l'indécidabilité).

Notons également que le prédicat $P_b(x)$, vrai si et seulement si x est une puissance de b, est moins expressif que b^x. En effet, on peut le définir par l'encodage suivant : $P_b(x) \equiv \exists i \in \mathbb{Z}\ (x = b^i)$. Là encore, on manipule directement la puissance, mais on ne connaît pas l'exposant i, qui est quantifié existentiellement (voir la fin de la section 2.1.2 pour plus de détails sur ce prédicat).

2.4 Aux frontières de la décidabilité

Dans cette section, on étudie deux nouvelles extensions de la logique additive mixte, afin d'unifier ces logiques mixtes dans une seule logique les contenant toutes et restant décidable. En effet, la logique additive mixte a été étudiée depuis une dizaine d'années, notamment pour fournir des procédures de décision [Wei99, BRW98], pour caractériser des modèles de systèmes [CJ99, XDISP03, BFSP09], pour la décomposer en fonction d'autres logiques [BFL08], pour la caractériser géométriquement [FL08], ou encore, pour être implémentée [BJW05, BDEK07, BH06].

On aimerait cependant étudier ce qui se situe entre cette logique et l'indécidabilité.

On a vu que les CPDBM, par exemple, correspondent à une logique indécidable, mais que la logique des RVA, FO $(\mathbb{R}, \mathbb{Z}, +, \leq, X_b)$, reste décidable [BRW98]. Or lorsque l'on recherche des études de cas, on se rend compte que l'on aurait souvent besoin de la multiplication. Dans le cas du protocole BRP (Bounded Retransmission Protocol), on multiplie une variable (le délai de retransmission) par une borne paramétrée ; le paramètre est un objet mathématique qui se situe "entre" les notions de constante et de variable, et sa multiplication est possible dans la logique des CPDBM, permettant à l'outil TREX [ABS01] de vérifier ce protocole dans [AAB00].

La multiplication, couplée avec les entiers et l'addition, donne une logique indécidable (ce que l'on voudrait éviter). On envisage deux possibilités pour combiner autant que possible la multiplication et les entiers, tout en restant dans une logique décidable. Une première piste serait de considérer l'arithmétique réelle, et d'y ajouter un prédicat donnant une partie des entiers ; par exemple, si l'on se restreint aux puissances entières de 2 (ou d'une base b, plus généralement), on obtient la logique décidable FO $(\mathbb{R}, +, \times, \leq, P_b)$ [vdD85, AY07] (voir la section précédente ainsi que la section 2.1.1). Une autre piste serait de partir de la logique additive mixte, et d'y ajouter un fragment de la multiplication, restreinte aux décimaux : c'est cette extension-là que l'on étudie dans la section 2.4.1.

2.4.1 PDM : Presburger avec multiplication décimale

Cette section introduit une logique étendant la logique additive mixte avec la multiplication décimale. On a en effet une partie de la multiplication, ainsi que les entiers, et on reste dans une logique décidable. La multiplication décimale peut avoir diverses applications, comme par exemple dans les probabilités : on peut en effet multiplier deux variables comprises dans l'intervalle $[0, 1[$, ce qui suffit pour la plupart des calculs de probabilités. Cette logique peut alors avoir des application intéressantes, notamment pour représenter des propriétés de vérification et des ensembles d'accessibilité pour des modèles de systèmes probabilistes (processus de décisions markoviens, automates probabilistes, etc.).

Un exemple de tel système est le suivant : deux joueurs A et B s'affrontent dans un jeu où, en partant de l'état de contrôle initial, il faut gagner deux manches successives pour que la partie se termine. L'automate de la figure 2.2 représente un tel jeu, sur lequel chaque transition est étiquetée par $\xi_t(\alpha)$, où la fonction $\xi_t : \{A, B\} \longrightarrow \mathbb{D}$ donne la probabilité que le joueur $\alpha \in \{A, B\}$ remporte la manche au temps t donné.

Une exécution acceptante d'un tel automate représente une partie, dans laquelle l'un des deux joueurs a remporté deux manches successives (en partant de l'état q_1) ; notons que chaque manche i est remportée à un certain temps t_i. On peut donc calculer, par exemple, la probabilité que A gagne, en jouant à deux instants t_1, t_2, en calculant le

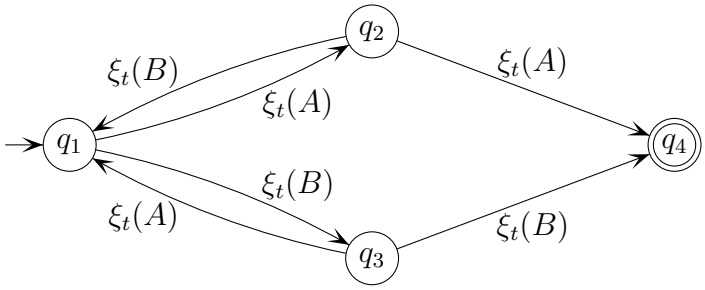


FIG. 2.2: Un automate modélisant un jeu probabiliste avec PDM

produit $\xi_{t_1}(A) \times \xi_{t_2}(A)$. Autre exemple, un peu plus riche : on peut également calculer la probabilité que B gagne la partie en 4 manches :

$$\Big(\Big(\xi_{t_1}(A) \times \xi_{t_2}(B)\Big) + \Big(\xi_{t_1}(B) \times \xi_{t_2}(A)\Big)\Big) \times \xi_{t_3}(B) \times \xi_{t_4}(B)$$

Si l'on se place dans un cadre stochastique et réaliste, on ne devrait pas avoir de probabilités qui atteigne 1 ; ces deux exemples de formules sont donc bien définissables dans FO $(\mathbb{D}, +, \times, \leq)$.

On a montré dans le théorème 2.9 que FO $(\mathbb{R}, \mathbb{Z}, +, \leq) =$ FO $(\mathbb{Z}, +, \leq) \uplus$FO $(\mathbb{D}, +, \leq)$. On définit ici une extension de cette logique avec la multiplication décimale, que l'on appelle *PDM* (pour *Presburger with Decimal Multiplication*) :

Définition 2.14. PDM *est la logique définie par FO* $(\mathbb{Z}, +, \leq) \uplus$ *FO* $(\mathbb{D}, +, \times, \leq)$.

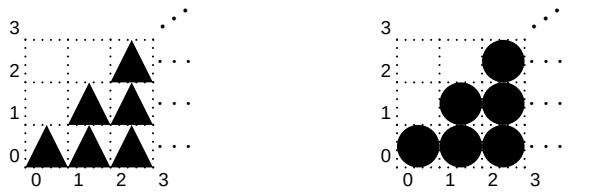
Soit la fonction $\tilde{\times} : \mathbb{D}^2 \longrightarrow \mathbb{D}$ telle que $(r_1 \,\tilde{\times}\, r_2 = r_3) \equiv (r_1 \times r_2 = r_3 \wedge r_1, r_2, r_3 \in \mathbb{D})$. En d'autres termes, $\tilde{\times}$ est la multiplication restreinte aux décimaux. On montre que PDM est équivalente à la logique FO $\big(\mathbb{R}, \mathbb{Z}, +, \tilde{\times}, \leq\big)$:

Proposition 2.15. *FO* $\big(\mathbb{R}, \mathbb{Z}, +, \tilde{\times}, \leq\big) =$ *FO* $(\mathbb{Z}, +, \leq) \uplus$ *FO* $(\mathbb{D}, +, \times, \leq) =$ *PDM*

Démonstration. On montre les deux inclusions.
"$\supseteq$" : Tout ensemble définissable dans la logique FO $(\mathbb{Z}, +, \leq) \uplus$ FO $\big(\mathbb{D}, +, \tilde{\times}, \leq\big)$ est également définissable dans FO $\big(\mathbb{R}, \mathbb{Z}, +, \tilde{\times}, \leq\big)$ de façon triviale. En particulier, les applications de $\times$ sont restreintes à des variables dans $\mathbb{D}$, et correspondent donc à des applications de $\tilde{\times}$.
"$\subseteq$" : Tout comme pour le théorème 2.9, on remarque que les ensembles $\mathbb{R}$ et $\mathbb{Z}$, la fonction $+ : \mathbb{R}^2 \longrightarrow \mathbb{R}$, et le prédicat $\leq$, sont définissables dans FO $(\mathbb{Z}, +, \leq) \uplus$ FO $(\mathbb{D}, +, \times, \leq)$ d'après les exemples 2.1, 2.2, 2.3, et 2.4. La fonction $\tilde{\times} : \mathbb{D}^2 \longrightarrow \mathbb{D}$ est, par définition, exprimable dans FO $(\mathbb{Z}, +, \leq) \uplus$ FO $(\mathbb{D}, +, \times, \leq)$. De la stabilité par

opérations du premier ordre, on obtient l'inclusion FO $(\mathbb{R}, \mathbb{Z}, +, \leq) \subseteq$ FO $(\mathbb{Z}, +, \leq) \uplus$ FO $(\mathbb{D}, +, \times, \leq)$.
On en déduit alors l'égalité. □

Les diagrammes sur la figure 2.3 illustrent une caractérisation géométrique de l'expressivité de FO $(\mathbb{R}, \mathbb{Z}, +, \leq)$ et de PDM.



FO $(\mathbb{R}, \mathbb{Z}, +, \leq)$ FO $\left(\mathbb{R}, \mathbb{Z}, +, \tilde{\times}, \leq\right)$

FIG. 2.3: Caractérisation géométrique de FO $(\mathbb{R}, \mathbb{Z}, +, \leq)$ et PDM

On montre maintenant que PDM est une logique décidable. En effet, c'est la combinaison de la logique de Presburger FO $(\mathbb{Z}, +, \leq)$, qui est décidable, et de l'arithmétique décimale FO $(\mathbb{D}, +, \times, \leq)$, elle aussi décidable, puisqu'elle est un fragment de l'arithmétique réelle (prouvée décidable par Tarski). Cette combinaison est effectuée par unions finies de sommes de la forme FO $(\mathbb{Z}, +, \leq) \uplus$ FO $(\mathbb{D}, +, \times, \leq)$; par la stabilité par opérations du premier ordre et en utilisant la proposition 2.7, on obtient le résultat suivant :

Proposition 2.16. *La logique PDM est décidable.*

On s'intéresse alors à l'expressivité de cette logique. La seule logique mixte décidable que nous avons vue jusqu'à présent, et qui contienne l'arithmétique linéaire, est la logique des RVA, notée FO $(\mathbb{R}, \mathbb{Z}, +, \leq, X_b)$. On montre que la logique des RVA est incomparable avec PDM :

Proposition 2.17. *PDM et FO* $(\mathbb{R}, \mathbb{Z}, +, \leq, X_b)$ *sont incomparables.*

Démonstration. Montrons que $E_1 = \{2^{-n} \mid n \in \mathbb{N} \setminus \{0\}\}$ (les puissances négatives de 2) n'est pas définissable dans PDM. Le RVA sur la figure 2.4 représente E_1 (i.e. son langage est $0 \star 0^* 1 0^\omega$) : Supposons que E_1 soit définissable dans PDM ; alors $E_1 = \bigcup_{1 \leq i \leq p} (D_i + Z_i)$, où $D_i \subseteq \mathcal{P}(\mathbb{D}^n)$ et $Z_i \subseteq \mathcal{P}(\mathbb{Z}^n)$ pour tout $1 \leq i \leq p$, avec $p \in \mathbb{N}$. Par définition, $E_1 \subset]0, 1[$, et donc $E_1 \subset \mathbb{D}$. On a alors $E_1 = \bigcup_{\{i \mid Z_i = \{0\}\}} D_i$, ce qui signifie que E_1 est définissable dans FO $(\mathbb{D}, +, \times, \leq)$.

En utilisant la technique d'élimination de quantificateurs telle que détaillée dans la preuve du théorème 3 dans [Rab77], on remarque que toute formule de l'arithmétique

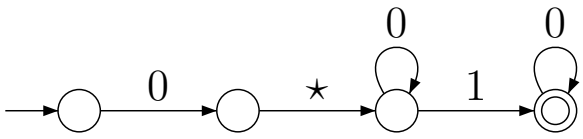


FIG. 2.4: Un RVA reconnaissant les puissances négatives de 2

réelle se décrit sous forme de disjonctions de conjonctions de polynômes. E_1 est donc de la forme $\cup(\cap[\![P(x)\#c]\!])$, où $\# \in \{\leq, <\}$, $P(x)$ est un polynôme en x, $c \in \mathbb{Z}$, et l'union et l'intersection sont finies. Cela signifie que E_1 est une union finie d'intervalles I_i ; donc il existe un indice i_0 tel que I_{i_0} contient une infinité de points de E_1. On a alors $I_{i_0} \cap E_1$ non-dénombrable, ce qui contredit la définition de E_1 (i.e. comme un ensemble dénombrable). On a donc un ensemble qui est définissable dans FO $(\mathbb{R}, \mathbb{Z}, +, \leq, X_2)$ (et donc dans FO $(\mathbb{R}, \mathbb{Z}, +, \leq, X_b)$ avec $b > 1$) mais pas dans FO $\left(\mathbb{R}, \mathbb{Z}, +, \tilde{\times}, \leq\right)$.

Montrons à présent que $E_2 = \left\{\frac{\sqrt{2}}{2}\right\} = \{x \in \mathbb{R}_+ \mid 2x^2 = 1\}$ n'est pas définissable dans FO $(\mathbb{R}, \mathbb{Z}, +, \leq, X_b)$. La formule suivante définit E_2 et appartient à FO $(\mathbb{D}, +, \times, \leq)$, et donc à PDM : $(x \times x) + (x \times x) = 1$. Supposons qu'il existe un RVA A représentant E_2 ; alors A aurait la forme suivante, puisque c'est un automate de Büchi (les pointillés représentent des suites de transitions) :

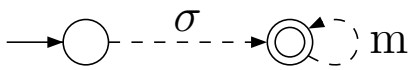


FIG. 2.5: Forme générale d'un RVA

La séquence de transitions σ (contenant le symbole $\star$) mènerait à un chemin infini passant par un état acceptant. Ce chemin infini est un circuit, étiqueté par un mot m, qui peut donc être lu infiniment souvent. Les mots acceptés par ce RVA sont donc de la forme $\mathsf{L}(A) = \sigma\mathrm{m}^\omega$, dont le décodage $[\![A]\!]$ est donné par ρ_b (que l'on note ρ puisqu'il n'y a pas d'ambigüité). On en déduit l'égalité suivante :

$$\begin{aligned}
\rho(\sigma\mathrm{m}^\omega) &= \rho(\sigma) + b^{-|\sigma|}.\rho(\mathrm{m}) + b^{-(|\sigma|+|\mathrm{m}|)}.\rho(\mathrm{m}) + b^{-(|\sigma|+|\mathrm{m}|+|\mathrm{m}|)}.\rho(\mathrm{m}) + \dots \\
&= \rho(\sigma) + \sum_{i=0}^{\infty} b^{-(|\sigma|+i.|\mathrm{m}|)}.\rho(\mathrm{m}) \qquad \text{(suite géométrique)} \\
&= \rho(\sigma) + b^{-|\sigma|}.\left(\frac{1}{1-\frac{1}{b^{|\mathrm{m}|}}}\right).\rho(\mathrm{m})
\end{aligned}$$

On aurait donc $[\![A]\!] \in \mathbb{Q}$; or, par définition de E_2, on veut $[\![A]\!] = \left\{\frac{\sqrt{2}}{2}\right\}$. On vient alors de montrer qu'il existe un ensemble définissable dans PDM qui n'est pas représentable par un RVA. □

On a donc deux logiques mixtes décidables et incomparables, dont l'expressivité semble très près de l'indécidabilité ; existe-t-il une logique décidable les contenant ? On va voir que c'est le cas, dans la section suivante.

2.4.2 RDM : la logique des RVA avec multiplication décimale

Dans cette section, on définit une logique mixte décidable plus expressive que toutes les autres logiques mixtes étudiées ici. On appelle cette logique RDM, pour "RVA with Decimal Multiplication" : c'est en fait la combinaison de PDM avec la logique des RVA. C'est une logique mixte qui permet donc de représenter des nombres ultimement périodiques, tout en disposant de la multiplication restreinte aux décimaux.

Quelques rappels :

- PDM est la logique FO $\left(\mathbb{R}, \mathbb{Z}, +, \tilde{\times}, \leq\right)$ = FO $(\mathbb{Z}, +, \leq) \uplus$ FO $(\mathbb{D}, +, \times, \leq)$, avec $\tilde{\times} : \mathbb{D}^2 \longrightarrow \mathbb{D}$ telle que $r_1 \tilde{\times} r_2 = r_3 \Leftrightarrow r_1 \times r_2 = r_3 \wedge r_1, r_2, r_3 \in \mathbb{D}$.

- la logique des RVA est FO $(\mathbb{R}, \mathbb{Z}, +, \leq, X_b) =$ FO $(\mathbb{Z}, +, \leq, V_b) \uplus$FO $(\mathbb{D}, +, \leq, W_b)$.

On définit alors la logique *RDM* comme la logique contenant à la fois ces deux logiques FO $\left(\mathbb{R}, \mathbb{Z}, +, \tilde{\times}, \leq\right)$ et FO $(\mathbb{R}, \mathbb{Z}, +, \leq, X_b)$:

Définition 2.18. RDM *est la logique FO* $\left(\mathbb{R}, \mathbb{Z}, +, \tilde{\times}, \leq, X_b\right)$.

On montre que RDM se décompose naturellement avec notre opérateur :

Proposition 2.19. *RDM* = *FO* $\left(\mathbb{R}, \mathbb{Z}, +, \tilde{\times}, \leq, X_b\right) =$ *FO* $(\mathbb{Z}, +, \leq, V_b) \uplus$*FO* $(\mathbb{D}, +, \times, \leq, W_b)$

Démonstration. La preuve découle directement de la combinaison des preuves de la proposition 2.15 et du théorème 2.13. □

La logique RDM est décidable, puisqu'elle est la combinaison (par unions finies de sommes) de deux logiques décidables, stables par opérations du premier ordre. La logique FO $(\mathbb{Z}, +, \leq, V_b)$ est en effet décidable [BHMV94]. La logique FO $(\mathbb{D}, +, \times, \leq, W_b)$ est elle aussi décidable, puisqu'elle est contenue dans la logique décidable FO $(\mathbb{R}, +, \times, \leq, P_b)$ [vdD85, AY07][6], vue dans la section 2.1.1. Cette inclusion se remarque aisément : $(x \in \mathbb{D}) \equiv x \in \mathbb{R}_+ \wedge x < 1$, les prédicats $+$, $\times$, et $\leq$ sont les mêmes (quoique définis sur un sous-ensemble), et $W_b(x) = y$ est équivalent à la formule $P_b(y) \wedge x < 1 \wedge x \geq y \wedge x < (b \times y)$, qui appartient à FO $(\mathbb{R}, +, \leq, P_b)$. D'après la proposition 2.7, on vient alors de prouver la proposition suivante :

[6] Ces deux articles considèrent le prédicat P_2 plutôt que P_b, mais la généralisation se fait naturellement, pour tout $b \geq 2$.

Proposition 2.20. *La logique RDM est décidable.*

RDM est une logique mixte, alors que FO $(\mathbb{R}, +, \times, \leq, P_b)$ est une logique réelle (avec quelques entiers). On pourrait croire que RDM contient toutes les logiques étudiées dans cette thèse, y compris les logiques uniquement réelles ; mais ce n'est pas le cas, puisque ces deux logiques sont incomparables, comme le montre la proposition suivante.

Proposition 2.21. *RDM et FO* $(\mathbb{R}, +, \times, \leq, P_b)$ *sont incomparables.*

Démonstration. Le principe de la preuve est le même que pour la proposition 2.17. Soit $E_3 = \mathbb{Z}$; E_3 est trivialement définissable dans RDM. En revanche, E_3 ne peut pas être défini dans FO $(\mathbb{R}, +, \times, \leq, P_b)$, car il rendrait cette logique indécidable.
Soit $E_4 = \{(x, y, z) \mid x \times y = z\}$; E_4 est définissable dans FO $(\mathbb{R}, +, \times, \leq, P_b)$. Cependant, pour la même raison que précédemment, il ne peut pas être défini dans RDM, sous peine de la rendre indécidable. □

Remarque : Cette démonstration ne fait aucunement appel aux prédicats X_b et P_b : elle utilise seulement le fait que les entiers et la multiplication ne peuvent être mélangés dans une même logique décidable. On peut donc en conclure, de façon plus générale, que chacune des logiques mixtes étudiées ici est incomparable avec les logiques réelles contenant la multiplication.

On pourrait alors vouloir construire une logique mixte maximale, qui englobe RDM et la logique réelle FO $(\mathbb{R}, +, \times, \leq, P_b)$: cependant, ajouter la multiplication à RDM la rendrait indécidable, et P_b est déjà contenu dans RDM car $P_b(x) \equiv X_b(x, x, 1)$ ou encore :

$$P_b(x) \equiv \Big(x \geq 1 \implies V_b(x) = x\Big) \wedge \Big(x < 1 \implies W_b(x) = x\Big)$$

Dans l'autre sens, on ne peut pas ajouter X_b à FO $(\mathbb{R}, +, \times, \leq, P_b)$, puisqu'elle deviendrait indécidable ; X_b permet en effet de donner tous les entiers, par l'encodage du prédicat $\mathcal{Z}$ (vu dans la section 2.1.3) :

$$\mathcal{Z}(x) \equiv \forall u \in \mathbb{R}\ \Big(u < 1 \implies X_b(x, u, 0)\Big)$$

On dispose alors de deux logiques qui semblent "maximales", puisqu'elles deviennent indécidables dès que l'on essaye de les combiner ; de plus, notons que la seule logique mixte parmi les deux est RDM.

Ce chapitre fournit à présent tous les éléments pour donner la vue hiérarchique de la figure 2.6 des quatre logiques mixtes étudiées ici. Notons que les logiques réelles n'apparaissent pas, puisqu'elles sont incomparables (comme indiqué dans la remarque ci-dessus).

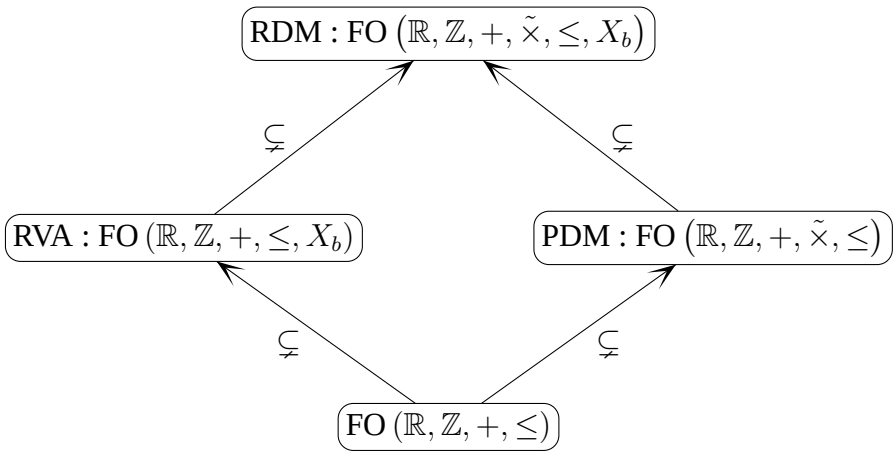


FIG. 2.6: Classification de logiques mixtes décidables en fonction de leur expressivité

Chapitre 3

Représentations symboliques et implémentation

Ce chapitre traite des différentes façons de représenter des ensembles de manière *effective*, c'est-à-dire d'une façon implémentable et utilisable dans un outil de vérification. Plus précisément, afin de manipuler les logiques étudiées dans cette thèse, on utilise ce que l'on appelle des *représentations symboliques*. Une représentation symbolique, ici, est un moyen de décrire des ensembles de vecteurs numériques, en fournissant une algorithmique pour construire et manipuler ces ensembles.

Plus précisément, on attend généralement d'une représentation symbolique que pour toute paire d'ensembles E_1, E_2 qu'elle peut décrire, les ensembles $E_1 \cup E_2$, $E_1 \cap E_2$, $E_1 \setminus E_2$, $\exists_i E_1$, et πE_1 soient calculables par un algorithme donnant le résultat dans cette même représentation symbolique. De plus, une telle représentation doit fournir un algorithme décidant si $E_1 = E_2$ (ou mieux, si $E_1 \subseteq E_2$), ainsi qu'une représentation pour l'ensemble vide et son dual (typiquement, $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$, $\mathbb{D}$, $\mathbb{R}_+$, etc.).

On cherche en général des représentations symboliques efficaces et complètes : on veut en effet disposer du plus grand nombre d'opérations possibles sur les ensembles représentés (union, intersection, différence, projection, etc.), que ces opérations soient les plus rapides possibles, en obtenant le meilleur compromis envisageable entre l'expressivité de la représentation et sa taille par rapport à la formule représentée.

Il existe diverses représentations symboliques implémentées et utilisées en vérification, que l'on peut regrouper en trois grandes familles. La première est celle des automates binaires, étudiée dans la section 3.1. La deuxième famille de représentations symboliques est celle des contraintes à deux variables représentées sous forme de matrices, dont on donne quelques exemples dans la section 3.2. Le troisième type de représentation symbolique, étudié dans la section 3.3, manipule directement des formules définissant des polyèdes convexes. Enfin, on propose dans la section 3.4 une

nouvelle représentation symbolique, publiée dans [BFL08], qui implémente l'opérateur de décomposition défini dans le chapitre précédent.

3.1 Automates binaires

Dans cette section, on présente les automates binaires, qui sont un des moyens les plus connus pour encoder des ensembles infinis de vecteurs numériques. Le principe de base est simple, et remonte au moins à [Büc60] : on encode chaque nombre de l'ensemble à représenter en un mot binaire, et on construit l'automate fini qui accepte uniquement ces mots binaires. Ceci permet d'encoder des entiers par des mots finis. Pour représenter des réels, on utilise des automates de Büchi afin d'obtenir une précision infinie pour les décimales (qui sont séparées de la partie entière par un symbole "marqueur" spécifique, généralement "$\star$").

Notons que l'on appelle couramment cette représentation "automates binaires", mais qu'elle s'étend naturellement à tout encodage positionnel de nombres dans une base de décomposition $b \geq 2$, comme vu dans la section 2.3.3. Cependant, en pratique, on choisit généralement $b = 2$.

On décrit ici les quatre types d'automates binaires utilisés en vérification, en indiquant dans quel outil ils sont implémentés. Pour plus de détails sur les trois premiers, voir la thèse de Jérôme Leroux [Ler03]. Concernant les automates de LIRA (le quatrième type décrit ici), voir [EK08].

3.1.1 NDD et LASH / MONA

Les NDD *(Number Decision Diagrams)* [WB95] sont une version améliorée des arbres de décision classiques (souvent appelés BDD, pour Binary Decision Diagrams), dans laquelle les feuilles sont étiquetées par des nombres en base $b \geq 2$, et chaque nœud peut avoir jusqu'à b fils ; chaque étage de l'arbre représente alors la valeur d'une variable. Les NDD correspondent à la partie entière des RVA, située avant séparateur décimal lorsque l'on parcourt l'automate ; il suffit donc d'un automate fini pour encoder des entiers, contrairement aux réels. Les NDD peuvent décrire exactement les ensembles définissables dans la logique FO $(\mathbb{Z}, +, \leq, V_b)$ [BHMV94], et sont implémentés dans l'outil LASH. Les NDD sont équivalents aux BDD multivariables de l'outil MONA, à quelques transitions près [Ler03].

L'outil LASH *(Liège Automata-based Symbolic Handler)* [LAS] vise à manipuler des ensembles infinis de réels (par RVA), d'entiers (par NDD), et de messages (pour les

canaux de communication, représentés par des QDD). Il fournit de plus un solveur pour la logique de Presburger.

L'outil MONA *(MONAdic second-order logic)* [Mon, HJJ+95], lui, est plutôt basé sur des arbres pour les logiques du deuxième ordre avec un ou deux successeurs. Il sert à construire des automates finis représentant des formules dans de telles logiques, afin d'effectuer diverses opérations.

3.1.2 UBA et FAST

Les UBA *(Unambiguous Binary Automata)* [Ler03] sont des NDD dans lesquels les transitions sont étiquetées par les symboles de l'alphabet $\{0, \ldots, b\}$, et non pas $\{0, \ldots, b\}^n$ comme les NDD (pour une base $b \geq 2$). Pour lire la $i^{ème}$ composante d'un vecteur, on lit une suite de n transitions dans un UBA, alors qu'une seule transition suffit, dans un NDD. Cette adaptation des NDD s'appelle la *sérialisation*, et permet d'éviter que la taille de l'alphabet soit multipliée par le nombre de variables. Ceci permet de gagner en concision dans de nombreux cas, et donc d'améliorer les performances pratiques de la représentation. En effet, bien que les UBA nécessitent plus de transitions que les NDD, ils sont déterministes et complets, et en les minimisant, on obtient une représentation généralement plus concise (cela provient également du fait que l'alphabet étant exponentiellement plus petit pour les UBA, les transitions se factorisent plus efficacement dans la minimisation). Les UBA sont implémentés dans l'outil FAST [BFLP03, BFLP08].

L'outil FAST *(Fast Acceleration of Symbolic Transition systems)* [FASa] permet d'analyser les systèmes à compteurs fonctionnels (voir la section 4.4). Il implémente une technique d'accélération automatisée, qui calcule des ensembles d'accessibilité. Il offre également la possibilité de vérifier des propriétés de sûreté.

Automates partagés. Dans [Cou04], Jean-Michel Couvreur a développé un moyen d'implémenter les automates finis de manière très efficace : les *automates partagés*. L'idée principale est de décomposer un automate en composantes fortement connexes, et de partager celles-ci entre tous les automates. Les automates partagés bénéficient d'une forme canonique, à l'instar des BDD. Le gain d'espace mémoire est important, et les temps d'exécution des opérations classiques s'en trouvent améliorés. L'exemple le plus convaincant est probablement celui du test d'égalité entre deux automates, qui est alors effectué en temps constant (comparaison de pointeurs sur les composantes fortement connexes).

Ces automates partagés sont implémentés dans SATAF *(Shared Automata)* [TAF], et sont utilisés par l'outil PRESTAF, qui implémente le principe des automates binaires

pour les ensembles définissables dans *Pres*burger. L'outil PRESTAF est utilisé par la version actuelle de FAST [FASb], qui a été intégrée à la plateforme modulaire GENEPI (voir la section 3.4).

3.1.3 RVA et LASH

Les RVA (Real Vector Automata) sont décrits dans la section 2.3.3. Ce sont des NDD dans lesquels chaque mot est suivi du séparateur décimal puis d'un "NDD infini", c'est-à-dire un automate de Büchi binaire qui reconnaît les décimales de chaque mot. Plus précisément, on utilise en pratique des automates de Büchi faibles et déterministes, pour leur efficacité (la complémentation d'un automate faible déterministe se fait simplement en inversant les conditions d'acceptation sur les composantes fortement connexes). Cette représentation est implémentée dans l'outil LASH, mais elle est difficilement utilisable en pratique du fait de la taille des automates qu'elle produit.

L'outil LASH est décrit dans la section 3.1.1. Une des extensions prévues par l'auteur principal (Bernard Boigelot) est de permettre l'exploration de l'espace des configurations d'automates temporisés ou hybrides (voir chapitre 5), en représentant ces configurations par des RVA. Ce projet semble cependant changer de direction, du fait des récents travaux sur l'accélération hybride et de leur implémentation dans le prototype HAT (voir page 104), basé sur les RVA de LASH et sur les polyèdres de POLYLIB (voir section 3.3).

3.1.4 "Don't Care RVA" et LIRA

Afin de réduire la taille des RVA, l'idée des "don't care words" (issue des BDD) a été généralisée et retravaillée dans [EK06] ; cette idée consiste à ne plus se soucier du fait que certains mots soient acceptés ou non par l'automate. En effet, pour la plupart des nombres définissables dans FO $(\mathbb{R}, \mathbb{Z}, +, \leq, X_b)$, il existe deux RVA les représentant. Par exemple, l'entier 2 peut s'écrire comme les mots binaires réels infinis $10 \star 0^\omega$ et $1 \star 1^\omega$. L'idée principale de ces "Don't Care RVA" est de négliger tous les mots de la forme $\{0,1\}^+ \star \{0,1\}^* 1^\omega$, qui introduisent un double encodage pour certains nombres à représenter.

Cette technique permet alors d'obtenir un RVA minimal et unique pour chaque ensemble $R \subset \mathbb{R}^n$ représentable, et ainsi de tester le vide en temps linéaire. Cependant, la projection (ou quantification) d'une variable peut ne plus donner un RVA, et prend plus de temps dans le cas des "Don't Care RVA". Toutefois, la projection est une opération que l'on n'est pas nécessairement amené à utiliser en pratique. De manière générale, cette amélioration des RVA permet d'obtenir des résultats intéressants tant en théorie qu'en pratique, au niveau de la taille des automates et des temps d'exécution pour les

opérations [BDEK07]. L'outil implémentant ces "Don't Care RVA" s'appelle LIRA.

L'outil LIRA *(Linear Integer/Real Arithmetic solver)* [LIR] fournit un solveur pour la logique additive mixte[1] FO $(\mathbb{R}, \mathbb{Z}, +, \leq)$. Les benchmarks montrent une efficacité nettement supérieure à celle de LASH, OMEGA, et MONA, et sont disponibles à l'adresse suivante : `http://lira.gforge.avacs.org/toolpaper/#benchmarks`. Cependant, ces benchmarks montrent aussi que pour la logique de Presburger, l'outil PRESTAF reste le plus efficace (grâce à la mise en mémoire cache des composantes fortement connexes d'automates partagés, ce qui évite de les calculer plusieurs fois).

Conclusion relative aux automates binaires

L'avantage principal des automates binaires est qu'ils offrent un pouvoir d'expression important ainsi que la stabilité par les opérations du premier ordre et booléennes. Cependant, les automates binaires sont généralement assez lourds à manipuler en pratique, notamment lors de leur construction à partir de contraintes linéaires [WB00]. Le cas particulier des automates de LIRA amoindrit considérablement ce problème, au détriment de la stabilité par projection (ce qui est un sacrifice que l'on peut toutefois se permettre dans de nombreux cas). Lorsque l'on se restreint aux entiers, cependant, les automates partagés font de PRESTAF un outil très efficace.

3.2 Matrices de contraintes

Cette section présente les représentations symboliques basées sur les matrices à différences bornées, qui permettent de représenter des ensembles délimités par des contraintes portant sur deux variables à la fois. Le principe de base, tel que décrit dans la section 2.2.1, consiste à borner la différence entre deux variables. Cette borne peut être une constante entière (dans le cas des DBM), ou un terme arithmétique défini dans une logique donnée (comme pour les CPDBM), selon le principe décrit dans la section 2.3.1.

On présente d'abord le cas le plus simple (DBM), puis une extension (CPDBM). Pour finir, on donne quelques exemples d'autres extensions des DBM.

3.2.1 DBM et UPPAAL

La version la plus simple des matrices de contraintes est appelée DBM *(Difference Bound Matrices)* [BM83, Dil89], et est formellement définie dans la section 2.2.1. Le

[1] Du fait de la perte de stabilité par projection, les ensembles représentables par "Don't Care RVA" ne sont pas caractérisables par une logique ; les ensembles définissables dans FO $(\mathbb{R}, \mathbb{Z}, +, \leq)$ donnent cependant une idée assez réaliste de ce que l'on peut exprimer dans LIRA.

principe est le suivant : on cherche à représenter des ensembles de réels définis par des conjonctions de contraintes de la forme $x_1 - x_2 \prec c$, où $\prec \in \{\leq, <\}$ et c est une constante entière. Une DBM est donc une matrice bornant la différence entre chaque couple de variables par une constante ; l'ensemble représenté par une DBM est donc délimité par la conjonction des contraintes de la DBM. Plus précisément, une DBM définit un polyèdre convexe ; on peut donc manipuler des unions de polyèdres convexes, en utilisant plusieurs DBM.

Les DBM sont très efficaces, en pratique : les opérations d'union et d'intersection se font très rapidement, et elles nécessitent assez peu de mémoire. Cependant, les DBM ne sont pas stables par complémentation. De plus, les DBM souffrent d'un manque d'expressivité ; si des unions de polyèdes convexes suffisent pour les automates temporisés, ce n'est pas le cas pour des systèmes hybrides plus complexes ou des systèmes à compteurs. L'implémentation la plus célèbre des DBM est certainement l'outil UPPAAL.

L'outil UPPAAL [Upp] est développé et maintenu conjointement dans les universités d'*Upp*sala (Suède) et *Aal*borg (Danemark), depuis une douzaine d'années. Cet outil est un model-checker pour les automates temporisés (voir la section 5.2), constamment amélioré et étendu avec de nouvelles fonctionnalités. Il a la particularité d'être très utilisé (nombreuses études de cas à l'appui), et ce même dans le milieu industriel, ce qui témoigne des performances de cet outil. Plusieurs outils basés sur UPPAAL ont été mis en œuvre, notamment pour la modélisation et la vérification de systèmes temps-réel embarqués (TIMES) et d'automates probabilistes (PRO), pour l'ordonnancement (CORA), le test temps-réel (TRON), les jeux temporisés (TIGA), etc. Toutes ces extensions, ainsi qu'une panoplie de tutoriels, publications, et présentations sont disponibles sur le site web [Upp].

3.2.2 CPDBM et TREX

Une extension des DBM connue est celle des CPDBM *(Constrained Parametric DBM)* [AAB00], qui sont présentées formellement dans les sections 2.2.1 et 2.3.1. L'idée est d'augmenter l'expressivité des DBM, en définissant les bornes de différences non plus par des constantes entières, mais par des termes arithmétiques. Ces termes arithmétiques sont définis dans l'arithmétique mixte, mais sans quantificateurs. Le problème principal des CPDBM est que cette logique, dans laquelle sont définies les bornes, est indécidable (du fait de la présence combinée des entiers avec l'addition et la multiplication). Les CPDBM ont été définies pour être implémentées dans l'outil TREX.

L'outil TREX *(Tool for Reachability analysis of compleX systems)* [TRe, ABS01] est un outil pour analyser des systèmes de nature différente : systèmes à compteurs, automates temporisés paramétrés, automates avec canaux de communication non-fiables,

ou une combinaison de ces trois modèles. TREX tente de générer l'ensemble des configurations accessibles dans un système, puis de vérifier des propriétés de sûreté (voire de vivacité). Toutefois, étant donnée l'indécidabilité inhérente des CPDBM et des modèles en question, l'analyse d'accessibilité ne termine pas nécessairement.

3.2.3 Autres extensions des DBM

Afin de clarifier l'expressivité des CPDBM, on les a paramétrées par la logique $\mathscr{L}$ dans laquelle les bornes des matrices sont définies (voir la section 2.3.1). Ainsi, on a défini les $CP\text{-}DBM_{\mathscr{L}}$, qui sont des DBM dans lesquelles les termes arithmétiques t (bornant les différences de la forme $x_1 - x_2 \leq t$) sont définis dans la logique $\mathscr{L}$. On a alors caractérisé les CPDBM de [ABS01] comme étant des $CP\text{-}DBM_{\mathscr{L}}$ où $\mathscr{L}$ est l'arithmétique mixte sans quantificateurs. De plus, on a montré que pour le cas où $\mathscr{L}$ est la logique de Presburger, on peut caractériser les ensembles représentés par des $CP\text{-}DBM_{\mathscr{L}}$ en utilisant notre opérateur de décomposition du chapitre précédent (voir la proposition 2.8 page 48).

Il existe de nombreuses autres extensions des DBM. Par exemple, Péron et Halbwachs [PH07] ont ajouté aux contraintes classiques des DBM ($x_1 - x_2 \leq c$) des contraintes de la forme $x_1 - x_2 \neq 0$, afin d'obtenir des dDBM *(disequalities DBM)* ; leur objectif était de différencier des alias de variables (i.e. des doublons) dans des programmes à vérifier.

Afin d'optimiser l'implémentation des DBM, les CDD *(Clock Difference Diagrams)* [LWYP99] et les DDD *(Difference Decision Diagrams)* [MLAH99] ont été définis : les DBM ont en effet un problème de redondance d'information, qui peut facilement être traité en se basant sur les BDD.

Enfin, dans le but de généraliser les contraintes des DBM, Antoine Miné a défini les *octogones* [Min01], dans lesquels les contraintes sont de la forme $\pm x_1 \pm x_2 \leq c$. Il a poursuivi ce travail en définissant les *domaines numériques abstraits* [Min02], qui manipulent des contraintes de la forme $x_1 - x_2 \in \mathcal{C}$ où $\mathcal{C}$ est un domaine abstrait (voir [Min02] pour plus de détails à ce sujet).

Conclusion relative aux matrices de contraintes

De manière générale, les matrices définissant des contraintes sur des couples de variables partagent les mêmes avantages et inconvénients. Leur avantage principal est l'efficacité pratique, qui permet une manipulation rapide des ensembles représentés, en occupant relativement peu d'espace mémoire. Cependant, l'expressivité de ces représentations est souvent trop limitée pour l'utilisation que l'on aimerait en faire (i.e. la vérification de systèmes infinis numériques temporisés). De plus, elles ne sont généralement

pas stables par les opérations classiques (complémentation, projection, etc.). Cela dit, ces représentations symboliques sont bien adaptées à l'utilisation pour laquelles elles sont prévues, c'est-à-dire pour représenter les configurations d'automates temporisés dans UPPAAL (DBM) et pour analyser les ensembles d'accessibilité dans les systèmes de TREX (CPDBM).

3.3 Formules polyédriques

Cette section décrit brièvement la représentation d'ensembles de vecteurs numériques qui semble la plus naturelle : les formules arithmétiques. Ces ensembles sont alors décrits par des conjonctions de contraintes linéaires sur des vecteurs de variables numériques, formant des polyèdres.

Il existe plusieurs implémentations suivant cette approche. Une des plus anciennes, restreinte aux entiers, s'apelle OMEGA *(The Omega Project : Frameworks and Algorithms for the Analysis and Transformation of Scientific Programs)* [OME, Pug91] ; elle n'est plus maintenue depuis longtemps, et son efficacité a été largement dépassée depuis.

Une autre implémentation, POLYLIB (*Poly*hedra *Lib*rary) [POLb], manipule des unions de polyèdres fermés, c'est-à-dire que les inégalités dans les contraintes ne doivent pas être strictes. Ceci est une restriction trop forte pour nous ; en effet, on ne pourrait même pas représenter les décimaux, puisqu'ils sont ouverts sur 1.

Cependant, une autre bibliothèque de fonctions a été développée en surcouche à POLYLIB : c'est la bibliothèque NEWPOLKA [POLa], qui permet aux polyèdres d'être ouverts. Cependant, les polyèdres doivent être convexes ; on peut toutefois manipuler des unions de polyèdres convexes, ce qui augmente considérablement l'expressivité de cette représentation.

Enfin, l'implémentation qui semble la plus efficace pour les formules polyédriques est PPL *(Parma Polyhedra Library)* [PPL]. Elle offre les mêmes fonctionnalités que NEWPOLKA, à savoir la manipulation de polyèdres convexes (ouverts ou fermés) ; l'utilisateur écrit et lit des formules (par exemple, $x + 2 * y + 5 * z <= 7$), et les opérations sont faites directement sur ces formules, en passant si besoin par des générateurs (i.e. bases et périodes). Le site web de PPL propose des benchmarks montrant qu'elle est plus rapide que NEWPOLKA ; les résultats de ces tests sont disponibles à l'adresse `http://www.cs.unipr.it/ppl/performance`.

La complexité dans le pire des cas est exponentielle en fonction de la taille de la formule et du nombre de variables, pour la plupart des opérations (union, intersection, projection, etc.). Cette limite bien connue pas les théoriciens contraint également

les implémentations, aussi optimisées soient-elles. En particulier, puisque ces formules définissent des unions de polyèdres convexes, elles supportent très mal la complémentation (par laquelle elles ne sont pas stables).

Conclusion relative aux formules polyédriques

D'après notre expérience pratique, le principal avantage de cette représentation symbolique est qu'il semble naturel de décrire des ensembles par des formules, qui sont la représentation la plus concise (seulement quelques caractères alphanumériques) : de ce fait, l'occupation de l'espace mémoire est réduite (par exemple par rapport à une structure de données manipulant un automate binaire). De plus, la modélisation se fait plus facilement, pour un humain. Cependant, certaines opérations peuvent être fastidieuses à calculer (notamment la complémentation). Leur expressivité est la même que celle des DBM, qui peuvent parfois être plus rapides en pratique.

3.4 IDS et GENEPI

Dans cette section, on présente une implémentation de notre opérateur $\uplus$ défini dans le chapitre précédent. Cette manière de composer entiers et réels est conçue de façon à correspondre au cadre de GENEPI. L'outil GENEPI [GEN] est une plateforme modulaire destinée à supporter des solveurs basés sur la logique de Presburger ainsi que des model-checkers. Le noyau central de GENEPI est un gestionnaire de *plugins* (i.e. de modules additionnels), qui effectue les opérations classiques (opérations booléennes, quantification, satisfiabilité, etc.) sur des ensembles de vecteurs. Ces ensembles sont encodés comme les solutions de formules arithmétiques, et le gestionnaire central les manipule à un niveau d'abstraction générique ; le véritable calcul des opérations est ensuite effectué par un plugin approprié.

Différentes implémentations existent donc pour chaque opération, dépendant de la représentation symbolique utilisée par le plugin en question. Un plugin est en fait une surcouche à un outil existant, le rendant compatible avec le niveau d'abstraction générique manipulé par le gestionnaire de plugins. Actuellement, GENEPI possède des plugins pour les outils PRESTAF, LIRA, LASH, MONA, OMEGA, et PPL. Nous avons programmé un prototype pour notre décomposition, appelé IDS (pour "Integer-Decimal Sums"). Comme expliqué dans la section 2.2.2, les ensembles que l'on peut représenter sont des unions finies de sommes "entiers+décimaux". On présente ici les IDS, qui utilisent deux autres représentations symboliques : un plugin pour les entiers, et un autre pour les décimaux.

En utilisant les IDS, on peut alors combiner n'importe quelle paire de plugins :

celle qui nous a semblé la plus efficace[2] jusqu'alors est d'utiliser PRESTAF sur les entiers et PPL sur les décimaux. Pour des ensembles dont la construction fait appel à des complémentations, il s'avère plus efficace d'utiliser le plugin LIRA sur les décimaux, du fait de la complexité de la complémentation dans PPL. On peut également essayer d'autres combinaisons : par exemple, on peut essayer de combiner deux instances du plugin LASH, l'une sur les entiers et l'autre sur les décimaux. Cette combinaison permettrait de voir si, en pratique, notre décomposition améliore les performances des RVA par rapport à leur outil dédié.

Afin d'implémenter notre décomposition, on a besoin d'un moyen unique de représenter des ensembles, principalement pour éviter les doublons ; en effet, un même point (ou ensemble) pourrait être contenu plusieurs fois dans une formule, ce qui génèrerait la construction d'une représentation inutilement grande. Dans le reste de cette section, on présente le cadre théorique, nécessaire à l'unicité de la représentation, qui est implémenté dans les IDS.

Soient $\mathfrak{Z} \subseteq \mathcal{P}(\mathbb{Z}^n)$ et $\mathfrak{D} \subseteq \mathcal{P}(\mathbb{D}^n)$. Remarquons que si $R = (Z + D_1) \cup (Z + D_2)$, alors $R = Z + D$ avec $D = D_1 \cup D_2$; sans perdre en généralité, on supposera que $\mathfrak{D}$ est toujours clos par union. On remarque alors que $R \subseteq \mathbb{R}^n$ peut être représenté par une fonction partielle f_R :

$$\begin{aligned} f_R : \mathfrak{Z} &\longrightarrow \mathfrak{D} \\ Z_i &\longmapsto D_i \end{aligned}$$

On définit l'interprétation de cette fonction comme $[\![f_R]\!] = \bigcup_{i=1}^{p} \Big(Z_i + f_R(Z_i) \Big)$, ce qui correspond à l'écriture naturelle de R introduite dans la section 2.2.2. Notons que cette représentation f_R n'est pas unique.

Pour des raisons techniques[3], on étend f_R à la fonction totale $\overline{f_R}$ telle que $\overline{f_R}(Z) = \emptyset$ si $Z \notin \mathsf{dom}(f_R)$ et $\overline{f_R}(Z) = f_R(Z)$ sinon. De plus, on définit le support de $\overline{f_R}$ comme $\mathsf{supp}(\overline{f_R}) = \{Z \mid \overline{f_R}(Z) \neq \emptyset\}$. Par la suite, on utilisera la notation f_R au lieu de $\overline{f_R}$, sans ambiguïté.

On est donc capable de représenter l'ensemble R avec une fonction, que l'on souhaite alors manipuler : c'est pourquoi on veut identifier f_R à $[\![f_R]\!]$. Pour ce faire, cette

[2] Nous avons effectué des tests comparatifs sur la taille des structures de données et les temps d'exécution pour la construction d'ensembles définis par des conjonctions de contraintes linéaires (de 1 à 7 contraintes sur 1 à 5 variables).

[3] En pratique, on veut éviter de manipuler des ensembles qui ne sont pas définis ; ils correspondent naturellement à l'ensemble vide, ce par quoi on les représente donc.

interprétation doit être une fonction injective ; mais ce n'est pas le cas, en général. En effet, en utilisant les définitions précédentes, on pourrait avoir plusieurs écritures pour $[\![f_R]\!]$. Cependant, si les images par f_R sont disjointes, alors l'interprétation $[\![f_R]\!]$ est une fonction injective. Enfin, pour des raisons d'effectivité, on ne considère que des fonctions dont le support est fini.

Le reste de cette section est entièrement consacré à la formalisation du raisonnement développé dans le paragraphe précédent.

Soit $\mathcal{F}_{\mathfrak{Z}\to\mathfrak{D}} = \{f : \mathfrak{Z} \longrightarrow \mathfrak{D} \mid \mathsf{supp}(f) \text{ est fini}\}$.

Définition 3.1. *La* fonction d'interprétation $[\![.]\!]$ *associe à tout* $f \in \mathcal{F}_{\mathfrak{Z}\to\mathfrak{D}}$ *un ensemble de vecteurs réels défini par* $[\![f]\!] = \bigcup_{Z\in\mathsf{supp}(f)} \Big(Z + f(Z)\Big)$.

Remarquons que puisque $\mathsf{supp}(f)$ est fini, $\mathcal{F}_{\mathfrak{Z}\to\mathfrak{D}}$ ne suffit pas à représenter tous les ensembles de vecteurs réels, comme le montre le contre-exemple page 46. On se restreint alors aux fonctions suivantes :

Définition 3.2. *Un* IDF (Integer-Decimal Function) *est une fonction* $f \in \mathcal{F}_{\mathfrak{Z}\to\mathfrak{D}}$ *telle que* $\bigcup_Z f(Z) = \mathbb{D}^n$ *et* $Z \neq Z' \implies f(Z) \cap f(Z') = \emptyset$. *On note* $IDF_{\mathfrak{Z}\to\mathfrak{D}} = \{f \in \mathcal{F}_{\mathfrak{Z}\to\mathfrak{D}} \mid f \text{ est un IDF}\}$ *pour décrire tous les IDF. On utilisera également la notation* $[\![IDF_{\mathfrak{Z}\to\mathfrak{D}}]\!] = \{[\![f]\!] \mid f \in IDF_{\mathfrak{Z}\to\mathfrak{D}}\}$.

Les ensembles des exemples 2.1, 2.2, 2.3, et 2.4 sont représentés par les IDF suivants :

Exemple 3.3. *L'ensemble vide* $\emptyset$ *est représenté par l'IDF* $f_\perp$ *défini par* $f_\perp(Z) = \emptyset$ *pour tout* $Z \neq \emptyset$ *et par* $f_\perp(\emptyset) = \mathbb{D}^n$. *L'ensemble* $\mathbb{R}^n$ *est représenté par l'IDF* $f_\top$ *(parfois noté* $f_{\mathbb{R}^n}$*) défini par* $f_\top(\mathbb{Z}^n) = \mathbb{D}^n$ *et* $f_\top(Z) = \emptyset$ *sinon. L'ensemble* $\mathbb{Z}^n$ *est représenté par l'IDF* $f_{\mathbb{Z}^n}$ *défini par* $f_{\mathbb{Z}^n}(\mathbb{Z}^n) = \{\mathbf{0}\}$ *et* $f_{\mathbb{Z}^n}(Z) = \emptyset$ *sinon.*

Exemple 3.4. *L'ensemble* $R_= = \{\boldsymbol{r} \in \mathbb{R}^2 \mid r_1 = r_2\}$ *est représenté par l'IDF* $f_=$ *défini par* $f_=(Z_=) = D_=$, $f_=(\emptyset) = \mathbb{D}^2 \setminus D_=$ *et* $f_=(Z) = \emptyset$ *sinon, où :*

$$Z_= = \{\boldsymbol{z} \in \mathbb{Z}^2 \mid z_1 = z_2\} \qquad D_= = \{\boldsymbol{d} \in \mathbb{D}^2 \mid d_1 = d_2\}$$

Exemple 3.5. *L'ensemble* $R_\leq = \{\boldsymbol{r} \in \mathbb{R}^2 \mid r_1 \leq r_2\}$ *est représenté par l'IDF* $f_\leq$ *défini par* $f_\leq(Z_<) = D_>$, $f_\leq(Z_\leq) = D_\leq$ *et* $f_\leq(Z) = \emptyset$ *sinon, où :*

$$Z_< = \{\boldsymbol{z} \in \mathbb{Z}^2 \mid z_1 < z_2\} \qquad D_> = \{\boldsymbol{d} \in \mathbb{D}^2 \mid d_1 > d_2\}$$
$$Z_\leq = \{\boldsymbol{z} \in \mathbb{Z}^2 \mid z_1 \leq z_2\} \qquad D_\leq = \{\boldsymbol{d} \in \mathbb{D}^2 \mid d_1 \leq d_2\}$$

Exemple 3.6. *L'ensemble* $R_+ = \{\boldsymbol{r} \in \mathbb{R}^3 \mid r_1 + r_2 = r_3\}$ *est représenté par l'IDF* f_+ *défini par* $f_+(Z_0) = D_0$*,* $f_+(Z_1) = D_1$*,* $f_+(\emptyset) = \mathbb{D}^3 \setminus (D_1 \cup D_2)$ *et* $f_+(Z) = \emptyset$ *sinon, où (intuitivement,* $c \in \{0, 1\}$ *est la retenue) :*

$$Z_c = \{\boldsymbol{z} \in \mathbb{Z}^3 \mid z_1 + z_2 + c = z_3\}$$
$$D_c = \{\boldsymbol{d} \in \mathbb{D}^3 \mid d_1 + d_2 = d_3 + c\}$$

On remarque que tout ensemble dans $[\![IDF_{\mathfrak{Z}_n \to \mathfrak{D}_n}]\!]$ est dans $\mathfrak{Z}_n \uplus \mathfrak{D}_n$. L'inclusion inverse est obtenue en prouvant la proposition suivante :

Proposition 3.7 (Clôture par l'union)**.** *Soit* $R \in [\![IDF_{\mathfrak{Z}_n \longrightarrow \mathfrak{D}_n}]\!]$*. Alors, pour tout* $Z \in \mathfrak{Z}_n$ *et* $D \in \mathfrak{D}_n$*, on a également* $R \cup (Z + D) \in [\![IDF_{\mathfrak{Z}_n \to \mathfrak{D}_n}]\!]$.

Démonstration. Considérons un IDF $f : \mathfrak{Z}_n \longrightarrow \mathfrak{D}_n$ tel que $[\![f]\!] = R$, et deux ensembles $Z \in \mathfrak{Z}_n$ et $D \in \mathfrak{D}_n$. On doit alors prouver qu'il existe un IDF $f' : \mathfrak{Z}_n \longrightarrow \mathfrak{D}_n$ tel que $[\![f']\!] = R'$ avec $R' = R \cup (Z + D)$. On considère la fonction suivante :

$$\begin{array}{rccl} f' : & \mathfrak{Z}_n & \longrightarrow & \mathfrak{D}_n \\ & Z' & \longrightarrow & \Big(f(Z') \setminus D\Big) \displaystyle\bigcup_{Z'' \mid Z'' \cup Z = Z'} \Big(f(Z'') \cap D\Big) \end{array}$$

On va donc prouver que f' est un IDF tel que $[\![f']\!] = R'$. Commençons par montrer que f' est un IDF.

Tout d'abord, remarquons que $\bigcup_{Z'} f'(Z') = \mathbb{D}^n$. Ensuite, soient $Z'_1, Z'_2 \in \mathfrak{Z}_n$ tels que $f'(Z'_1) \cap f'(Z'_2) \neq \emptyset$; soit on a $(f(Z'_1) \setminus D) \cap (f(Z'_2) \setminus D) \neq \emptyset$, soit il existe Z''_1, Z''_2 tels que $Z''_1 \cup Z = Z'_1$ et $Z''_2 \cup Z = Z'_2$ et $(f(Z''_1) \cap D) \cap (f(Z''_2) \cap D) \neq \emptyset$, puisque les autres cas sont impossibles.

Cependant, $(f(Z'_1) \setminus D) \cap (f(Z'_2) \setminus D) \neq \emptyset$ implique que $f(Z'_1) \cap f(Z'_2) \neq \emptyset$, et puisque f est un IDF, on obtient $Z'_1 = Z'_2$. De plus, $(f(Z''_1) \cap D) \cap (f(Z''_2) \cap D) \neq \emptyset$ implique que $Z''_1 = Z''_2$, et en particulier que $Z'_1 = Z'_2$. On a alors prouvé que f' est un IDF.

Enfin, l'égalité $[\![f']\!] = R'$ provient de l'écriture suivante :

$$\begin{aligned}
[\![f']\!] =& \bigcup_{Z'}(Z' + f'(Z')) \\
=& \bigcup_{Z'}\Big((Z' + (f(Z') \setminus D)) \\
& \bigcup_{Z''|Z''\cup Z=Z'} (Z' + (f(Z'') \cap D))\Big) \\
=& \bigcup_{Z'}(Z' + (f(Z') \setminus D)) \bigcup_{Z''}((Z'' \cup Z) + (f(Z'') \cap D)) \\
=& \bigcup_{Z''}(Z'' + ((f(Z'') \setminus D) \cup (f(Z'') \cap D))) \\
& \cup (Z + D \cap (\bigcup_{Z''} f(Z''))) \\
=& [\![f]\!] \cup (Z + D)
\end{aligned}$$

□

On vient alors de montrer la proposition suivante :

Proposition 3.8. $\mathfrak{Z}_n \uplus \mathfrak{D}_n = [\![IDF_{\mathfrak{Z}_n \to \mathfrak{D}_n}]\!]$

Prouvons que cette représentation est unique pour tout ensemble représenté :

Proposition 3.9. *Pour tous* $f_1, f_2 \in IDF_{\mathfrak{Z}\to\mathfrak{D}}$, $[\![f_1]\!] = [\![f_2]\!] \implies f_1 = f_2$.

Démonstration. Considérons $Z_1 \subseteq \mathbb{Z}^n$ et prouvons que $f_1(Z_1) \subseteq f_2(Z_1)$. Naturellement, on peut supposer que $f_1(Z_1) \neq \emptyset$ (sinon, l'inclusion est immédiate). Dans ce cas, il existe un vecteur $\boldsymbol{d} \in f_1(Z_1)$. Puisque $(f_2(Z))_Z$ forme une partition de $\mathbb{D}^n$, il existe Z_2 tel que $\boldsymbol{d} \in f_2(Z_2)$.

Prouvons que $Z_1 \subseteq Z_2$. On peut supposer que $Z_1 \neq \emptyset$. Soit $\boldsymbol{z}_1 \in Z_1$ et remarquons que $\boldsymbol{r}_1 = \boldsymbol{z}_1 + \boldsymbol{d} \in [\![f_1]\!]$, et que de $[\![f_1]\!] = [\![f_2]\!]$, on déduit $\boldsymbol{r}_1 \in [\![f_2]\!]$. Ainsi, il existe Z_2' tel que $\boldsymbol{r}_1 \in Z_2' + f_2(Z_2')$. Puisque $Z_2' \subseteq \mathbb{Z}^n$ et $f_2(Z_2') \subseteq \mathbb{D}^n$, on en déduit que $\boldsymbol{z}_1 \in Z_2'$ et $\boldsymbol{d} \in f_2(Z_2')$. Comme $(f_2(Z))_Z$ forme une partition de $\mathbb{D}^n$ et $\boldsymbol{d} \in f_2(Z_2) \cap f_2(Z_2')$, on en déduit $Z_2 = Z_2'$. En particulier, $\boldsymbol{z}_1 \in Z_2$, et on a montré que $Z_1 \subseteq Z_2$.

L'autre inclusion $Z_2 \subseteq Z_1$ est obtenue de manière symétrique. On a alors prouvé que $Z_1 = Z_2$. Ainsi, $f_1(Z_1) \subseteq f_2(Z_1)$ pour tout Z_1. Par symétrie, on en déduit que $f_1(Z) = f_2(Z)$ pour tout Z, et donc que $f_1 = f_2$. □

Notons qu'en pratique, cette unicité dépend de la représentation symbolique des ensembles dans $\mathfrak{Z}$ et $\mathfrak{D}$. En effet, si l'une de ces représentations n'est pas unique, alors on ne peut pas garantir qu'un $IDF_{\mathfrak{Z}\to\mathfrak{D}}$ sera unique.

Conclusion relative aux IDS et à GENEPI

Un des avantages des IDS est qu'ils permettent d'effectuer des opérations que l'on ne sait pas faire sur certaines logiques. Par exemple, il n'y a actuellement aucun algorithme pour calculer directement l'enveloppe convexe d'un ensemble défini dans FO $(\mathbb{R}, \mathbb{Z}, +, \leq)$; mais grâce à notre décomposition, le problème se réduit au calcul de l'enveloppe convexe d'ensembles définissables dans la logique de Presburger (par automates [BHMV94] ou ensembles semi-linéaires [GS66]), et de l'enveloppe convexe d'ensembles dans FO $(\mathbb{D}, +, \leq)$ (par unions finies d'ensembles convexes, en utilisant Fourier-Motzkin [Fou26, Mot51]). On peut étendre ce raisonnement à d'autres représentations symboliques, ainsi qu'à d'autres opérations, telles que la clôture par le haut (ou par le bas).

On pourrait s'attendre à un coût exagéré dû à l'utilisation simultanée de deux représentations symboliques, chacune avec son outil dédié. En effet, il y a toujours un temps d'exécution et une utilisation de l'espace mémoire minimaux, requis par tout logiciel, notamment pour son démarrage, son initialisation, et la construction des structures de données. Néanmoins, le cadre modulaire de GENEPI réduit ce besoin additionnel de ressources. De plus, en utilisant PRESTAF sur les entiers, on bénéficie des avantages considérables des automates partagés (forte réduction de l'utilisation de l'espace mémoire et tests d'égalité d'ensembles effectué en temps constant).

Globalement, cette séparation des entiers et des réels vise à accélérer le processus de développement d'autres logiciels, grâce à la facilité de réutilisation des plugins existants. On peut en effet tester la combinaison de n'importe quelle paire de plugins, du moment que l'un couvre les entiers et que l'autre manipule des réels (que l'on réduit trivialement à des décimaux). Ainsi, lorsque l'on souhaite implémenter de nouvelles représentations symboliques, on peut les tester directement dans GENEPI ; par exemple, une nouvelle représentation portant uniquement sur les réels pourra immédiatement être étendue avec les entiers, en la couplant à PRESTAF (ou à tout autre plugin gérant les entiers). L'autre sens (étendre un nouveau plugin entier avec des réels) est évidemment possible, de manière tout à fait symétrique.

L'outil GENEPI a récemment été intégré dans un environnement logiciel plus vaste, lui offrant de nouvelles fonctionnalités très agréables [LP09]. Cet environnement s'appelle TAPAS (Talence Presburger Arithmetic Suite) [TaP], et permet à l'utilisateur de GENEPI de manipuler directement des formules arithmétiques. En effet, un humain comprend généralement beaucoup plus vite un ensemble décrit par une formule que par un automate ; de même, pour fournir des jeux de données, il est généralement plus agréable et rapide d'exprimer les ensembles à représenter directement par des formules.

Logiques arithmétiques du premier ordre : conclusion

On a proposé une décomposition de cinq classes d'ensembles de vecteurs réels en unions finies de sommes d'entiers et de décimaux, ce qui a permis de caractériser ces classes. Cette décomposition nous a amené à définir deux nouvelles logiques (parmi les cinq), plus proches de la frontière de la décidabilité, en obtenant ainsi une logique mixte "maximale" (i.e. contenant les autres) qui reste cependant décidable. Cette décomposition peut encore nous amener à explorer d'autres sous-classes de l'arithmétique mixte, en facilitant certaines preuves d'équivalence et de décidabilité.

L'intérêt de manipuler des logiques mixtes les plus expressives possibles est de pouvoir représenter les configurations de systèmes dont les variables peuvent être entières et réelles. Pour ce faire, on donne un début d'implémentation avec les IDS, qui bénéficient du cadre de GENEPI et TAPAS.

Deuxième partie

Modèles de systèmes numériques

Modèles de systèmes numériques : introduction

Cette deuxième partie est consacrée à l'étude de modèles de systèmes, en vue de leur vérification. On présente divers modèles, en les classant selon leur type de variables (compteurs ou horloges, entier ou réel, etc.) et selon la logique définissant la façon dont sont manipulées ces variables. On s'intéresse également aux problèmes d'accessibilité dans ces modèles, et on donne des résultats, nécessitant parfois de se restreindre à des sous-classes de systèmes.

Le chapitre 4 présente les systèmes à compteurs, à variables entières, ou parfois, réelles. Le chapitre 5 fait un tour d'horizon des systèmes temporisés, et propose dans la section 5.4 un modèle de systèmes à compteurs temporisés, combinant compteurs entiers et horloges réelles. Enfin, le chapitre 6 est consacré à un modèle peu commun, dans lequel les compteurs peuvent être modifiés d'une valeur réelle choisie de façon non-déterministe à chaque franchissement de transition.

Chapitre 4

Systèmes à compteurs

Soit X un ensemble de n variables, appelées *compteurs* dans ce chapitre. Dans cette thèse, on étudie différentes classes de systèmes à compteurs, la plupart se basant sur la définition 4.1. Chacun de ces systèmes utilise un alphabet fini d'instructions opérant sur les compteurs, défini par $\mathscr{L}$. La façon dont cet alphabet d'étiquetage est défini caractérise précisément les classes de systèmes à compteurs.

Prenons tout d'abord un modèle de système à compteurs très général, sans spécifier sa sémantique (dans un premier temps) :

Définition 4.1. *Un* $\mathscr{L}$-Système à Compteurs *(ou, plus souvent, Système à Compteurs) est un couple* $\langle Q, T\rangle$ *tel que* Q *est un ensemble fini d'états de contrôle, et* $T \subseteq Q \times \mathscr{L} \times Q$ *est un ensemble fini de transitions.*

Un $\mathscr{L}$-Système à Compteurs est donc un $\mathscr{L}$-système (définition 1.7) dans lequel les variables sont appelées "compteurs". Les principales différences avec d'autres types de systèmes (par exemple, temporisés) se situeront au niveau de $\mathscr{L}$, mais aussi dans la sémantique.

Les deux sections suivantes présentent les premiers modèles en deux grands courants : d'abord, l'approche "machines à compteurs" (section 4.1), puis l'approche "réseaux de Petri" (section 4.2). Ensuite, la section 4.3 montre un modèle de systèmes à compteurs relationnels, puis la section 4.4 détaille un modèle de systèmes à compteurs fonctionnels. Enfin, la section 4.5 étudie des systèmes mixtes, comportant à la fois des compteurs entiers et des compteurs réels.

4.1 Des machines de Minsky aux machines à compteurs reversal-bornées

De Marvin Minsky en 1960 à Oscar H. Ibarra en 1978, les bases de ce qu'on appelle les machines à compteurs on été posées, principalement dans [Min61, Min67, Iba78]. Historiquement, le premier modèle est celui de la Machine de Minsky ; c'est généralement le plus élémentaire que l'on considère, et probablement le plus connu.

Une machine de Minsky possède un ensemble fini d'états de contrôle, et passe de l'un à l'autre en exécutant des instructions. Plus préciment, c'est un Système à Compteurs dans lequel les formules de $\mathscr{L}$ sont de la forme $(x' = x + 1)$, $(x' = x = 0)$, $(x > 0 \wedge x' = x - 1)$, ou $true$. Généralement, on considère qu'une machine de Minsky n'a que deux compteurs : c'est le plus petit modèle ayant la puissance d'une machine de Turing, et donc le problème d'accessibilité y est indécidable. En effet, si l'on enlève la possibilité de tester si l'un des deux compteurs vaut 0, alors ce problème devient décidable (et a fortiori, s'il n'y a qu'un seul compteur aussi). Du fait de la simplicité de ce modèle, la plupart des preuves d'indécidabilité d'autres modèles sont faites en exhibant une simulation d'une machine de Minsky.

Afin de faciliter la modélisation et de généraliser les machines de Minsky, les machines à compteurs ont été définies. En effet, une machine à compteurs peut incrémenter ou décrémenter d'une valeur entière donnée (pas seulement 1), tout en posant des conditions sur le franchissement des transitions. Ces conditions, appelées *gardes*, permettent de tester si la valeur d'un compteur est au moins égale, ou tout simplement égale, à une constante entière donnée. Les incréments et décréments, quant à eux, peuvent être vus géométriquement comme une translation du vecteur des valuations de compteurs. Ainsi, les transitions d'une machine à compteurs sont appelées des *translations gardées*, et sont définies comme suit :

Définition 4.2. *Une* translation gardée *est une fonction* $t : \mathbb{N}^n \longrightarrow \mathbb{N}^n$ *telle qu'il existe* $\succ \in \{=, \geq\}^n$, $\mu \in \mathbb{N}^n$, *et* $\gamma \in \mathbb{Z}^n$ *avec* $\mu + \gamma \geq 0$ *et* $\mathsf{dom}(t) = \{\mathbf{v} \in \mathbb{N}^n \mid \mathbf{v} \succ \mu\}$, *de sorte que* $t(\mathbf{v}) = \mathbf{v}'$, *où* $\mathbf{v}$ *et* $\mathbf{v}'$ *sont les valeurs des compteurs avant et après la translation gardée.*

Définition 4.3. *Une* machine à compteurs *est un* $\mathscr{L}$*-Système à Compteurs pour lequel* $\mathscr{L}$ *est un ensemble de translations gardées.*

Intuitivement, pour une transition d'une machine à compteurs, μ est la condition de la garde, et γ est la "longueur" de la translation (d'un point de vue géométrique). On note parfois une translation gardée par ses trois vecteurs caractéristiques, c'est-à-dire $(\succ, \mu, \gamma)$.

Il est à noter qu'une translation gardée est bel et bien une relation binaire correspondant à la définition d'un $\mathscr{L}$-Système à Compteurs. En effet, pour une translation $t : \mathbb{N}^n \longrightarrow \mathbb{N}^n$ et deux valuations de compteurs $\mathbf{v}$ et $\mathbf{v}'$, on a $(\mathbf{v}, \mathbf{v}') \in t$ si et seulement si $\mathbf{v} \in \mathsf{dom}(t)$ et $\mathbf{v}' = t(\mathbf{v})$. On peut donc écrire une translation gardée $(\succ, \mu, \gamma)$ sous forme d'une contrainte $\bigwedge_{i\in[1..n]}(\mu_i \succ_i v_i \ \wedge \ v'_i = v_i + \gamma_i)$.

Ce modèle de machines à compteurs (définition 4.3) est en fait équivalent à celui des machines de Minsky, et possède donc lui aussi la puissance des machines de Turing. Cependant, Ibarra a introduit dans [Iba78] une restriction sur le comportement des machines à compteurs, rendant ainsi ce modèle décidable. Cette restriction est appelée "reversal-boundedness" en anglais, et majore le nombre d'alternances entre les phases croissantes et décroissantes des valeurs de chaque compteur. On appelera parfois de telles alternances des "renversements" (pour "reversal", en anglais).

Soit $\mathcal{M}$ une machine à compteurs, initialisée par une configuration donnée $c_0 \in Q \times \mathbb{N}^n$. On dit que $\mathcal{M}$ est *r-reversal-bornée*, avec $r \in \mathbb{N}$, si dans toutes les exécutions de $\mathcal{M}$ démarrant dans la configuration initiale s_0, chaque valuation de compteur alterne au plus r fois entre une suite non-décroissante et une suite non-croissante. Cette contrainte peut être vérifiée dans le système de transitions de $\mathcal{M}$, pourvu que l'on étende sa notion de configuration, en y ajoutant le mode (croissant ou non) et le nombre d'alternances.

Dans [FS08], la notion de compteur k-reversal-borné est étendue à celle de k-reversal-b-borné, en spécifiant que la contrainte ne s'applique que lorsque les valuations de compteurs sont supérieures à une borne b donnée. La définition originelle correspond donc au cas $b = 0$. Cette extension définit en fait une classe de Systèmes à Compteurs dont l'ensemble des configurations accessibles est effectivement définissable dans la logique de Presburger, ce qui donne une caractérisation plus claire et une hiérarchie mieux définie. Pour plus de détails à ce sujet (encodages, définitions formelles, propriétés, classification de Systèmes à Compteurs, etc.), voir la thèse d'Arnaud Sangnier [San08].

L'avantage principal des machines à compteurs reversal-bornées est qu'elles constituent une sous-classe décidable et assez large. On verra par la suite que d'autres classes de Systèmes à Compteurs, indécidables dans le cas général, deviennent décidables lorsqu'on les contraint à être reversal-bornées.

4.2 Réseaux de Petri et VASS

En 1962, dans sa thèse de doctorat [Pet62], Carl Adam Petri a défini ce que l'on appelle aujourd'hui les réseaux de Petri. Les réseaux de Petri constituent un forma-

lisme des plus utilisés pour la modélisation de systèmes, et ce dans de nombreux domaines, que ce soit au niveau de la recherche théorique ou des applications industrielles. Au départ défini comme un moyen de gérer l'aspect concurrent des communications entre systèmes, un réseau de Petri peut également être considéré comme un système à compteurs. En effet, un autre nom pour les réseaux de Petri est P/T nets (réseaux Places/Transitions) : ils ressemblent à des automates, qui relient des places au moyen de transitions, tout en ayant une information quantitative en plus d'un flot de contrôle (à l'instar d'un système à compteurs).

Il y a trois différences principales entre les réseaux de Petri et les systèmes à compteurs.

Premièrement, en considérant les systèmes comme des graphes, les "transitions"[1] d'un réseau de Petri ne sont pas les arcs du graphe, mais un type de sommet : un réseau de Petri est en effet un graphe biparti, dont les sommets sont partitionnés en un ensemble de places et un ensemble de "transitions". L'information circule sous la forme de jetons (analogiquement à des compteurs), qui sont stockés dans les places et qui sont nécessaires au franchissement des "transitions". Les arcs du graphe relient donc les places et les transitions entre elles.

Deuxièmement, les arcs ne sont pas étiquetés comme c'est le cas dans les systèmes à compteurs. Ce sont en fait les "transitions" du réseau de Petri qui se comportent implicitement comme des gardes, en testant si le nombre de jetons dans les places précédentes est supérieur ou égal à une constante donnée, et comme des opérations, en soustrayant un certain nombre de jetons aux places précédentes et en l'additionnant aux places suivantes.

Troisièmement, lors d'une exécution, un réseau de Petri ne se trouve pas dans une place en particulier ; en effet, on peut décider de franchir n'importe quelle "transition" activable dans tout le réseau, pas seulement depuis une place donnée. Une place n'est donc pas tout à fait un état de contrôle.

Voici la définition formelle d'un réseau de Petri :

Définition 4.4. *Un* réseau de Petri *est un quadruplet* $\mathcal{R} = \langle P, T, W^-, W^+ \rangle$ *dans lequel :*
- P *est un ensemble fini de places*
- T *est un ensemble fini de "transitions" tel que* $P \cap T = \emptyset$
- $W^- : T \longrightarrow \mathbb{N}^{|P|}$ *est l'application d'incidence arrière*
- $W^+ : T \longrightarrow \mathbb{N}^{|P|}$ *est l'application d'incidence avant*

Dans chaque place et à tout moment, il y a un nombre positif ou nul de jetons ; un *marquage* du réseau est un vecteur d'entiers indiquant le nombre de jetons présents dans

[1] Le mot "transition" pour les réseaux de Petri est mis entre guillemets, puisqu'il ne désigne pas la même chose qu'une transition dans un automate fini, contrairement aux autres systèmes vus dans cette thèse.

chaque place.

Le comportement d'un réseau de Petri $\mathcal{R}$ peut être décrit par un système de transitions $TS(\mathcal{R}) = \langle \mathbb{N}^{|P|}, T, \rightarrow \rangle$ défini comme suit :
- $\mathbb{N}^{|P|}$ *est l'ensemble des marquages*
- T *est un alphabet d'étiquetage (correspondant aux noms des "transitions" de* $\mathcal{R}$*)*
- $\rightarrow \subseteq \mathbb{N}^{|P|} \times T \times \mathbb{N}^{|P|}$ *est une relation telle que pour tout* $M, M' \in \mathbb{N}^{|P|}$ *et pour tout* $t \in T$, $M \xrightarrow{t} M'$ *si et seulement si* $M \geq W^-(t)$ *et* $M' = M - W^-(t) + W^+(t)$

Les réseaux de Petri ont de nombreuses propriétés intéressantes et bien connues, notamment la décidabilité du problème d'accessibilité, qui nous intéressera par la suite.

Initiés par Karp et Miller en 1969 [KM69], puis étendus par Hopcroft et Pansiot dix ans plus tard [HP79], les VASS (Vector Addition Systems with States[2]) constituent un modèle équivalent à celui des réseaux de Petri. Un VASS est en fait une machine à compteurs (voir la définition 4.3) dans laquelle les gardes n'ont pas le droit au test d'égalité, mais seulement aux comparaisons "au moins égal".

Formellement :

Définition 4.5. *Un* VASS *est une machine à compteurs dans laquelle les translations gardées sont de la forme* $(\geq, \mu, \gamma)$.

Le fait d'interdire les tests d'égalité (entre un compteur et une constante) permet de rendre cette machine à compteurs décidable (pour le problème 1.9). En effet, il faut se rappeler qu'une machine à compteurs est avant tout une machine de Minsky, et que le problème d'accessibilité devient indécidable sur ce modèle dès qu'on a deux compteurs avec test d'égalité.

Le modèle des VASS est de plus très intéressant, puisqu'il permet de faire un lien direct et fort entre les systèmes usuels (basés sur des automates) et les réseaux de Petri (un peu plus éloignés). Puisque l'on sait qu'un VASS a toujours un réseau de Petri équivalent (et vice versa), alors on saura comparer plus facilement de nouveaux systèmes, comme ceux définis dans cette thèse, avec ce modèle incontournable que sont les réseaux de Petri.

On présente ici les encodages pour passer d'un VASS à un réseau de Petri, et inversement. La preuve de l'équivalence de ces deux modèles suit les idées suivantes.

N.B. : Pour faciliter la compréhension, on utilisera pour ces encodages la notation $[\![.]\!]$ comme une valuation ; ainsi, le marquage d'une place p_i sera noté $[\![p_i]\!]$ et la valeur d'un compteur c_i sera notée $[\![c_i]\!]$.

[2]En français, SAVE : Systèmes à Addition de Vecteurs avec États

Encodage d'un VASS en un réseau de Petri.

Soient q_i les états de contrôle d'un VASS donné. On crée une place de réseau de Petri p_i pour chaque q_i, dans laquelle il y aura en tout temps au plus un seul jeton ; l'ensemble de ces places p_i devra vérifier en tout temps l'égalité $\sum_i [\![p_i]\!] = 1$. On crée également une place p_i^c pour chaque compteur c_i du VASS à encoder. On utilise alors les encodages de la figure 4.1 pour passer d'un VASS à un réseau de Petri ayant la même relation d'accessibilité.

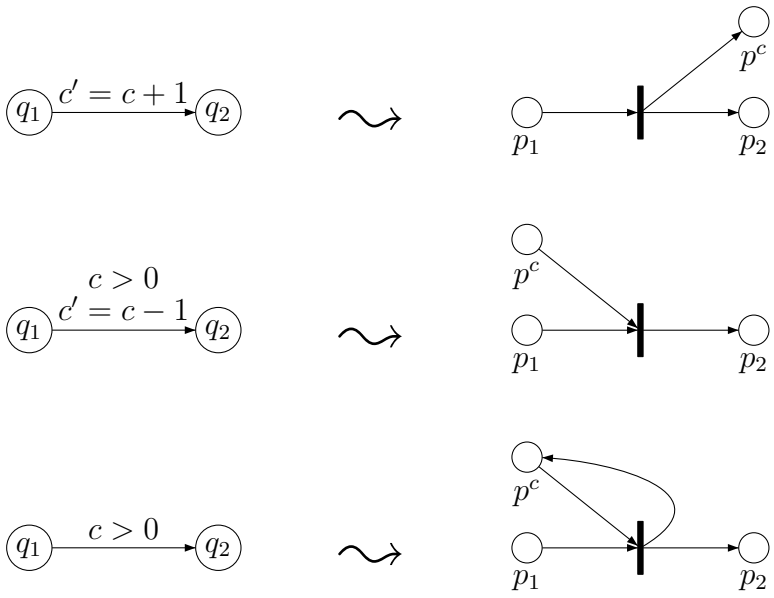
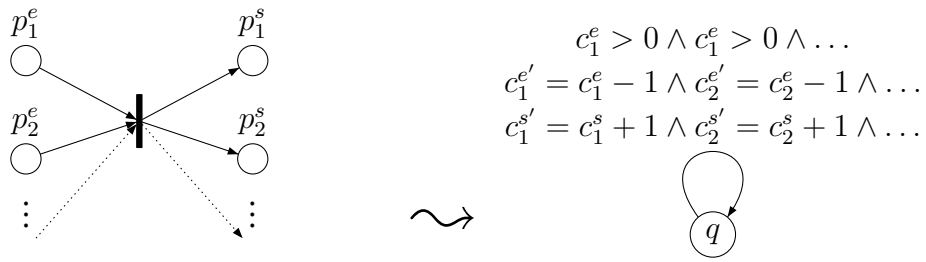


FIG. 4.1: Encodage d'un VASS en un réseau de Petri

Encodage d'un réseau de Petri en un VASS.

Pour chaque transition du réseau de Petri à encoder, soit $I, J \subset \mathbb{N}$ les ensembles des indices des places entrantes p_i^e et sortantes p_j^s relatives à cette transition, avec $i \in I$ et $j \in J$. On crée des compteurs c_i^e et c_j^s tels que $\forall i \in I, [\![p_i^e]\!] = [\![c_i^e]\!]$ et $\forall j \in J, [\![p_j^s]\!] = [\![c_j^s]\!]$. On définit alors le VASS $\langle Q, T \rangle$ tel que $Q = \{q\}$ et T est construit, pour chaque transition, comme indiqué sur la figure 4.2 (chaque transition du réseau de Petri donne lieu à une nouvelle boucle sur l'état de contrôle q).



En particulier :

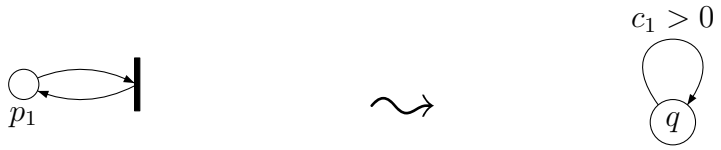


FIG. 4.2: Encodage d'un réseau de Petri en un VASS

4.3 Systèmes à compteurs relationnels de Comon-Jurski

Lors de son doctorat dirigé par Hubert Comon-Lundh, Yan Jurski a défini un modèle de Systèmes à Compteurs dont les formules étiquetant les transitions sont des relations comparant les valeurs de compteurs deux à deux et/ou avec une constante. Les formules sont donc un peu plus riches que dans les modèles précédents, puisqu'on a la possibilité de comparer directement la valeur de deux compteurs. De plus, ces comparaisons se font sur les valeurs des compteurs avant la transition, mais aussi après, ou les deux à la fois. De telles relations apportent alors une nouvelle source de non-déterminisme, puisque pour une configuration donnée, le franchissement d'une même transition pourra conduire à plusieurs configurations différentes (voire une infinité).

Une autre particularité de ce modèle est qu'il autorise les compteurs à prendre leurs valeurs dans les entiers, mais aussi dans les rationnels ou les réels, et ce, restreints aux nombres positifs ou non : cela donne donc le choix entre plusieurs types de compteurs. On a également une factorisation agréable de la structure de contrôle, puisque les transitions peuvent être étiquetées par des conjonctions de formules (au lieu d'une simple petite formule par transition comme dans les machines de Minsky).

Ce modèle a été d'abord publié dans [CJ98], puis réutilisé dans [CJ99] pour étendre les résultats ; cependant, on se basera sur la version longue de ce premier article, qui inclut les extensions de résultats. D'autres articles ont repris ce modèle par la suite, comme [BIL09] (version longue de leur article à ICALP'06).

Formellement, voici la définition de ce modèle. Soit $\mathcal{D} \in \{\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{Q}_+, \mathbb{R}, \mathbb{R}_+\}$. Soit $\mathscr{L}_{CJ}$ la logique composé de conjonctions finies $F(X, X') = \bigwedge(y \bowtie z + k)$, où $k \in \mathcal{D}$, $\bowtie \in \{\geq, >, =, <, \leq\}$, $y \in X \cup X'$, et $z \in X \cup X' \cup \{0\}$.

Définition 4.6. *Un* système à compteurs relationnel *est un Système à Compteurs $\langle Q, T\rangle$ pour lequel $\mathscr{L} = \mathscr{L}_{CJ}$.*

Le comportement d'un tel système peut naturellement être donné par un système de transitions des plus classiques, avec des configurations $(q, \mathbf{v}) \in Q \times \mathcal{D}^n$ et des transitions $(q, \mathbf{v}) \xrightarrow{\phi} (q', \mathbf{v}')$ définies si et seulement si $(q, \phi, q') \in T$ et $(\mathbf{v}, \mathbf{v}') \models \phi$.

Dans [CJ98], les auteurs donnent une caractérisation de la relation d'accessibilité de ce modèle. Ils montrent en effet que l'ensemble des paires de configurations $\Big((q, \mathbf{v}), (q', \mathbf{v}')\Big)$ telles que $(q, \mathbf{v}) \rightarrow^* (q', \mathbf{v}')$ est définissable dans la logique additive du premier ordre sur $\mathcal{D}$. Par exemple, si l'on considère les compteurs définis sur $\mathbb{N}$, alors la relation d'accessibilité est définissable dans Presburger.

Cependant, ce résultat ne s'applique que si les points fixes des boucles (i.e. des circuits) dans le système de transitions, tout au long des exécutions possibles, restent définissables dans cette même logique additive. Les auteurs identifient alors une classe de systèmes pour laquelle cette contrainte de point fixe est toujours vérifiée : c'est la classe des systèmes à compteurs relationnels plats.

Un Système à Compteurs est *plat* si, et seulement si, le graphe de sa structure de contrôle est plat, c'est-à-dire qu'il ne contient pas de circuits élémentaires imbriqués. On rappelle qu'un circuit est élémentaire si et seulement s'il contient au plus une fois chaque sommet du graphe (sauf le premier et le dernier, qui sont confondus). Une autre façon de le voir est sur le système de transitions : pour toute configuration c, il existe au plus une exécution passant au moins deux fois par c (et au plus une fois par les autres configurations entre deux occurences de c). Formellement :

Définition 4.7. *Un Système à Compteurs $\langle Q, T\rangle$ est* plat *si et seulement si chaque état de contrôle $q \in Q$ apparaît au plus dans un seul circuit élémentaire.*

Cette idée de calculer les itérations de circuits dans un systèmes à compteurs est à la base de la théorie de l'accélération, expliquée dans la section suivante. On y retrouvera le fait que ce calcul fonctionnera particulièrement bien quand on se restreint aux modèles plats.

Comme démontré dans la première partie de cette thèse, les logiques additives du premier ordre sur les entiers et/ou les réels sont décidables. Ainsi, puisqu'elles permettent de caractériser la relation d'accessibilité des systèmes à compteurs relationnels plats, on en déduit immédiatement que les systèmes à compteurs relationnels plats sont

décidables.

Un autre intérêt de ce modèle est qu'il a permis de montrer que les automates temporisés (définis dans le chapitre 5) sont aplatissables, c'est-à-dire que pour tout automate temporisé, on peut calculer un automate temporisé plat donc la relation d'accessibilité est la même [CJ99]. La notion de système aplatissable sera développée dans la section suivante.

4.4 Systèmes à compteurs fonctionnels de Finkel-Leroux

Comme son nom l'indique, un système à compteurs fonctionnel est un Système à Compteurs dont les transitions sont étiquetées par des fonctions, et non pas des relations générales (comme dans le cas des systèmes à compteurs relationnels de la section précédente). On rappelle qu'une fonction est une relation par laquelle chaque élément a au plus une seule image, c'est-à-dire qu'une relation binaire f est une fonction si et seulement si $\forall x, \forall y_1, y_2, (x, y_1) \in f \wedge (x, y_2) \in f \implies y_1 = y_2$. Dans le cadre des Système à Compteurs, cela signifie que pour une valuation des compteurs $\mathbf{v}$ connue avant le franchissement d'une transition t, la valuation $\mathbf{v}'$ (obtenue après franchissement de t) sera unique, contrairement aux systèmes à compteurs relationnels, pour lesquels on obtiendra un ensemble de valuations $\mathbf{v}'$ possibles.

Formellement :

Définition 4.8. *Un* système à compteurs fonctionnel *est un Système à Compteurs $\langle Q, T\rangle$ tel que toutes les formules de $\mathscr{L}$ définissent des fonctions. Plus précisément, pour tout $(q, \phi, q') \in T$, il existe une fonction partielle $f : \mathbb{N}^n \longrightarrow \mathbb{N}^n$ telle que $(\mathbf{v}, \mathbf{v}') \models \phi$ si et seulement si $\mathbf{v} \in \mathsf{dom}(f)$ et $\mathbf{v}' = f(\mathbf{v})$.*

4.4.1 Systèmes à compteurs affines

Durant son doctorat encadré par Alain Finkel, Jérôme Leroux a étudié ces systèmes à compteurs fonctionnels. La définition sur laquelle on se basera ici est celle publiée dans [FL02], correspondant au cas particulier des systèmes à compteurs fonctionnels Presburger-affines, c'est-à-dire dont les formules étiquetant les transitions sont des fonctions affines dont le domaine est Presbuger-définissable. Le fait que ces fonctions soient affines est très important, car on ne pourra donc pas multiplier entre elles les valeurs de deux compteurs. Ainsi, on devine déjà que les transitions de tels systèmes seront définissables dans Presburger. Notons que dans les systèmes à compteurs fonctionnels considérés ici, les compteurs prennent leurs valeurs seulement dans $\mathbb{N}$.

Formellement :

Définition 4.9. *Une fonction partielle* $f : \mathbb{N}^n \longrightarrow \mathbb{N}^n$ *est* Presburger-affine *si et seulement s'il existe un triplet* $(\psi, A, \mathbf{b})$ *tel que* $\psi \in \mathrm{Presb}(X)$ *où* $X = \{x_1, \ldots, x_n\}$, A *est une matrice carrée de taille* n *à coefficients dans* $\mathbb{Z}$, *et* $\mathbf{b} \in \mathbb{Z}^n$, *qui vérifie* $\mathsf{dom}(f) = [\![\psi]\!]_X$ *et* $f(\mathbf{v}) = A.\mathbf{v} + \mathbf{b}$, *pour tout* $\mathbf{v} \in \mathsf{dom}(f)$.

Définition 4.10. *Un* système à compteurs affine *est un système à compteurs fonctionnel dont chaque fonction est Presburger-affine.*

Ce modèle de système à compteurs fonctionnel est en fait incomparable avec celui de la section précédente (définition 4.6), qui est relationnel. En effet, bien qu'une fonction ne soit qu'un cas particulier d'une relation, la richesse des formules est bien plus importante dans les systèmes à compteurs affines (définition 4.10) que dans les systèmes à compteurs relationnels (définition 4.6) : presque toutes les fonctions affines sont autorisées sur l'ensemble des compteurs, dans un cas, alors que seules de petites comparaisons entre deux compteurs et une constante sont permises, dans l'autre. Par exemple, une formule $x'_1 = x_1 + x_2 + x_3$ ne peut pas étiqueter une transition d'un système à compteurs relationnel, mais affine, oui. A contrario, une formule $x' > x$ ne peut pas étiqueter une transition d'un système à compteurs affine, mais relationnel, oui.

Le comportement d'un système à compteurs fonctionnel (ou affine) $\mathcal{S} = \langle Q, T \rangle$ est donné par un système de transitions $TS(\mathcal{S}) = \langle C, \rightarrow \rangle$, où $C = Q \times \mathbb{N}^n$ est l'ensemble des configurations, et la relation $\rightarrow \subseteq C \times T \times C$ est telle que pour toutes configurations $(q, \mathbf{v}), (q', \mathbf{v}') \in C$, on a $(q, \mathbf{v}) \xrightarrow{t} (q', \mathbf{v}')$ si et seulement si $t = (q, f, q') \in T$ et $\mathbf{v}' = f(\mathbf{v})$.

Là encore, ces deux modèles de systèmes à compteurs (fonctionnels et affines) ont la puissance des machines de Turing et sont donc indécidables. Cependant, outre le fait qu'ils permettent un raccourci important par rapport à la syntaxe des machines à compteurs, ils ont l'avantage d'être accélérables. En effet, la théorie de l'accélération développée dans [Ler03] permet de calculer l'ensemble d'accessibilité d'un système à compteurs affine. La section suivante présente rapidement l'accélération.

4.4.2 Accélération

L'accélération est une technique qui permet de décider, dans certains cas (détaillés ci-après), le problème d'accessibilité d'une configuration (problème 1.9). Cette technique consiste à "accélérer" le calcul des exécutions possibles d'un système, en calculant la fermeture transitive et réflexive de ses circuits en une seule fois. Cette idée,

d'abord présentée dans la thèse de Bernard Boigelot [Boi98] sous forme de calcul de *meta-transitions*, a été précisée et automatisée dans la thèse de Jérôme Leroux [Ler03] spécifiquement pour les systèmes à compteurs affines.

Plus précisément, pour un système à compteurs affine $\mathcal{S} = \langle Q, T \rangle$ avec $T \subseteq Q \times \Sigma \times Q$, on va chercher à calculer directement les limites des circuits étiquetés par $\sigma \in \Sigma^*$, qui correspondent en fait à des compositions de transitions (d'où le terme "meta-transition"). On cherche donc à calculer la fonction correspondant à une suite de transitions, c'est-à-dire la composition des fonctions associées à chacune des transitions de la séquence considérée. La fonction à accélérer est donc une fonction affine f, égale à la composition de plusieurs fonctions affines. On définit alors l'accélération de f comme sa fermeture réflexive et transitive $f^*(X) = \bigcup_{i \in \mathbb{N}} f^i(X)$.

Malheureusement, cette technique d'accélération ne donne pas toujours de résultat en un temps fini. Cependant, on connaît une condition suffisante pour que ce calcul termine : il suffit que la fonction à accélérer, correspondant à une séquence de transitions, soit à monoïde fini. On dit qu'une fonction affine $f(X) = A.X + \mathbf{b}$ est à monoïde fini si l'ensemble $\langle A \rangle = \{A^i \mid i \in \mathbb{N}\}$ est fini. En pratique, on utilise la propriété suivante : $\langle A \rangle$ est fini si et seulement s'il existe $j, k \in \mathbb{N}$ tels que $A^j = A^{j+k}$. Cette propriété signifie que le monoïde $\langle A \rangle$ est un ensemble contenant $j + k$ éléments tels que pour tout $i \geq j$, $A^i = A^{j+q.k+r}$, avec $q \in \mathbb{N}$ et $r < k$. On cherche donc à calculer cet entier j à partir duquel les puissances de A^i, $i \geq j$, sont congrues modulo k.

Ce calcul étant exponentiel, en le multipliant par le nombre de circuits possibles (potentiellement infini), on revient rapidement à un calcul trop complexe (voire impossible) ; c'est pourquoi on s'intéresse à la recherche automatique des circuits à accélérer. En effet, on ne connaît pas a priori tous les circuits possibles, et il y a une infinité de compositions de transitions envisageables ! Il faut donc réussir non seulement à calculer l'accélération d'un circuit donné, mais aussi à trouver les "bons" circuits à accélérer. Ce critère de sélection de circuits est donné par une heuristique qui consiste à fixer arbitrairement la taille maximale p des séquences de transitions, ce qui réduit l'espace d'exploration à un nombre fini de débuts d'exécutions (i.e. de séquences de transitions). Parmi ces séquences de transitions, on peut ensuite détecter les cycles de longueur inférieure ou égale à p. Tout ceci est détaillé dans [Ler03], et donne notamment un moyen de réduire ce nombre de séquences (généralement exponentiel en fonction de p) à un nombre polynomial en p.

Cette procédure d'accélération, couplée à l'heuristique de recherche des circuits, est implémentée dans l'outil FAST (voir [BFLP08] pour la publication la plus récente). Cet outil permet en effet, au moyen d'automates binaires, de représenter symboliquement

l'ensemble d'accessibilité d'un système à compteurs affine. On lui fournit une description textuelle du système, respectant une grammaire simple, puis on peut vérifier des propriétés de sûreté, en définissant un ensemble de configurations que l'on ne souhaite jamais atteindre.

Plusieurs cas de figure peuvent apparaître : si, depuis la configuration initiale donnée, FAST parvient à atteindre une des configurations "dangereuses", alors il s'arrête et répond que le système n'est pas sûr. Si FAST parvient à calculer l'ensemble (parfois infini) des configurations accessibles sans y trouver aucune configuration "dangereuse", alors il s'arrête et répond que le système est sûr. Sinon, l'outil ne s'arrêtera pas : on peut alors essayer de guider la recherche des circuits à accélérer, en réglant certains paramètres ou en spécifiant une autre stratégie.

Dans [BFLS05], on trouve une caractérisation des systèmes pour lesquels la procédure d'accélération termine : les systèmes à compteurs affines *aplatissables*. C'est une notion intéressante pour plusieurs raisons : tout d'abord, elle porte sur la structure même du système de transitions, et non pas sur un calcul complexe de compositions des fonctions d'une suite de transitions difficile à trouver. Ensuite, c'est une condition nécessaire et suffisante à la terminaison de l'accélération, ce qui permet d'identifier précisément les systèmes pour lesquels on pourra attendre une réponse de l'algorithme. Enfin, c'est une classe de systèmes qui est assez large, comme en témoignent les exemples de démonstration de l'outil FAST[3]. Cependant, savoir si un système est aplatissable est malheureusement un problème indécidable.

Une système plat est, comme indiqué dans la définition 4.7, un système dont le graphe ne contient pas de boucles imbriquées. Un système est dit *aplatissable* s'il existe un système plat ayant le même ensemble d'accessibilité. Un *aplatissement* d'un système est donc un dépliage partiel de sa structure de contrôle, afin de le rendre plat.

Cette notion d'aplatissement possède en fait quelques variantes, détaillées formellement dans [DFGvD06]. On distingue principalement trois sortes de systèmes aplatissables, chacune étant relative à une configuration initiale donnée. Elles sont décrites informellement ici, en allant de celle qui pose la contrainte la plus faible à celle qui pose la contrainte la plus forte.

Premièrement, les systèmes Post**-aplatissables*, pour lesquels il existe un système plat ayant le même ensemble d'accessibilité.

Deuxièmement, les systèmes *trace-aplatissables*, pour lesquels il existe un système plat ayant les mêmes traces (une trace est un mot étiquetant une exécution).

Troisièmement, les systèmes *bisimulation-aplatissables*, pour lesquels il existe un

[3]Ces exemples de systèmes ainsi que leur banc d'essai sur FAST sont disponibles sur internet, à l'adresse `http ://www.lsv.ens-cachan.fr/Software/fast/example.php`

système plat dans lequel, depuis chaque configuration du système de départ, si une transition est disponible alors elle l'est aussi dans le système plat, et ce, depuis une configuration identique.

4.5 Systèmes à compteurs mixtes

On peut également chercher à définir des systèmes à compteurs dans lesquels certains compteurs sont discrets et d'autres sont denses. En effet, chaque représentation (discret/entier ou dense/réel) a ses avantages et inconvénients. Les compteurs entiers sont plus faciles à représenter (voir chapitre 3) et sont très étudiés depuis environ un demi-siècle (contre seulement quelques articles sur les compteurs réels). Les compteurs réels, eux, apportent une information plus précise, tout en bénéficiant d'une arithmétique de complexité moindre (par exemple, la satisfaisabilité de FO $(\mathbb{R}, +, \leq)$ est double-exponentielle, et celle de FO $(\mathbb{N}, +, \leq)$ est triple-exponentielle).

Dans cette section, on s'intéresse à la combinaison de ces deux types de compteurs dans un même modèle. Pour cela, on étend le modèle des systèmes à compteurs affines (définition 4.10) avec un ensemble de compteurs réels.

Définition 4.11. *Une fonction partielle $f : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ est* mixte-affine *si et seulement s'il existe un triplet $(\psi, A, \mathbf{b})$ tel que ψ est définissable dans FO $(\mathbb{R}, \mathbb{Z}, +, \leq)$ avec pour variables libres $X = \{x_1, \ldots, x_n\}$, A est une matrice carrée de taille n à coefficients dans $\mathbb{Q}$, et $\mathbf{b} \in \mathbb{Q}^n$, qui vérifie $\mathsf{dom}(f) = [\![\psi]\!]_X$ et $f(\mathbf{v}) = A.\mathbf{v} + \mathbf{b}$, pour tout $\mathbf{v} \in \mathsf{dom}(f)$.*

Notons que l'on choisit A et $\mathbf{b}$ rationnels et non pas réels, car ils devront être spécifiés dans la description du modèle, de façon finie (typiquement, dans un fichier texte fourni en entrée de l'outil). Cela n'aurait donc aucun sens de vouloir donner des nombres irrationnels sous forme de chiffres avec une précision très limitée.

Définition 4.12. *Un* système à compteurs mixte *est un système à compteurs fonctionnel dont chaque fonction f est mixte-affine.*

Tout système à compteurs mixte peut en fait contenir un système à compteurs affine, si l'on contraint certaines variables à être entières (en utilisant des fonctions Presburger-affines plutôt que mixte-affines, sur certaines composantes). On va donc considérer les systèmes à compteurs mixtes comme une version "réelle" des systèmes à compteurs affines, dans lesquels on peut distinguer un sous-système "entier". Puisque l'on se place dans la logique mixte FO $(\mathbb{R}, \mathbb{Z}, +, \leq)$ (qui contient Presburger) et non pas dans une logique purement réelle, on peut en effet manipuler certaines variables comme des entiers,

en utilisant les définitions 4.11 et 4.12.

On rappelle que pour une matrice A, son monoïde est défini par $\langle A \rangle = \{A^i \mid i \in \mathbb{N}\}$. Une propriété d'un monoïde fini $\langle A \rangle$ est qu'il existe $j, k \in \mathbb{N}$ tels que $A^j = A^{j+k}$.

Afin d'implémenter l'accélération d'une fonction mixte-affine (ou même, Presburger-affine), il faut s'assurer que l'on peut décider si son monoïde est fini. En effet, les matrices A utilisées dans les fonctions Presburger-affines sont à coefficients entiers, mais celles que l'on utilise pour les fonctions mixtes-affines sont à coefficients rationnels. C'est en fait le cas, puisque ce problème est décidable même quand les coefficients sont rationnels [Jac78, Ler03].

Tout comme pour les fonctions Presburger-affines, on définit l'accélération f^* d'une fonction mixte-affine f par : $f^*(X) = \bigcup_{i \in \mathbb{N}} f^i(X)$.

On montre à présent que le théorème principal de la théorie de l'accélération, issu de [FL02], s'étend directement aux systèmes à compteurs mixtes.

Théorème 4.13. *Soit $f = A.X + \mathbf{b}$ une fonction mixte-affine dont le monoïde $\langle A \rangle$ est fini. Alors la relation $\mathbf{v}' \in f^*(\mathbf{v})$ est définissable dans FO $(\mathbb{R}, \mathbb{Z}, +, \leq)$.*

Démonstration. Considérons une fonction mixte-affine $f = (\psi, A, \mathbf{b})$. On note $\overline{f} = (true, A, \mathbf{b})$ la fonction totalement définie associée à f. Puisque le monoïde $\langle A \rangle$ est fini, il existe deux entiers positifs j et k tels que $A^{j+k} = A^j$. On remarque que $\forall t \in \mathbb{N}$, $\overline{f}^t(\mathbf{v}) = A^t.\mathbf{v} + \sum_{0 \leq i < t-1} A^i.\mathbf{b}$.

Prouvons que $\overline{f}^j(\overline{f}^k(\mathbf{v})) = \overline{f}^j(\mathbf{v}) + A^j.\overline{f}^k(0)$:

$$\begin{aligned}
\overline{f}^j(\overline{f}^k(\mathbf{v})) &= A^j.\overline{f}^k(\mathbf{v}) + \sum_{i=0}^{j-1} A^i.\mathbf{b} \\
&= A^j.A^k.\mathbf{v} + A^j.\sum_{i=0}^{k-1} A^i.\mathbf{b} + \sum_{i=0}^{j-1} A^i.\mathbf{b} \\
&= A^j.\mathbf{v} + A^j.\sum_{i=0}^{k-1} A^i.\mathbf{b} + \sum_{i=0}^{j-1} A^i.\mathbf{b} \qquad \text{(car } \langle A \rangle \text{ est fini)} \\
&= \overline{f}^j(\mathbf{v}) + A^j.\overline{f}^k(0)
\end{aligned}$$

Ainsi, pour tout $q \in \mathbb{N}$, on a $\overline{f}^{j+q.k}(\mathbf{v}) = \overline{f}^j(\mathbf{v}) + q.A^j.\overline{f}^k(0)$. On prouve cette égalité par induction sur q :

- $q = 0$: trivial
- $q = 1$ (cas de base) : on vient de le montrer, car $\overline{f}^{j+k}(\mathbf{v}) = \overline{f}^j(\overline{f}^k(\mathbf{v}))$
- $n \geq 1$ (hypothèse d'induction) : $\overline{f}^{j+n.k}(\mathbf{v}) = \overline{f}^j(\mathbf{v}) + n.A^j.\overline{f}^k(0)$

– $q = n + 1$ (induction) :

$$\begin{aligned}
\overline{f}^{j+(n+1).k}(\mathbf{v}) &= \overline{f}^{j+n.k}(\overline{f}^{k}(\mathbf{v})) \\
&= \overline{f}^{j}(\overline{f}^{k}(\mathbf{v})) + n.A^{j}.\overline{f}^{k}(0) && \text{(par l'hypothèse d'induction)} \\
&= \overline{f}^{j}(\mathbf{v}) + A^{j}.\overline{f}^{k}(0) + n.A^{j}.\overline{f}^{k}(0) && \text{(par le cas de base)} \\
&= \overline{f}^{j}(\mathbf{v}) + (n+1).A^{j}.\overline{f}^{k}(0)
\end{aligned}$$

Remarquons alors que la formule $\mathbf{v}' = \overline{f}^{i}(\mathbf{v}) \wedge i \geq 0$, dans les variables libres $\mathbf{v}', i, \mathbf{v}$, est définissable dans FO $(\mathbb{R}, \mathbb{Z}, +, \leq)$:

$$\begin{bmatrix} \mathbf{v}' = \overline{f}^{i}(\mathbf{v}) \\ \wedge i \geq 0 \end{bmatrix} \equiv \left(\bigvee_{0 \leq p < j} \begin{bmatrix} \mathbf{v}' = \overline{f}^{i}(\mathbf{v}) \\ \wedge i = p \end{bmatrix} \right) \vee \left(\bigvee_{0 \leq r < k} \exists q \geq 0 \begin{bmatrix} \mathbf{v}' = \overline{f}^{i}(\mathbf{v}) \\ \wedge i = j + q.k + r \end{bmatrix} \right)$$

$$\equiv \left(\bigvee_{0 \leq p < j} \begin{bmatrix} \mathbf{v}' = \overline{f}^{i}(\mathbf{v}) \\ \wedge i = p \end{bmatrix} \right) \vee \left(\bigvee_{0 \leq r < k} \exists q \geq 0 \left[\mathbf{v}' = \overline{f}^{r}(\overline{f}^{j}(\mathbf{v}) + q.A^{j}.\overline{f}^{k}(0)) \right] \right)$$

Remarquons enfin que $\mathbf{v}' \in f^*(\mathbf{v})$ est définissable dans FO $(\mathbb{R}, \mathbb{Z}, +, \leq)$:

$$\mathbf{v}' \in f^*(\mathbf{v}) \equiv \exists s \in \mathbb{N}\ \mathbf{v}' = \overline{f}^{s}(\mathbf{v}) \wedge \forall i \in \mathbb{N} \Big(i < s \implies \overline{f}^{i}(\mathbf{v}) \models \psi \Big)$$

□

On vient de montrer que l'on peut accélérer un circuit dans un système à compteurs mixte. En effet, il suffit d'accélérer la fonction $f = (\psi, A, \mathbf{b})$ égale à la composée des fonctions f_i étiquetant les transitions le long du circuit ; f est donc une fonction mixte-affine. On est alors capable d'accélérer tout système à compteurs mixte *plat*. En effet, dans un système plat, les circuits sont indépendants (i.e. non-imbriqués) : calculer la relation d'accessibilité d'un système à compteurs plat consiste donc uniquement à composer un nombre connu de fonctions mixte-affines, au milieu desquelles interviennent des accélérations de circuits. On vient alors de montrer le théorème suivant :

Théorème 4.14. *Si un système à compteurs mixte est plat, sa relation d'accessibilité est définissable dans FO* $(\mathbb{R}, \mathbb{Z}, +, \leq)$.

On peut donc accélérer les systèmes à compteurs mixtes, de la même façon que les systèmes à compteurs affines. Cependant, en vue d'une implémentation (notamment dans l'outil FAST), il reste à étudier le choix automatique des circuits à accélérer, ainsi que le calcul de la borne[4] à partir de laquelle le monoïde de A se stabilise (dans le cas entier, l'existence de cette borne est décidable en temps exponentiel).

[4]Cette borne est notée s dans la formule à la fin de la preuve du théorème 4.13.

4.6 Comparaison des différents modèles de systèmes à compteurs

Dans ce chapitre, on a étudié divers $\mathscr{L}$-Systèmes à Compteurs se différenciant les uns des autres par la logique $\mathscr{L}$ définissant leurs transitions. En effet, chaque définition de Système à Compteurs est paramétrée par une logique, plus ou moins riche. Bien que presque tous les Systèmes à Compteurs que l'on a étudiés soient généralement indécidables, on peut quand même les comparer en fonction de l'expressivité de leurs logiques $\mathscr{L}$.

Le diagramme 4.3 (page suivante) présente une hiérarchie des systèmes à compteurs étudiés dans ce chapitre.

Une flèche "$A \rightarrow B$" signifie que la logique qui paramètre B est plus riche que la logique qui paramètre A, et un symbole $\bowtie$ entre deux modèles signifie que les logiques les paramétrant sont incomparables. Les systèmes considérés sur ce diagramme ne tiennent pas compte des diverses restrictions que l'on a pu appliquer pour obtenir la décidabilité (plats, reversal-bornés, etc.).

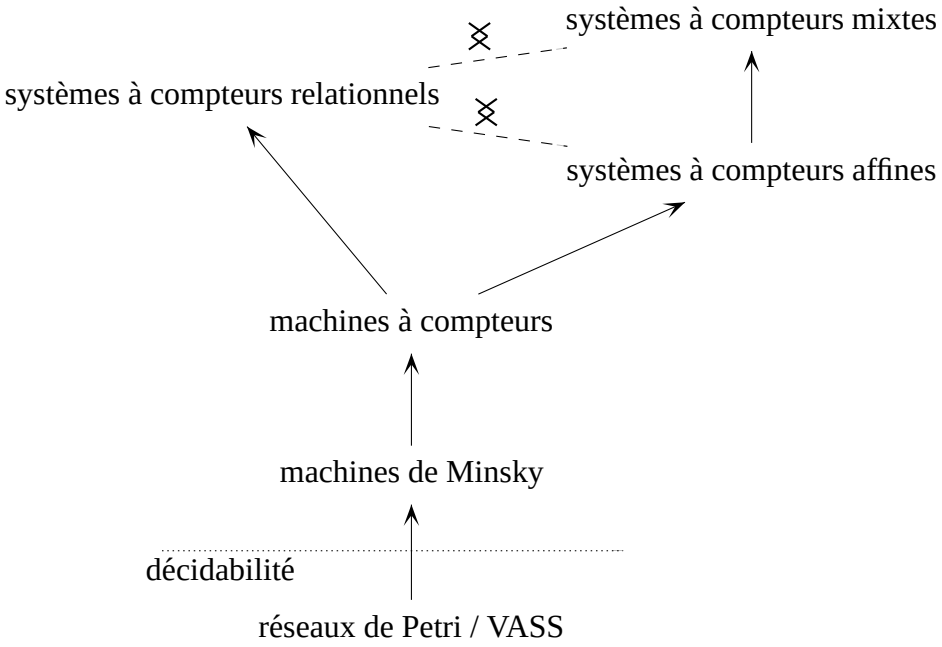


FIG. 4.3: Hiérarchie des systèmes à compteurs étudiés dans ce chapitre

Chapitre 5

Systèmes temporisés

Les systèmes temporisés constituent une grande famille de systèmes, utilisés notamment en vérification. Dans les systèmes à compteurs, les variables servent à compter et à effectuer diverses opérations, selon l'expressivité du modèle ; ils peuvent également simuler une abstraction d'une structure de donnée plus complexe (pile, file, etc.). Les systèmes temporisés, eux, peuvent être de nature assez différente, mais ont en commun une chose : la mesure de l'écoulement du temps, afin de modéliser des contraintes temporelles dans le comportement du système.

Tout comme pour les systèmes à compteurs, la plupart des modèles sont des automates finis munis de variables. Les dispositifs utilisés pour représenter le temps sont variés ; cependant, celui que l'on rencontre le plus souvent est l'*horloge*, sur lequel on se basera fréquemment dans cette thèse. Une horloge est une variable réelle (parfois entière ou rationnelle) qui a la particularité d'augmenter toute seule quand un système reste dans un état de contrôle sans franchir de transition. Contrairement aux compteurs, les horloges changent donc de valeur de façon implicite, mais ne peuvent pas être modifiées directement par le franchissement d'une transition (sauf pour être réinitialisées).

Notons que dans cette thèse, on considère que le temps s'écoule de façon continue. Il existe cependant de nombreux travaux considérant une modélisation discrète du temps.

La particularité des systèmes temporisés est en effet que les variables temporelles changent de valeur en fonction du temps qui passe lorsque le système reste dans un état de contrôle. Le franchissement d'une transition dans un système temporisé, cependant, a un fonctionnement similaire à la plupart des autres types de systèmes. Il existe de nombreuses variations, portant essentiellement sur la dynamique comportementale des variables et sur la puissance d'expression des formules étiquetant les transitions.

Dans cette thèse, on ne donnera pas de définition générique de systèmes tempo-

risés, du fait de leur hétérogénéité. Bien que les automates hybrides englobent tous les systèmes traités dans cette thèse, leur définition est tellement générale que l'on ne s'en servira pas comme base. On s'intéresse plutôt à diverses variations de sous-classes hybrides, qui font l'objet de ce chapitre, découpé en quatre sections.

La première section rappelle brièvement quelques classes d'automates hybrides, qui constituent la classe la plus générale de systèmes. Le but de cette section est principalement de faire un tour d'horizon des différentes familles d'automates hybrides, sans trop entrer dans le détail.

La deuxième section détaille le cas des automates temporisés, qui sont vraisemblablement la sous-classe d'automates hybrides la plus étudiée. Quelques variations de ce modèle seront également présentées.

La troisième section, elle, décrit différentes extensions temporisées des réseaux de Petri, en utilisant un formalisme différent de celui des automates hybrides. On verra donc des modèles proches des systèmes à compteurs (voir section 4.2), que l'on temporise de différentes façons.

Enfin, la section 5.4 introduit une nouvelle classe de systèmes temporisés combinant compteurs et horloges, publiée dans [BFS09] : les Systèmes à Compteurs Temporisés.

5.1 Automates Hybrides

Introduits au début des années 1990, notamment par Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, et Pei-Hsin Ho, les automates hybrides sont, depuis une vingtaine d'années, l'un des formalismes les plus étudiés en vérification de systèmes. La publication la plus citée à ce sujet est probablement [ACH+95] : elle définit les automates hybrides et quelques sous-classes, en montrant que l'accessibilité est un problème très difficile, même pour des sous-classes très restreintes. La présentation de [Hen96] semble toutefois plus claire et fournit des résultats plus avancés concernant l'analyse d'automates hybrides.

Cette section présente le modèle général des automates hybrides, puis cite quelques-unes des sous-classes intéressantes de ce modèle.

Un *automate hybride* (ou *système hybride*) est un automate fini muni de variables réelles qui suivent une loi d'évolution continue. Le qualificatif "hybride" vient du fait que ce modèle de systèmes permet de décrire un comportement à la fois discret, grâce à la structure de contrôle de l'automate, et continu, grâce à la dynamique des variables denses.

Tout comme dans un système à compteurs, on peut étiqueter les transitions par des formules (en fait, n'importe quel prédicat) sur les variables du système : bien que les variables soient denses, ceci constitue la partie discrète du comportement du système.

On modélise en effet un changement d'état du système par le franchissement d'une transition.

En revanche, lorsque le système se trouve dans un état de contrôle, l'automate hybride modélise l'écoulement du temps et les variables changent de valeur en conséquence. Chaque état de contrôle est muni d'une loi d'évolution des variables en fonction du temps : même dans le cas où le temps est représenté de façon discrète (cas mentionné plus tard), ceci constitue la partie continue du comportement du système. On modélise en effet les fluctuations du système quand il ne change pas d'état, ce qui permet une modélisation bien plus riche et précise.

Formellement, soit X un ensemble de n variables, et soit $\mathbf{v} \in \mathbb{R}^n$ une valuation de ces variables. Comme dans les systèmes à compteurs, X' désigne la valeur des variables X après franchissement d'une transition. On désigne par $\dot{X} = \frac{dX}{d\tau}$ la dérivée première de X par rapport au temps τ.

Définition 5.1. *Un* automate hybride *est un quintuplet* $\langle Q, \mathsf{flow}, \mathsf{inv}, \mathsf{jump}, T \rangle$*, où :*

- Q *est un ensemble fini d'états de contrôle*
- flow *est un prédicat dont les variables libres sont dans* $X \cup \dot{X}$
- inv *est un prédicat dont les variables libres sont dans* X
- jump *est un prédicat dont les variables libres sont dans* $X \cup X'$
- $T \subseteq Q \times \mathsf{jump} \times Q$ *est un ensemble fini de transitions*

Le comportement d'un automate hybride est donné, comme la plupart du temps, par un système de transitions : une configuration est un couple (état de contrôle, valuation des variables), et on relie les configurations entre elles, ce qui forme un graphe de transitions. Cependant, les systèmes hybrides ont deux types de comportement distincts : l'un est dit discret, et représente les transitions de l'automate, tandis que l'autre, dit continu (ou dense), représente l'évolution des variables dans un état de contrôle donné.

En prenant la définition 5.1, on voit que le comportement discret va être donné par les formules jump étiquetant les transitions, alors que le comportement continu va être donné par les formules flow et inv étiquetant les états de contrôle.

Intuitivement :

– flow est une formule sur $X \cup \dot{X}$ telle que pour tout $q \in Q$, $\mathsf{flow}(q)$ doit être satifait en tout temps par la valuation $\mathbf{v}$ et sa dérivée : c'est l'équation différentielle qui définit l'évolution des variables quand le système reste dans un état de contrôle.

– inv est une formule sur X telle que pour tout $q \in Q$, $\mathsf{inv}(q)$ doit être satifait en tout temps par la valuation $\mathbf{v}$: c'est l'invariant d'un état de contrôle.

– jump est une formule sur $X \cup X'$ telle que pour tout $q, q' \in Q$, $\mathsf{jump}(q, q')$ doit être satisfait par les valuations $\mathbf{v}$ et $\mathbf{v}'$: c'est l'étiquette de la transition, qui définit la formule de garde et de mise à jour des variables.

Comme d'habitude, une transition change l'état de contrôle courant, mais doit aussi satisfaire une formule, appelée ici jump. Intuitivement, jump désigne une formule combinant un test des valuations de variables (i.e. une *garde*) ainsi que leur mise à jour, comme dans les systèmes à compteurs. Cette définition serait d'ailleurs syntaxiquement équivalente à la définition générique 4.1 si l'on ignorait flow et inv.

De façon similaire, on a une condition et une mise à jour dans chaque état de contrôle. En effet, le comportement continu consiste à rester dans un état de contrôle, plutôt que de franchir une transition ; la condition inv est donc une condition pour *rester* dans l'état de contrôle, et on l'appelle un *invariant*. La mise à jour des valuations de variables flow, elle, se fait de façon continue, en fonction du temps qui passe (par opposition à une mise à jour discrète, effectuée une seule fois à chaque franchissement de transition).

On a donc pour chaque comportement (discret ou continu) une condition, appelée "garde" ou "invariant", et une mise à jour des variables. Le fait qu'on utilise deux prédicats pour le comportement continu et un seul pour le comportement discret peut sembler étrange. En effet, on pourrait vouloir regrouper inv et flow dans un même prédicat, comme pour jump : les nouvelles valeurs sont notées $\dot{X}$ au lieu de X', et sont déterminées en fonction de X, si X vérifie les conditions requises. Cependant, on suivra ici la tradition, qui distingue les deux types de formules dans le cas temporisé, mais pas nécesairement dans le cas discret.

De plus, puisque flow correspond à la dérivée $\dot{X}$, sa valeur dépend également du temps, ce qui pourrait justifier que ce prédicat soit défini séparément. Le temps, quant à lui, sera représenté de façon absolue par une valeur $\tau \in \mathbb{R}_+$.

Il existe plusieurs façons de définir la sémantique d'un automate hybride, selon le niveau d'abstraction considéré. En effet, pour représenter le comportement d'un automate hybride par un système de transitions, on est obligé de discrétiser les lois d'évolution continues des variables. Pour ce faire, on abstrait le caractère continu, en le rendant discret ; selon l'abstraction utilisée, le système de transition obtenu peut être fini ou infini. Dans notre cas, on ne s'intéressera qu'à la sémantique dite "temporisée", qui génère un espace de configurations infini. Il existe aussi notamment une sémantique "à temps abstrait", que l'on développera plutôt dans le cas des automates temporisés (sections 5.2 et 5.4).

Formellement, soit predic un prédicat dont les variables libres sont $X = \{x_1, \ldots, x_n\}$. Une instanciation de predic sera notée $\mathsf{predic}[x_1 \leftarrow v_1, x_2 \leftarrow v_2, \ldots, x_n \leftarrow v_n]$, signi-

fiant que les valeurs $\mathbf{v} = v_1, v_2, \ldots, v_n$ sont affectées aux variables libres X (qui deviennent alors liées).

Définition 5.2. *La sémantique d'un automate hybride* $\mathcal{S} = \langle Q, \mathsf{flow}, \mathsf{inv}, \mathsf{jump}, T\rangle$ *est donnée par un* système de transitions $ST(\mathcal{S}) = \langle C, \rightarrow\rangle$ *tel que :*

- $C = Q \times \mathbb{R}^n$ *est l'ensemble des configurations, défini par* $(q, \mathbf{v}) \in C$ *si et seulement si* $\mathsf{inv}(q)[X \leftarrow \mathbf{v}]$ *est vrai*
- $\rightarrow \subseteq C \times (T \cup \mathbb{R}_+) \times C$ *est une relation définie par deux cas appelés "discret" et "continu" :*
  - *"discret" :* $(q, \mathbf{v}) \xrightarrow{t} (q', \mathbf{v}')$ *si et seulement s'il existe* $t = (q, \mathsf{jump}, q') \in T$ *telle que* $\mathsf{jump}(q, q')[X \leftarrow \mathbf{v}, X' \leftarrow \mathbf{v}']$ *est vrai*
  - *"continu" :* $(q, \mathbf{v}) \xrightarrow{\tau} (q', \mathbf{v}')$ *si et seulement si* $q' = q$ *et il existe* $\tau \in \mathbb{R}_+$ *et une fonction différentiable* $f : [0, \tau] \longrightarrow \mathbb{R}^n$ *de dérivée première* $\dot{f} :]0, \tau[\longrightarrow \mathbb{R}^n$ *tels que (1)* $f(0) = X$*, (2)* $f(\tau) = X'$*, et (3) pour tout réel* $\xi \in]0, \tau[$ *, les prédicats* $\mathsf{inv}(q)[X \leftarrow f(\xi)]$ *et* $\mathsf{flow}(q)[X \leftarrow f(\xi), \dot{X} \leftarrow \dot{f}(\xi)]$ *sont vrais*

Une exécution d'un automate hybride s'appelle une *trajectoire*, et se définit comme une suite de configurations $(c_i)_{i\geq 0}$ telles que pour tout $i \geq 0$, soit il existe $t \in T$ telle que $c_i \xrightarrow{t} c_{i+1}$, soit il existe $\tau \in \mathbb{R}_+$ tel que $c_i \xrightarrow{\tau} c_{i+1}$.

Il est clair que le problème d'accessibilité est indécidable dans ce modèle, ne serait-ce qu'à cause du fait que les prédicats flow, inv, et jump peuvent être définis dans n'importe quelle logique. L'espace des configurations est naturellement infini, puisque l'on peut observer le système à tout temps τ, et ce, aussi souvent que désiré. C'est pourquoi il existe de nombreuses sous-classes de systèmes hybrides, cherchant à restreindre les prédicats autorisés et donc les comportements possibles, afin de se rapprocher de la décidabilité.

Automates hybrides linéaires (LHA)

Les automates hybrides linéaires (Linear Hybrid Automata, LHA) [ACH+95, Hen96] sont, comme leur nom l'indique, une classe de systèmes hybrides dans laquelle les prédicats sont restreints à des contraites linéaires. Leur particularité est que la loi d'évolution des variables dans un état de contrôle ne dépend que du temps, et pas de la valeur courante des variables. De plus, on définit jump comme deux prédicats disjoints : guard, qui donne la condition de franchissement d'une transition, et assign, qui donne la mise à jour des valeurs après franchissement de la transition.

Plus formellement, un automate hybride est dit linéaire lorsqu'on définit ses prédicats comme suit. Les matrices P et les vecteurs $\mathbf{u}$ sont à coefficients dans $\mathbb{Z}$.

- $\mathsf{jump} = \mathsf{guard} \cup \mathsf{assign}$
- guard et inv sont de la forme $P.X \leq \mathbf{u}$
- assign est de la forme $X' = P.X + \mathbf{u}$
- flow est de la forme $P.\dot{X} \leq \mathbf{u}$

Le problème d'accessibilité le plus simple (accessibilité d'un état de contrôle) reste indécidable sur ce modèle. Cependant, il existe un cadre pour l'analyse de ces systèmes hybrides linéaires, qui utilise une technique d'accelération récemment développée dans [BH06]. Le fonctionnement de cette technique est basé sur le fait que les prédicats génèrent des ensembles convexes, du fait de leur linéarité. D'autres propriétés des automates hybrides linéaires sont églement exploitées par cette technique, comme le fait que les compositions de transitions étiquetant des circuits de l'automate sont définissables dans la logique décidable FO $(\mathbb{R}, \mathbb{Z}, +, <)$. Un prototype implémentant cette accélération de systèmes hybrides linéaires est en cours de développement, sous le nom de HAT (Hybrid Acceleration Toolkit)[1].

Il existe d'autres classes intéressantes d'automates hybrides linéaires, comme les automates rectangulaires [Hen96]. La classe la plus célèbre est probablement celle des automates temporisés, à laquelle la section suivante est dédiée.

Notons enfin que l'outil HYTECH [HHWT97] est dédié à l'analyse des systèmes hybrides par model-checking.

5.2 Automates Temporisés

Définis à l'aube des années 1990, les automates temporisés constituent certainement la classe de systèmes hybrides linéaires la plus étudiée et utilisée à ce jour. Rajeev Alur et David L. Dill ont publié les premières bases des automates temporisés dans [AD90], ainsi qu'une abstraction permettant de les analyser. Ils ont complété ces résultats dans [AD94], tout en les présentant plus clairement.

Un automate temporisé est un automate hybride linéaire dans lequel les variables, appelées *horloges*, évoluent toutes à une même vitesse constante. De plus, les mises à jour des variables sont limitées à des remises à zéro, et les gardes et invariants sont réduits à des conjonctions de bornes sur la valeur d'une variable (ou sur la différence de

[1] La seule référence disponible à ce jour concernant HAT est la présentation de Frédéric Herbreteau au workshop *Automata and Verification'08*, disponible à l'adresse suivante : `ftp://ftp.umh.ac.be/pub/ftp_info/info_th/av08/Herbreteau.pdf`

deux variables, comme on le verra un peu plus loin).

Formellement, on définit les variables, dorénavant appelées *horloges*, comme dans les automates hybrides (voir page 101). Pour tout $x \in X$, $x \bowtie k$ est une *contrainte atomique*, avec $\bowtie \in \{<, \leq, =, \geq, >\}$ et $k \in \mathbb{N}$. Soit $\mathcal{C}(X)$ l'ensemble des *contraintes d'horloges*, c'est-à-dire des combinaisons booléennes de contraintes atomiques. On dit qu'une valuation d'horloges satisfait une contrainte d'horloges $\psi \in \mathcal{C}(X)$ si $\psi[X \leftarrow \mathbf{v}]$ est vraie ; dans ce cas, on écrit $\mathbf{v} \models \psi$.

Définition 5.3. *Un* automate temporisé *est un automate hybride linéaire* $\langle Q, \mathsf{flow}, \mathsf{inv}, \mathsf{guard}, \mathsf{assign}, T\rangle$ *dans lequel :*

- Q *est un ensemble fini d'états de contrôle*
- flow *est tel que* $\dot{x} = 1$*, pour tout* $x \in X$
- inv *et* guard *sont des contraintes d'horloges de* $\mathcal{C}(X)$
- assign *est réduit à des opérations de la forme* $x' = 0$*, avec* $x' \in X$
- $T \subseteq Q \times \mathsf{guard} \times \mathsf{assign} \times Q$ *est un ensemble fini de transitions*

La sémantique d'un automate temporisé est donnée par un système de transitions similaire à celui d'un automate hybride. Chaque configuration existe à condition qu'elle respecte l'invariant de l'état de contrôle concerné, et les transitions sont soit discrètes, soit continues. Les transitions discrètes peuvent simplement remettre des horloges à zéro (i.e. faire un *reset*), à condition que leur garde soit vérifiée. Les transitions continues, réduites à des *délais*, ne peuvent que laisser passer le temps de façon uniforme, et ce pour toutes les horloges à la fois. Ces délais, ou "temporisations", sont à l'origine du nom "automates temporisés".

Formellement :

Définition 5.4. *La sémantique d'un automate temporisé* $\mathcal{S} = \langle Q, \mathsf{flow}, \mathsf{inv}, \mathsf{guard}, \mathsf{assign}, T\rangle$ *est donnée par un* système de transitions $ST(\mathcal{S}) = \langle C, \rightarrow\rangle$ *tel que :*

- $C = Q \times \mathbb{R}^n$ *est l'ensemble des configurations, défini par* $(q, \mathbf{v}) \in C$ *si et seulement si* $\mathsf{inv}(q)[X \leftarrow \mathbf{v}]$ *est vrai*
- $\rightarrow \subseteq C \times (T \cup \mathbb{R}_+) \times C$ *est une relation définie par deux cas appelés "discret" et "continu" :*
  - *"discret" :* $(q, \mathbf{v}) \xrightarrow{t} (q', \mathbf{v}')$ *si et seulement s'il existe* $t = (q, \mathsf{guard}, \mathsf{assign}, q') \in T$ *telle que* $\mathsf{guard}(q, q')[X \leftarrow \mathbf{v}, X' \leftarrow \mathbf{v}']$ *et* $\mathsf{assign}(q, q')[X \leftarrow \mathbf{v}, X' \leftarrow \mathbf{v}']$ *sont vrais*
  - *"continu" :* $(q, \mathbf{v}) \xrightarrow{\tau} (q', \mathbf{v}')$ *si et seulement si* $q' = q$ *et il existe un délai* $\tau \in \mathbb{R}_+$ *tel que* $\mathbf{v}' = \mathbf{v} + \tau$

Une exécution d'un automate temporisé est tout à fait identique à celle d'un automate hybride (voir page 103).

Notons que le comportement est extrêmement simplifié par rapport à un automate hybride, même linéaire. De plus, une transition continue n'a pas besoin de vérifier que l'invariant est satisfait tout au long du délai. En effet, si la transition $(q, \mathbf{v}) \xrightarrow{\tau} (q', \mathbf{v}')$ existe, c'est que les deux configurations $(q, \mathbf{v})$ et $(q, \mathbf{v} + \tau)$ existent, et donc que celles-ci respectent l'invariant $\mathsf{inv}(q)$; et puisque les invariants sont simplement des bornes constantes sur la valeur des horloges, $\mathsf{inv}(q)$ sera forcément satisfait sur l'intervalle $[\mathbf{v}, \mathbf{v} + \tau]$. Notons également que la notation $\mathbf{v} + \tau$ capture totalement le prédicat flow, qui exprime simplement la contrainte que la dérivée de chaque horloge est égale à 1, et donc que lorsqu'on laisse τ unités de temps passer dans un état de contrôle, toutes les variables sont augmentées exactement de τ.

On pourrait présenter le comportement de manière encore plus simple, mais le formalisme s'éloignerait de celui des automates hybrides. Un tel formalisme plus simple (et aussi plus répandu) sera utilisé dans la section 5.4, où l'on combinera automates temporisés et systèmes à compteurs dans un même modèle.

Régions. L'espace des configurations d'un automate temporisé reste potentiellement infini, en général. Cependant, du fait de la simplicité des gardes et des invariants, une abstraction simple est utilisée pour rendre l'espace des configurations fini : regrouper les configurations en *régions*, qui sont des classes d'équivalences. La notion d'équivalence utilisée ici est définie par rapport aux contraintes d'horloges : dans une même région, toutes les configurations satisfont exactement les mêmes gardes et invariants. Dans un automate temporisé donné, puisqu'on connaît a priori toutes les contraintes d'horloges utilisées, on peut fixer la constante maximale à laquelle les horloges seront comparées, et donc rendre le nombre de régions fini. Les détails formels de la construction des régions seront donnés dans le cadre des systèmes à compteurs temporisés, dans la section 5.4.

Analyse. En utilisant cette abstraction par régions, on est en mesure de résoudre le problème d'accessibilité d'un état de contrôle, et donc d'analyser le système. Cette analyse peut se faire de deux façons : en avant ou en arrière. Le problème est le suivant : à partir d'une configuration initiale, on cherche à déterminer si un ensemble de configurations donné est accessible. Si l'on calcule l'ensemble d'accessibilité "en avant", on arrive donc potentiellement à une infinité de configurations, puisque le temps peut diverger à l'infini. Cependant, le calcul "en arrière", en remontant les transitions ayant pu mener aux configurations d'arrivée, donne toujours un ensemble fini ; par contre, il calcule également des configurations qui ne sont pas accessibles depuis la configuration initiale. C'est pourquoi, à l'aide d'une abstraction de l'espace des configurations,

on préfère utiliser l'analyse "en avant", qui termine alors toujours grâce à la finitude de l'abstraction.

Contraintes diagonales. Il existe de nombreuses variantes dans la définition des automates temporisés, et elles ne sont pas toutes équivalentes. La variante la plus célèbre est celle des *contraintes diagonales* : dans une contrainte d'horloge, on peut autoriser de borner la différence de deux valuations d'horloges, en plus de simplement borner des valuations d'horloges. Cependant, Patricia Bouyer a montré, notamment dans [Bou04] et [BBFL03], que l'analyse en avant n'est pas correcte pour certaines contraintes diagonales, car certaines configurations sont atteintes dans l'analyse alors qu'elles ne devraient pas l'être. En outre, on sait grâce à [BDGP98] que les contraintes diagonales peuvent toujours être transformées en contraintes d'horloges classiques ; mais l'encodage génère une croissance de la taille du modèle qui est exponentielle dans le nombre de contraintes diagonales, ce qui n'est pas raisonnablement utilisable en pratique. C'est pourquoi des alternatives sont donnés dans [BLR05], afin de pouvoir analyser efficacement des automates temporisés avec contraintes diagonales.

Définitions alternatives des automates temporisés. D'autres variantes dans la définition des automates temporisés sont souvent recontrées dans la littérature. Par exemple, on interdit parfois les invariants dans les états de contrôle (i.e. $\mathsf{inv}(q)$ est toujours vrai, pour tout $q \in Q$), simplement parce qu'ils peuvent être transformés en gardes sur des transitions, et que leur présence ne change pas les résultats d'accessibilité. On autorise parfois des mises à jour beaucoup plus riches, notamment avec un ensemble de valeurs possibles (ce qui est une source de non-déterminisme). Une autre question importante, en termes de langage d'un automate temporisé, est la présence ou non de transitions silencieuses (également appelées ε-transitions), c'est-à-dire d'autoriser ou non des transitions qui ne soient pas étiquetées par un symbole. La plupart de ces variantes ont été étudiées en détail dans la thèse de Patricia Bouyer [Bou02] : un tableau récapitulatif y est présenté à la page 85.

Aplatissabilité et accélérabilité. Dans [CJ99], Hubert Comon-Lundh et Yan Jurski ont montré que tout automate temporisé est aplatissable. Comme mentionné dans les sections 4.3 et 4.4, un système est plat s'il ne contient pas de boucles imbriquées. Plus précisément, la définition 4.7 est donnée pour les systèmes à compteurs, mais peut être appliquée à l'identique sur les automates temporisés. Ici, un système est dit aplatissable s'il existe un système plat ayant le même relation d'accessibilité (à un mappage des états de contrôle près). Le fait que tout automate temporisé soit aplatissable est très intéressant ; en effet, les mêmes auteurs ont montré dans [CJ98] que la relation d'accessibilité de tout système à compteurs relationnel plat est définissable dans la logique

décidable FO $(\mathbb{R}, \mathbb{Z}, +, <)$. Or en utilisant un encodage proposé dans [Fri98], Comon et Jurski montrent que l'on peut traduire un automate temporisé en un système à compteurs relationnel plat.

Ainsi, on peut définir la relation d'accessibilité d'un automate temporisé dans une logique décidable, ce qui signifie que l'on peut utiliser le concept de meta-transitions afin d'accélérer le calcul de l'ensemble d'accessibilité d'un automate temporisé. On rappelle qu'une meta-transition est une composée de transitions, utilisée notamment pour le calcul de la fermeture transitive d'un circuit dans un automate. Mentionné dans [CJ99], ce résultat est retrouvé dans [BH06] dans le cadre de l'accélération de systèmes linéaires hybrides. Cependant, il ne semble pas encore exister de technique d'accélération spécifiquement dédiée aux automates temporisés. En effet, ces automates bénéficient d'une algorithmique très efficace, du fait de la simplicité de leurs gardes, invariants, et actions ; il serait donc dommage de ne pas en profiter pour adapter les techniques d'accélération hybride à cette sous-classe très réduite mais néanmoins intéressante.

5.3 Quelques autres modèles temporisés

5.3.1 Variantes des automates temporisés

Forts de leur succès dans le monde de la recherche théorique et des applications industrielles, les automates temporisés sont fréquemment déclinés dans des versions assez proches, plus ou moins expressives. On donne ici une description succinte de quatre de ces modèles de systèmes.

Event-clock automata [AFH99]

Un des problèmes des automates temporisés est qu'ils ne sont pas clos par complémentation. En effet, en terme de langages (ou de traces), le complémentaire d'un automate temporisé ne donne pas forcément un automate temporisé. En fait, on ne sait pas déterminiser un automate temporisé, de façon générale. Ceci pose problème notamment parce que la procédure de complémentation d'un automate intervient dans sa vérification par model-checking, et fait appel à sa déterminisation [HU79, SVW87]. C'est pourquoi la classe des event-clock automata a été introduite : c'est une extension déterminisable des automates temporisés, qui est également close par complémentation (ainsi que par les autres opérations booléennes). Leur particularité est qu'ils peuvent prédire la prochaine occurrence, et se souvenir de la dernière occurrence, d'un symbole étiquetant une transition.

Integer-reset timed automata [SPKM08]

Une autre classe intéressante est celle des integer-reset timed automata (IRTA), dans laquelle la remise à zéro d'une horloge ne peut s'effectuer que lorsque sa valuation est entière. Cette restriction des automates temporisés revient en quelque sorte à discrétiser, en partie, le temps. C'est une sous-classe intéressante parce que tout comme les event-clock automata, elle est déterminisable et close par complémentation. Les auteurs motivent également la définition d'une telle sous-classe pour des applications dans le domaine des services web, dans lequel cette semi-discrétisation du temps semble plus réaliste.

Protocol timed automata [PBCT07]

Les protocol timed automata sont une autre sous-classe des automates temporisés, dont les propriétés et les applications sont similaires à celles des IRTA. En effet, ils ont été définis en réponse à des problématiques de vérification de protocoles pour les services web ; et tout comme les IRTA, ils sont clos par complémentation. Un protocol timed automaton est un automate temporisé respectant les quatre conditions suivantes : (1) chaque horloge est remise à zéro sur une seule transition et chaque transition remet à zéro une seule horloge, (2) à chaque instant, et pour chaque symbole d'étiquetage donné, il n'y a qu'une seule transition partant de l'état de contrôle courant et dont la garde est vérifiée (i.e. l'automate est déterministe), (3) chaque ε-transition (i.e. sans symbole d'étiquetage) comporte au moins une contrainte d'égalité dans sa garde, et (4) les ε-transitions ne sont pas prioritaires sur les autres transitions (voir le théorème 1 de [PBCT07] pour les détails techniques).

Interrupt timed automata [BH09]

Récemment, une nouvelle classe de systèmes hybrides, proche des automates temporisés, a été introduite pour modéliser le mécanisme d'interruptions dans un environnement mono-processeur. Cette classe, appelée interrupt timed automata, est en fait incomparable aux automates temporisés (en termes de langages). Outre son intérêt pour la modélisation, cette classe est décidable vis-à-vis du problème d'accessibilité. Par contre, elle n'est stable ni par complémentation, ni par intersection. Dans [BH09], les auteurs montrent qu'en combinant les interrupt timed automata avec les controlled timed automata (une extension des automates temporisés, définie dans [DZ98]), le problème d'accessibilité reste décidable, et ce modèle combiné est donc une extension des automates temporisés.

5.3.2 Extensions temporisées des réseaux de Petri

Les réseaux de Petri (présentés dans la section 4.2) ont été étendus de nombreuses façons depuis plusieurs décennies. On rappelle que les réseaux de Petri sont équivalents aux VASS, qui sont un cas particulier de systèmes à compteurs. Cette section décrit rapidement quelques-unes de ces extensions, combinant la capacité de comptage des réseaux de Petri à un aspect temporisé. Notons que pour la quasi-totalité de ces extensions, le problème d'accessibilité est indécidable.

Réseaux de Petri temporels [Mer74]

Une des extensions temporisées les plus classiques est celle des Time Petri Nets (TPN) [Mer74], que l'on traduit par "réseaux de Petri temporels". La notion de temps est ici représentée par un intervalle sur chaque transition, auquel l'âge d'une transition doit appartenir pour que l'on puisse franchir cette transition. L'âge d'une transition est le temps écoulé depuis qu'elle est "activée" en termes de jetons, c'est-à-dire depuis que les places précédant cette transition comportent suffisamment de jetons pour la franchir (i.e. depuis que $M \geq W^-(t)$). Une configuration d'un réseau de Petri temporel est donc un couple (M, ν), où M est le marquage du réseau et ν donne l'âge des transitions. Comme dans tout système hybride, on peut soit laisser passer du temps, soit franchir une transition.

On trouve également des variantes, dans lesquelles le temps n'est plus sur les transitions, mais sur les arcs, ou encore dans les places (voir [BR08]).

Il existe plusieurs sémantiques pour les réseaux de Petri temporels : les deux principaux facteurs les différenciant sont la politique de mémorisation et l'urgence.

Certaines sémantiques remettent systématiquement à zéro l'âge d'une transition après qu'elle ait été franchie, mais d'autres non. On distingue trois cas de politique de mémorisation, qui sont étudiés dans [BCH$^+$05] et [RS09].

Certaines sémantiques imposent l'urgence, c'est-à-dire que l'âge d'une transition ne peut pas dépasser la borne supérieure de son intervalle. En d'autres termes, l'urgence signifie qu'une transition ne peut pas être désactivée par l'écoulement du temps. La sémantique avec urgence est appelée sémantique forte, et celle sans urgence est appelée sémantique faible. Pour des précisions sur ces variations et leur étude comparative, voir [BR08].

Enfin, pour des précisions et quelques résultats sur les réseaux de Petri temporels, voir la section 2.2 de la thèse de Pierre-Alain Reynier [Rey07].

Réseaux de Petri temporisés [BLT90]

Une autre extension, proche de la précédente et tout aussi répandue, est celle des Timed Petri Nets (TdPN) [BLT90], appelés "réseaux de Petri temporisés " en français.

Dans ce modèle, ce sont les jetons qui ont un âge ; les contraintes sur cet âge sont alors des intervalles associés aux applications d'incidence avant W^+ et arrière W^-. Ainsi, un jeton ne peut servir à franchir une transition t que si son âge appartient à l'intervalle associé à $W^-(t)$. De plus, l'âge du jeton est mis à jour en choisissant une valeur de façon non-déterministe dans l'intervalle associé à W^+. Comme dans tout système hybride, on peut soit laisser passer du temps, soit franchir une transition.

Dans ce modèle, il n'y a généralement pas de notion d'urgence ni de politique de mémorisation. Cependant, on trouve quand même quelques variantes, et la recherche est encore très active quant à l'étude de problèmes de vérification des réseaux de Petri temporisés. Parmi les principaux résultats, on peut citer [BHR08, AMM07, dFEVRMA00, VRCGdFE99].

Là encore, pour des précisions et quelques résultats sur les réseaux de Petri temporisés, voir la section 2.3 de la thèse de Pierre-Alain Reynier [Rey07].

Réseaux de Petri hybrides et différentiels temporisés

Il existe de nombreuses autres extensions des réseaux de Petri, le plus souvent classées dans ce qu'on appelle les réseaux de Petri hybrides. Le qualificatif "hybride" a ici la même signification que pour les automates hybrides, dans le sens où le système comporte à la fois une composante discrète et une composante continue. Cependant, les définitions ne sont pas les mêmes. En effet, "discret" se réfère ici aux réseaux de Petri classiques, alors que "continu" se réfère aux réseaux de Petri continus qui sont un modèle à part entière. Les réseaux de Petri hybrides sont une combinaison de réseaux de Petri classiques (i.e. discrets) et continus : certaines places et transitions sont discrètes, et les autres sont continues.

Un réseau de Petri continu est composé uniquement de places continues et de transitions continues. Une place (respectivement, une transition) continue, ici, est une place (respectivement, une transition) qui ne manipule que des fractions de jetons, et non pas des jetons entiers. Ce modèle de réseaux de Petri continus a été introduit par René David et Hassane Alla dans [DA87] ; ces mêmes auteurs ont également publié un livre sur les réseaux de Petri hybrides et continus [DA05]. Dans cet ouvrage, on trouve une description d'une version temporisée des réseaux de Petri hybrides et continus, ainsi que de nombreuses références à ce sujet.

Un autre modèle est celui des réseaux de Petri différentiels temporisés [DK96], qui correspond à celui des automates hybrides dans lequel il n'y a pas de transitions discrètes. Un réseau de Petri différentiel temporisé est constitué de transitions différentielles et de places différentielles (dont le marquage peut être négatif). Chaque arc est étiqueté par un flux de données et une vitesse de franchissement ; le flux de données

est similaire à la quantité fractionnaire de jetons à manipuler, et couplé à la vitesse de franchissement, régit la dynamique du système au moyen d'une équation différentielle.

Ce modèle, bien que déterministe, a la puissance des machines de Turing [HRS06] ; ceci montre que les réseaux de Petri différentiels temporisés sont légèrement plus expressifs que les équations différentielles, puisque ces équations définissent une théorie décidable [BCGH07].

TVASS : VASS temporisés [GS94]

Un autre modèle, que l'on peut également classer dans la famille des extensions temporisées des réseaux de Petri, est celui des VASS temporisés, ou Timed VASS (TVASS). Ce modèle a été introduit dans [GS94] sous le nom de Timed P/T Nets, puis repris dans [Bér95] et [BFS09]. Il consiste en fait à combiner les automates temporisés avec les VASS : on a donc des compteurs entiers, que l'on peut augmenter et dont on peut vérifier s'ils sont plus grands qu'une constante donnée, et de façon disjointe, des horloges réelles, qui augmentent constamment et uniformément et que l'on peut tester et remettre à zéro comme dans les automates temporisés.

Bien qu'il ait été très peu étudié, ce modèle est intéressant puisqu'il combine deux des modèles les plus connus : réseaux de Petri et automates temporisés. Ces deux modèles sont décidables et même vérifiables par model-checking. Leur combinaison garde ces propriétés, comme on le verra dans la section 5.4.3. C'est en fait un cas particulier de Systèmes à Compteurs Temporisés, que l'on définit et étudie dans la section 5.4.

5.3.3 Comparaison des systèmes temporisés

Le diagramme 5.1 (page suivante) présente une hiérarchie partielle des systèmes temporisés mentionnés dans ce chapitre. Il n'a pas la prétention d'être exhaustif : une absence de chemin entre deux modèles signifie simplement que nous n'avons pas connaissance de l'expressivité comparée de ces deux modèles. En particulier, le fait que deux modèles soient à la même hauteur n'indique pas nécessairement qu'ils sont incomparables ou d'un pouvoir d'expression similaire. En revanche, une flèche "$A \to B$" signifie que le modèle B peut simuler le modèle A, et un symbole $\bowtie$ entre deux modèles indique qu'ils sont incomparables. De plus, le problème d'accessibilité est indécidable pour les modèles situés au-dessus de la ligne pointillée, et est décidable pour ceux situés en-dessous.

Le sigle “TA” (Timed Automata) désigne les automates temporisés, et “PN” (Petri Nets) désigne les réseaux de Petri.

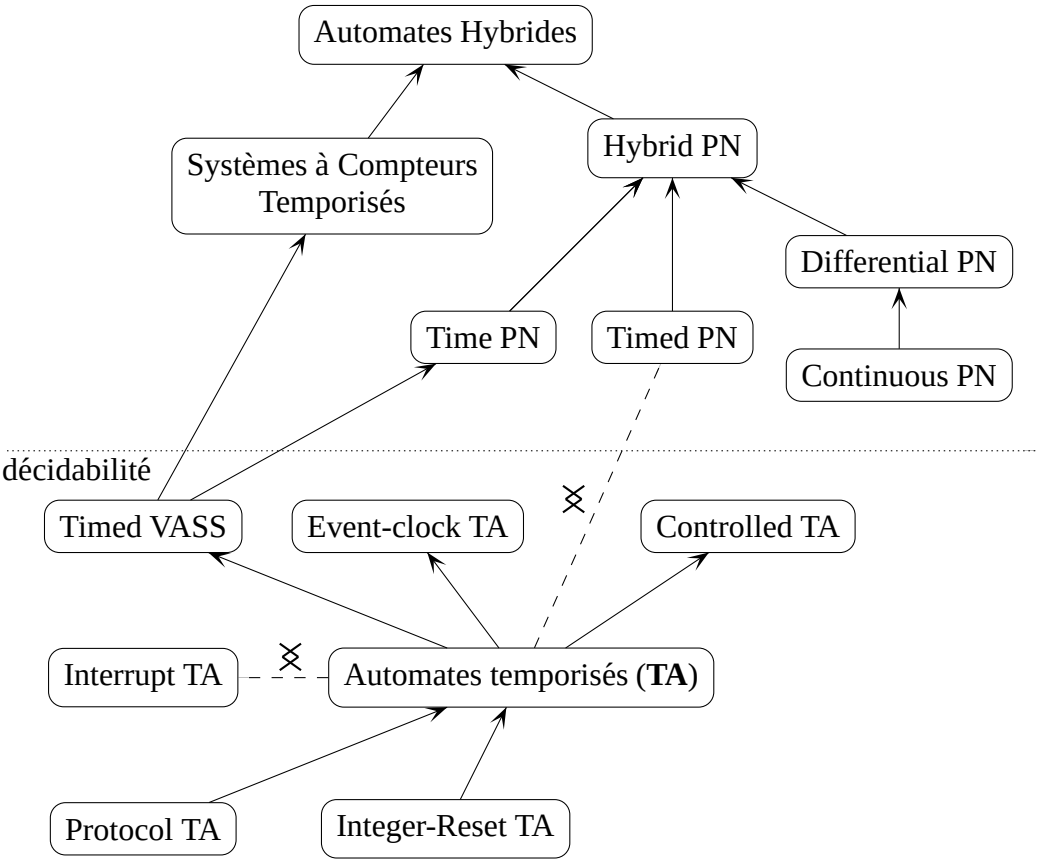


FIG. 5.1: Hiérarchie partielle des systèmes temporisés étudiés dans ce chapitre

Notons que ces comparaisons sont très délicates à présenter de façon aussi simplifiée, du fait de l’importance que peut avoir la moindre variation dans la définition du modèle ou de la relation de comparaison. Par exemple, les réseaux de Petri temporisés (TdPN) sont incomparables avec les automates temporisés (TA) dans le cas général, mais deviennent moins expressifs que les automates temporisés (en termes de langages) dès que l’on borne le nombre de jeton maximum [BHR08].

5.4 Systèmes à Compteurs Temporisés

Dans cette section, on introduit les Sytèmes à Compteurs Temporisés, appelés TCS (pour Timed Counter Systems). Ils ont été introduits dans [BFS09], et sont une nou-

velle classe de systèmes combinant compteurs et horloges. Naturellement, l'ensemble des configurations d'un tel système est infini, et leurs problèmes d'accessibilité sont en général indécidables. En utilisant la fameuse technique d'abstraction des horloges en régions, introduite pour les automates temporisés, on s'intéresse au problème d'accessibilité d'une configuration partielle, dans lequel on ne cherche pas à connaître la valeur des horloges. On montre que ce problème est décidable pour trois sous-classes : les TVASS, les TCS bornés, et les TCS reversal-bornés. Tous ces résultats ont été publiés dans [BFS09].

5.4.1 Définitions

Afin d'utiliser un modèle homogène pour des systèmes combinant compteurs et horloges, on définit précisément, dans les deux paragraphes suivants, la façon dont on manipule ces deux types de données. Ces définitions sont prioritaires sur celles définies dans les préliminaires (section 1.3), dans le chapitre 4, et au début de ce chapitre.

Horloges. Soit X un ensemble de m variables réelles, appellées *horloges*. Une *valuation d'horloge* sur X est un vecteur $\mathbf{x} \in \mathbb{R}_+^m$. Étant données une valuation d'horloge $\mathbf{x}$ et une durée $\tau \in \mathbb{R}_+$, $\mathbf{x}+\tau$ est la valuation d'horloge définie par $(\mathbf{x}+\tau)_i = \mathbf{x}_i + \tau$ pour tout $i \in [1, m]$.

Soit $R_X = G_X \times \{0,1\}^m$ l'ensemble des opérations sur les horloges, où :

- G_X est l'ensemble des contraintes d'horloges (ou *gardes*) définies par la grammaire suivante $g ::= x_1 - x_2 \bowtie b \mid x \bowtie b \mid g \wedge g \mid \neg g$, avec $\bowtie \in \{<, \leq, =, \geq, >\}$, $x, x_1, x_2 \in X$, et $b \in \mathbb{N}$.
- $\{0,1\}^m$, intuitivement, sélectionne les horloges à remettre à zéro.

Pour une garde $g \in G_X$ et une valuation d'horloge $\mathbf{x} \in \mathbb{R}_+^m$, on écrit $\mathbf{x} \models g$ quand la valuation $\mathbf{x}$ satisfait la garde g. Par convention, quand $X = \emptyset$, alors $R_X = \{\emptyset\}$. Soit $\mathbf{x}, \mathbf{x}' \in \mathbb{R}_+^m$ et $(g, \lambda) \in R_X$. Alors $(\mathbf{x}, \mathbf{x}') \models (g, \lambda)$ est défini par : $\mathbf{x} \models g$ et $\forall i \in [1, m], \lambda_i = 0 \implies \mathbf{x}'_i = 0$ et $\lambda_i = 1 \implies \mathbf{x}'_i = \mathbf{x}_i$ (ou plus simplement, $\mathbf{x}'_i = \lambda_i \mathbf{x}_i$).

Dans la suite, par souci de lisibilité, on suppose que les gardes sur les horloges ne contiennent pas de contraintes diagonales (i.e. de la forme $x - y \bowtie b$), et ce, sans perte de généralité : en effet, [BDGP98] donne une traduction des contraintes diagonales en contraintes non-diagonales (i.e. de la forme $x \bowtie b$).

Compteurs. Soit Y un ensemble de n variables entières, appelées *compteurs*. Une *valuation de compteurs* sur Y est un vecteur $\mathbf{y} \in \mathbb{Z}^n$. Soit $R_Y \subseteq \mathbb{Z}^n \times \mathbb{Z}^n$ l'ensemble des relations définissables par une formule de Presburger. Intuitivement, de telles relations binaires décrivent l'effet des transitions sur les compteurs ; c'est-à-dire, pour $r \in R_Y$, $(\mathbf{y}, \mathbf{y}') \in r$ signifie que la valuation des compteurs est $\mathbf{y}$ avant la transition étiquetée

par r, et $\mathbf{y}'$ après cette transition. En fait, on encode les gardes et les opérations sur les compteurs par une seule formule r, dont les solutions sont $(\mathbf{y}, \mathbf{y}')$. Par convention, quand $Y = \emptyset$, alors $R_Y = \{\emptyset\}$.

Définition 5.5. *Un* Timed Counter System *(*TCS*) est un quadruplet $\langle Q, X, Y, T\rangle$ où :*
- *Q est un ensemble fini d'états de contrôle*
- *$T \subseteq Q \times R_X \times R_Y \times Q$ est un ensemble fini de transitions*

Notons qu'un TCS est en fait la combination de deux modèles bien connus : les automates temporisés (section 5.2) et les systèmes à compteurs (définition 4.1, avec $\mathscr{L}$ la logique de Presburger). Avec les notations de cette section, on peut redéfinir ces deux modèles comme suit :

Définition 5.6. *Un* automate temporisé *est un TCS $\mathcal{S}$ dans lequel $Y = \emptyset$. Un* système à compteurs *est un TCS $\mathcal{S}$ dans lequel $X = \emptyset$.*

Pour étudier le comportement d'un TCS, on peut envisager trois points de vue, selon les variables que l'on interprète. En effet, un TCS peut être analysé selon ses horloges seulement, ou selon ses compteurs seulement, ou les deux à la fois. On dit qu'un TCS interprété en fonction de ses horloges est un système de transitions d'horloges, noté ST_{horl}. De même, on dit qu'un TCS interprété en fonction de ses compteurs est un système de transitions de compteurs, noté ST_{cptr}. Si les horloges et les compteurs sont interprétés en même temps, alors la sémantique totale d'un TCS est donnée par un système de transitions, noté ST.

Définition 5.7. *La sémantique temporisée d'un TCS $\mathcal{S} = \langle Q, X, Y, T\rangle$ est donnée par un couple $ST_{horl}(\mathcal{S}) = \langle C_{horl}, \rightarrow_{horl}\rangle$, où :*
- *$SC_{horl} = Q \times \mathbb{R}_+^m$ est l'ensemble des configurations*
- *$\rightarrow_{horl} \subseteq C_{horl} \times (T \cup \mathbb{R}_+) \times C_{horl}$ est la relation de transition composée de délais et d'actions discrètes :*

$$(q, \mathbf{x}) \rightarrow_{horl} (q', \mathbf{x}') \iff \begin{cases} \textit{(délai, noté } \xrightarrow{\tau}_{horl} \textit{)} \\ q = q' \textit{ et } \exists \tau \in \mathbb{R}_+ \textit{ tel que } \mathbf{x}' = \mathbf{x} + \tau \\ \\ \textit{(action, notée } \xrightarrow{t}_{horl} \textit{)} \\ \exists t = (q, (g, \lambda), r, q') \in T \textit{ telle que } (\mathbf{x}, \mathbf{x}') \models (g, \lambda) \end{cases}$$

On remarque que si $\mathcal{S}$ est un automate temporisé, alors $ST_{horl}(\mathcal{S})$ donne la sémantique habituelle d'un automate temporisé.

Définition 5.8. *La sémantique de comptage d'un TCS $\mathcal{S} = \langle Q, X, Y, T\rangle$ est donnée par un couple $ST_{cptr}(\mathcal{S}) = \langle C_{cptr}, \rightarrow_{cptr}\rangle$, où :*

- $C_{cptr} = Q \times \mathbb{Z}^n$ *est l'ensemble des configurations*
- $\rightarrow_{cptr} \subseteq C_{cptr} \times T \times C_{cptr}$ *est la relation de transition définie par* $(q, \mathbf{y}) \xrightarrow{t}_{cptr} (q', \mathbf{y}') \iff \exists (q, (g, \lambda), r, q') \in T$ *telle que* $(\mathbf{y}, \mathbf{y}') \in r$

On remarque que si $\mathcal{S}$ est un système à compteurs, alors $ST_{cptr}(\mathcal{S})$ donne la sémantique habituelle des systèmes à compteurs.

Définition 5.9. *La sémantique totale d'un TCS* $\mathcal{S} = \langle Q, X, Y, T \rangle$ *est donnée par un couple* $ST(\mathcal{S}) = \langle C, \rightarrow \rangle$*, où :*

- $C = Q \times \mathbb{R}_+^m \times \mathbb{Z}^n$ *est l'ensemble des configurations*
- $\rightarrow \subseteq C \times (T \cup \mathbb{R}_+) \times C$ *est la relation de transition composée de délais et d'actions discrètes :*

$$(q, \mathbf{x}, \mathbf{y}) \rightarrow (q', \mathbf{x}', \mathbf{y}') \iff \begin{cases} \textit{(délai, noté } \xrightarrow{\tau} \textit{)} \\ q = q' \textit{ et } \mathbf{y} = \mathbf{y}' \textit{ et } \exists \tau \in \mathbb{R}_+ \textit{ tel que :} \\ \mathbf{x}' = \mathbf{x} + \tau \\ \\ \textit{(action, notée } \xrightarrow{t} \textit{)} \\ \exists t = (q, (g, \lambda), r, q') \in T \textit{ telle que :} \\ (\mathbf{y}, \mathbf{y}') \in r \textit{ et } (\mathbf{x}, \mathbf{x}') \models (g, \lambda) \end{cases}$$

En utilisant ces dernières définitions, on peut formuler la relation entre ces sémantiques comme suit :

Proposition 5.10. *Soit* $\mathcal{S} = \langle Q, X, Y, T \rangle$ *un TCS. Alors, on a :*

1. *Pour toute transition* $t \in T$*,*
   $(q, \mathbf{x}, \mathbf{y}) \xrightarrow{t} (q', \mathbf{x}', \mathbf{y}')$ *si et seulement si* $(q, \mathbf{x}) \xrightarrow{t}_{horl} (q', \mathbf{x}')$ *et* $(q, \mathbf{y}) \xrightarrow{t}_{cptr} (q', \mathbf{y}')$*.*
2. *Pour tout délai* $\tau \in \mathbb{R}_+$*,*
   $(q, \mathbf{x}, \mathbf{y}) \xrightarrow{\tau} (q, \mathbf{x}', \mathbf{y})$ *si et seulement si* $(q, \mathbf{x}) \xrightarrow{\tau}_{horl} (q, \mathbf{x}')$*.*

Exemple de TCS

La figure 5.2 donne un exemple de TCS, avec deux états de contrôle q_1, q_2, deux compteurs y_1, y_2, et deux horloges x_1, x_2. On considère la configuration initiale comme étant dans q_1 avec $y_1 = y_2 = x_1 = x_2 = 0$. On remarque que x_2 n'apparaît pas sur les transitions, mais sert uniquement d'horloge de référence.

Ce TCS représente un service proposé sur la plupart des télévisions numériques : on modélise ici une contrainte dans la location de films qu'un client peut effectuer depuis sa propre télévision numérique. Ce modèle donne principalement les informations suivantes : le nombre total de films qu'un client a loués jusqu'à présent (y_2), le nombre de

films loués depuis 24 heures (y_1), le temps écoulé depuis que le client utilise ce service (x_2), et le temps écoulé depuis la première location du jour (x_1).

Typiquement, la propriété que ce modèle vise à vérifier est “Un client peut louer au maximum 5 films dans une période de 24 heures”. On pourrait également modéliser les tarifs, et en utilisant x_2 et y_2, offrir un film gratuit au bout de 30 locations durant le premier mois d'abonnement. D'autres statistiques peuvent facilement être calculées dans ce modèle, comme le nombre moyen de films qu'un client loue par tranche de 10 heures.

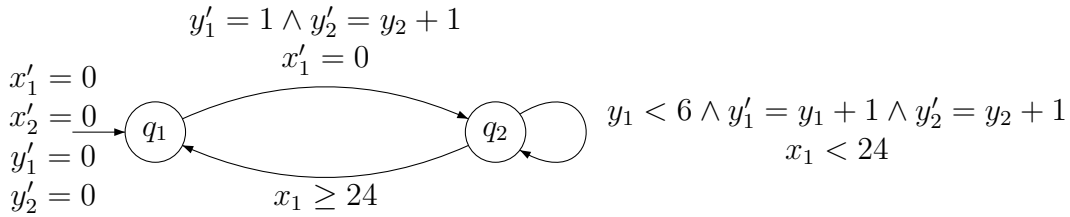


FIG. 5.2: Un exemple de TCS

Accessibilité

Un problème typique dans le domaine de la vérification est le problème d'accessibilité 1.9, qui consiste à décider, étant données deux configurations c et c' d'un système, s'il existe une exécution du système allant de c à c'. Ici, pour les TCS, on raffine ce problème : au lieu de vérifier si une configuration complète est accessible, on va ne s'intéresser qu'à la valeur des compteurs et à l'état de contrôle.

Formellement, soit $\mathcal{S}$ un TCS et $ST(\mathcal{S}) = \langle C, \rightarrow \rangle$ sa sémantique totale. On dénote par $\xrightarrow{*}$ la fermeture réflexive transitive de $\rightarrow$. De façon similaire, on définit $\xrightarrow{*}_{cptr}$ pour la sémantique de comptage.

On définit alors les *ensembles d'accessibilité* de $\mathcal{S}$ comme suit :
– Reach$(\mathcal{S}, c_0) = \{c \in C \mid c_0 \xrightarrow{*} s\}$, pour tout $c_0 \in C$
– Reach$_{cptr}(\mathcal{S}, c_0) = \{c \in C_{cptr} \mid c_0 \xrightarrow{*}_{cptr} c\}$, pour tout $c_0 \in C_{cptr}$
On s'intéresse ici au Problème d'Accessibilité avec Compteurs, que l'on définit comme suit :

Problème d'Accessibilité avec Compteurs :
Données : Un TCS $\mathcal{S}$, une configuration initiale c_0 de $ST(\mathcal{S})$, et une configuration

$(q, \mathbf{y})$ de $ST_{cptr}(\mathcal{S})$.

Question : Existe-t-il une valuation d'horloges $\mathbf{x}$ telle que $(q, \mathbf{x}, \mathbf{y}) \in \text{Reach}(\mathcal{S}, c_0)$?

Ce problème prend donc en compte les valuations de compteurs, mais pas d'horloges. On préfère s'intéresser à cette version partielle du problème d'accessibilité (plutôt qu'avec des configurations totales) parce que les horloges sont utilisées principalement pour introduire des contraintes temporelles dans le comportement du système ; par conséquent, dans une problématique de vérification, on n'a pas forcément besoin de mémoriser leurs valeurs exactes.

On remarque que ce Problème d'Accessibilité avec Compteurs est une extension du problème d'accessibilité 1.9 dans le cas des systèmes à compteurs : la seule différence est que l'on quantifie existentiellement sur la valuation d'horloges, afin que la configuration d'arrivée corresponde à une configuration complète de C, et pas seulement de C_{cptr}. On peut donc parler de ce Problème d'Accessibilité avec Compteurs aussi bien pour les TCS que pour les systèmes à compteurs (ce qui se ramène au problème 1.9 dans ce dernier cas).

Le Problème d'Accessibilité avec Compteurs est évidemment indécidable pour les TCS, puisqu'il l'est déjà pour les systèmes à compteurs. Afin d'analyser les systèmes à compteurs, des restrictions ont été données afin de les rendre décidables, comme par exemple les systèmes plats ou reversal-bornés, les VASS, etc. Comme on le montre dans la section 5.4.2, certaines de ces restrictions peuvent être portées au niveau des TCS. L'idée principale utilisée ici repose sur le fait que l'indécidabilité des TCS est causée par la présence de compteurs ; c'est pourquoi on tire profit des résultats connus sur les automates temporisés (détaillés dans la section 5.4.2) et sur certaines sous-classes des systèmes à compteurs (rappelés dans la section 5.4.3).

5.4.2 Analyse des TCS par abstraction des horloges

On cherche à analyser les TCS pour résoudre des problèmes de vérification ; puisque l'ensemble des configurations d'un TCS est infini, on ne peut pas essayer de les calculer. Une méthode classique pour analyser de tels systèmes infinis consiste à trouver une abstraction finie, en utilisant par exemple des classes d'équivalence sur les configurations, et ensuite à s'assurer que le problème d'accessibilité considéré peut être résolu en raisonnant sur le système abstrait. L'approche utilisée ici repose sur cette idée ; cependant, au lieu de raisonner sur des classes d'équivalence sur tout l'ensemble des configurations, on n'abstrait que les valuations d'horloges. Pour ce faire, on construit un *graphe des régions*, comme on le fait habituellement pour les automates temporisés.

On pourrait considérer l'approche duale : abstraire les compteurs d'abord, plutôt que

les horloges. Les deux principales raisons pour lesquelles on ne cherche pas à abstraire les compteurs sont (1) parce que les compteurs évoluent de façon discrète à travers les formules étiquetant les transitions, et non pas constamment dans un espace dense en restant dans un état de contrôle, et (2) parce que le graphe des régions a été étudié depuis longtemps et s'est avéré efficace dans plusieurs outils[2].

Construction du graphe des régions

Soit $\mathcal{S}$ un TCS à m horloges. Soit $\mathfrak{m}_i$ la plus grande constante à laquelle chaque horloge x_i sera comparée dans les gardes, pour tout $i \in [1, m]$. On considère la relation d'équivalence sur les valuations d'horloges, telle qu'elle est définie dans [AD94]. Deux valuations d'horloges $\mathbf{x}$ et $\mathbf{x}'$ dans $\mathbb{R}_+^m$ sont dites *équivalentes*, ce que l'on note $\mathbf{x} \approx \mathbf{x}'$, quand les trois conditions suivantes sont vérifiées (où $\lfloor v \rfloor$ (respectivement, $\llcorner v \lrcorner$) dénote la partie entière (respectivement, fractionnaire) de tout $v \in \mathbb{R}$) :

1. $\lfloor \mathbf{x}_i \rfloor = \lfloor \mathbf{x}'_i \rfloor$ ou $\mathbf{x}_i, \mathbf{x}'_i > \mathfrak{m}_i$ pour tout $i \in [1, m]$.
2. $\llcorner \mathbf{x}_i \lrcorner = 0$ si et seulement si $\llcorner \mathbf{x}'_i \lrcorner = 0$ pour tout $i \in [1, m]$ tel que $\mathbf{x}_i \leq \mathfrak{m}_i$.
3. $\llcorner \mathbf{x}_i \lrcorner \leq \llcorner \mathbf{x}_j \lrcorner$ si et seulement si $\llcorner \mathbf{x}'_i \lrcorner \leq \llcorner \mathbf{x}'_j \lrcorner$, pour tout $i, j \in [1, m]$ tel que $\mathbf{x}_i, \mathbf{x}_j \leq \mathfrak{m}_i$.

Cette relation d'équivalence peut être étendue aux configurations de $ST_{horl}(\mathcal{S})$, en précisant que $(q, \mathbf{x}) \approx (q', \mathbf{x}')$ si et seulement si $q = q'$ et $\mathbf{x} \approx \mathbf{x}'$. On écrit $[\mathbf{x}]$ pour désigner la classe d'équivalence à laquelle $\mathbf{x}$ appartient. Une *région* ρ est une classe d'équivalence de valuations d'horloges ; l'ensemble de toutes les régions est noté $\mathscr{R}$, et est fini. On écrit indifféremment $x \in \rho$ et $[x] = \rho$.

Une propriété agréable de la relation d'équivalence $\approx$ est qu'elle est compatible avec les contraintes d'horloges (ce que l'on note $(c.h.)$) et avec l'écoulement du temps (ce que l'on note $(\acute{e}.t.)$) :

$$\mathbf{x} \approx \mathbf{x}' \implies \begin{cases} (c.h.) & \forall g \in G_X,\ \mathbf{x} \models g \iff \mathbf{x}' \models g \\ (\acute{e}.t.) & \forall \tau \in \mathbb{R}_+, \exists \tau' \in \mathbb{R}_+ \text{ tel que } \mathbf{x} + \tau \approx \mathbf{x}' + \tau' \end{cases}$$

Ce deuxième point $(\acute{e}.t.)$ permet de définir une fonction "successeur" sur $\mathscr{R}$. Pour une région $\rho \in \mathscr{R}$, on note $\mathsf{Succ}(\rho)$ l'ensemble de ses *successeurs temporels*, définis comme suit : $\rho' \in \mathsf{Succ}(\rho) \iff \exists \mathbf{x} \in \rho\ \exists \tau \in \mathbb{R}_+$ tel que $\mathbf{x} + \tau \in \rho'$. On est alors capable de définir le graphe des régions de $\mathcal{S}$:

Définition 5.11. *Soit $\mathcal{S} = \langle Q, X, Y, T \rangle$ un TCS ; son* graphe des régions *est le couple $GR(\mathcal{S}) = \mathcal{S}_{/\approx} = \langle \Gamma, \rightarrow_{GR} \rangle$ tel que :*

[2] Les outils utilisent en fait des unions de régions, appelées *zones* ; mais le principe est le même.

– $\Gamma = Q \times \mathscr{R}$ *est l'ensemble des configurations ; on écrit parfois* $q_{\mathbf{x}}$ *pour désigner une configuration* $(q, [\mathbf{x}])$

– $\rightarrow_{GR} \subseteq \Gamma \times T \times \Gamma$ *est la relation de transition telle que* $\forall t = (q, (g, \lambda), r, q') \in T$, $(q, \rho) \xrightarrow{t}_{GR} (q', \rho')$ *si et seulement si* $\exists \rho'' \in \mathsf{Succ}(\rho)$ *tel que* $\forall \mathbf{x}'' \in \rho''$, $\mathbf{x}'' \models g$ *et* $\mathbf{x}' \in \rho'$ *et* $\forall i \in [1, m]$, $\mathbf{x}'_i = \lambda_i \mathbf{x}_i$.

Un tel graphe graphe de régions est le même que celui habituellement défini pour les automates temporisés ; sa particularité est que ses transitions sont étiquetées par des relations sur les compteurs, qui n'ont pas encore été prises en compte jusqu'alors. La prochaine étape est, bien entendu, d'utiliser ces relations sur les compteurs afin de se rapprocher de la sémantique totale d'un TCS.

Le graphe des régions est un système à compteurs

On montre ici que le graphe des régions d'un TCS peut être analysé comme un système à compteurs. Ensuite, on prouve que le problème d'accessibilité peut être porté au niveau du graphe des régions.

Le graphe des régions bénéficie de la propriété suivante [AD94] :

Proposition 5.12. *Soit* $\mathcal{S} = \langle Q, X, Y, T \rangle$ *un TCS,* $ST_{horl}(\mathcal{S}) = \langle C_{horl}, \rightarrow_{horl} \rangle$ *son système de transitions à horloges, et* $GR(\mathcal{S}) = \langle \Gamma, \rightarrow_{GR} \rangle$ *son graphe des régions. Alors, étant donné un* $(q, \mathbf{x}) \in C_{horl}$*, on a pour tout* $t \in T$ *:*

1. *Si* $\exists \tau \in \mathbb{R}_+$ *et* $\exists (q', \mathbf{x}')$ *tels que* $(q, \mathbf{x}) \xrightarrow{\tau}_{horl} (q, \mathbf{x} + \tau) \xrightarrow{t}_{horl} (q', \mathbf{x}')$*, alors* $q_{\mathbf{x}} \xrightarrow{t}_{GR} q'_{\mathbf{x}'}$
2. *Si* $\exists q'_{\mathbf{x}'}$ *tel que* $q_{\mathbf{x}} \xrightarrow{t}_{GR} q'_{\mathbf{x}'}$*, alors* $\exists \tau \in \mathbb{R}_+$ *et* $\exists \mathbf{x}'' \in [\mathbf{x}']$ *tels que* $(q, \mathbf{x}) \xrightarrow{\tau}_{horl} (q, \mathbf{x} + \tau) \xrightarrow{t}_{horl} (q', \mathbf{x}'')$

Cette propriété porte sur les transitions, et peut être naturellement étendue à des suites de telles transitions, c'est-à-dire à des exécutions. On obtient alors la fameuse *bisimulation à temps abstrait* entre $ST_{horl}(\mathcal{S})$ et $GR(\mathcal{S})$, notée $\simeq$. De manière informelle, $ST_{horl}(\mathcal{S}) \simeq GR(\mathcal{S})$ signifie que $ST_{horl}(\mathcal{S})$ et $GR(\mathcal{S})$ peuvent suivre exactement les mêmes exécutions ; la seule différence avec une bisimulation normale est que $GR(\mathcal{S})$ ne mémorise pas les valuations d'horloges, mais seulement leurs classes d'équivalence.

Notons que puisque le graphe des régions a un nombre fini de configurations et que ses transitions sont étiquetées par des relations sur les compteurs, on peut le voir comme un système à compteurs classique. En effet, on peut voir $GR(\mathcal{S})$ comme un TCS $\mathcal{S}' = \langle Q', X', Y', T' \rangle$, où $Q' = Q \times \mathscr{R}$, $X' = \emptyset$, $Y' = Y$ et $T' = T$ (avec $R_{X'} = \{\emptyset\}$,

puisque $X' = \emptyset$). Ainsi, on qualifiera indifféremment $GR(\mathcal{S})$ de TCS, de système à compteurs, et de graphe de régions, et ce, sans perte de généralité.

On peut à présent prouver que l'on peut analyser un TCS à travers la sémantique de comptage de son graphe des régions, en générant un système qui est une abstraction exacte (par rapport à l'accessibilité avec compteurs seulement) de sa sémantique totale. En effet, d'après les propositions 5.10 et 5.12, on déduit la propriété suivante :

Proposition 5.13. *Soit $\mathcal{S}$ un TCS. On a alors que :*

1. *si* $(q', \mathbf{x}', \mathbf{y}') \in Reach\Big(\mathcal{S}, (q, \mathbf{x}, \mathbf{y})\Big)$, *alors* $(q'_{\mathbf{x}'}, \mathbf{y}') \in Reach_{cptr}\Big(GR(\mathcal{S}), (q_{\mathbf{x}}, \mathbf{y})\Big)$
2. *si* $(q'_{\mathbf{x}'}, \mathbf{y}') \in Reach_{cptr}\Big(GR(\mathcal{S}), (q_{\mathbf{x}}, \mathbf{y})\Big)$, *alors* $\exists \mathbf{x}'' \in \mathbb{R}_+^m$ *telle que* $(q', \mathbf{x}'', \mathbf{y}') \in Reach\Big(\mathcal{S}, (q, \mathbf{x}, \mathbf{y})\Big)$ *et* $\mathbf{x}'' \in [\mathbf{x}']$.

Le dessin sur la figure 5.3 montre les différentes façons d'interpréter un TCS, ainsi que les relations existant entre elles. De plus, il illustre la proposition 5.13.

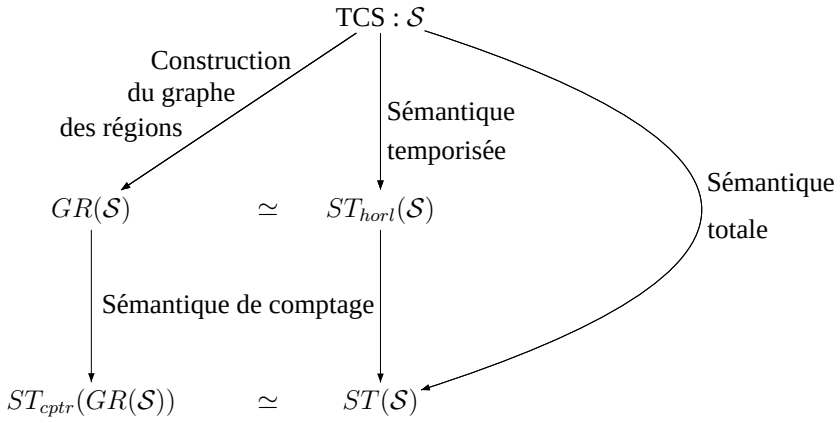


FIG. 5.3: Les liens entre les différentes sémantiques d'un TCS

Soit $\mathfrak{C}$ une classe de TCS pour laquelle il existe un algorithme qui résoud le Problème d'Accessibilité avec Compteurs pour $GR(\mathcal{S})$, et ce pour tout $\mathcal{S} \in \mathfrak{C}$. D'après la proposition 5.13 et le fait qu'il existe une nombre fini de régions, on en déduit le théorème principal :

Théorème 5.14. *Le Problème d'Accessibilité avec Compteurs est décidable pour $\mathfrak{C}$.*

Démonstration. Soit $(q, \mathbf{x}, \mathbf{y})$ un configuration initiale de $ST(\mathcal{S})$ et $(q, \mathbf{y})$ une configuration de $ST_{cptr}(\mathcal{S})$. Alors, d'après la proposition 5.13, on déduit qu'il existe une valuation d'horloges $\mathbf{x}'$ telle que $(q, \mathbf{x}', \mathbf{y}') \in Reach(\mathcal{S}, (q, \mathbf{x}, \mathbf{y}))$ si et seulement s'il existe

une région ρ telle que $((q,\rho),\mathbf{y}') \in Reach_{cptr}(GR(\mathcal{S}),(q_{\mathbf{x}},\mathbf{y}))$ et $\mathbf{x}' \in \rho$. Puisqu'un TCS donné génère un nombre fini de régions, si on suppose que le Problème d'Accessibilité avec Compteurs est décidable pour le système à compteurs $GR(\mathcal{S})$, alors le Problème d'Accessibilité avec Compteurs est décidable pour $\mathcal{S}$. □

Dans la section suivante, on utilise ce théorème pour montrer que plusieurs des restrictions menant à la décidabilité quand on étudie les systèmes à compteurs peuvent être appliquées également aux TCS, dans le but d'obtenir la décidabilité du Problème d'Accessibilité avec Compteurs.

5.4.3 Sous-classes de TCS

On peut à présent traiter le Problème d'Accessibilité avec Compteurs pour les TCS, en fonction de la classe de systèmes à compteurs à laquelle le graphe des régions du TCS appartient. Dans cette section, on introduit quatre sous-classes de TCS, dans l'optique de résoudre le Problème d'Accessibilité avec Compteurs.

Machines à compteurs temporisées et TVASS

On définit ici la classe des machines à compteurs temporisées, qui sont une extension des machines à compteurs d'Ibarra (définition 4.3). Une machine à compteurs est un système à compteurs dont les formules étiquetant les transitions sont des translations gardées ; on rappelle qu'une translation gardée (définition 4.2) est un triplet $(\succ,\mu,\gamma)$, où $\mathbf{y} \succ \mu$ est la garde et $\mathbf{y}' = \mathbf{y} + \delta$ est la translation de la valuation de compteurs.

En fait, on a défini une translation gardée comme une fonction $t : \mathbb{N}^n \longrightarrow \mathbb{N}^n$; on peut donc également la voir comme une relation sur $\mathbb{Z}^n \times \mathbb{Z}^n$. En effet, pour une translation gardée $t : \mathbb{N}^n \longrightarrow \mathbb{N}^n$ et deux valuations de compteurs $\mathbf{y}$ et $\mathbf{y}'$, on a $(\mathbf{y},\mathbf{y}') \in t$ si et seulement si $\mathbf{y} \in \mathsf{dom}(t)$ et $\mathbf{y}' = t(\mathbf{y})$. Ainsi, pour reprendre le formalisme des TCS, une translation gardée est une relation de la forme $\bigwedge_{i\in[1..n]} \mathbf{y}_i \succ_i \mu_i \wedge \mathbf{y}'_i = \mathbf{y}_i + \gamma_i$.

Définition 5.15. *Une* machine à compteurs temporisée *est un TCS $\mathcal{S} = \langle Q, X, Y, T\rangle$ tel que pour toute transition $(q,(g,\lambda),r,q') \in T$, r est une translation gardée.*

On remarque que le Problème d'Accessibilité avec Compteurs reste indécidable pour les machines à compteurs temporisées. Afin d'obtenir la décidabilité, une solution est de restreindre les translations gardées, en particulier en interdisant les tests d'égalité dans les gardes. Cette restriction nous ramène aux TVASS, qui sont une extension temporisée des VASS, un modèle équivalent aux réseaux de Petri (voir sections 4.2 et 5.3.2). Afin qu'ils correspondent au formalisme des TCS, on adapte la définition des TVASS, introduits sous le nom Timed P/T Nets dans [GS94][3] :

[3] En fait, le problème du vide d'un langage d'un TVASS a été prouvé décidable dans [GS94] et [Bér95].

Définition 5.16. *Un* Timed Vector Addition System with States *(abbrévié* TVASS*) est une machine à compteurs temporisée $\mathcal{S} = \langle Q, X, Y, T\rangle$ telle que pour toute transition $(q, (g, \lambda), r, q') \in T$, r est une translation gardée $(\succ, \mu, \gamma)$ telle que $\succ = (\geq, \ldots, \geq)$.*

Propriétés d'un TCS et de son graphe des régions

On peut restreindre les systèmes à compteurs de plusieurs façons afin d'obtenir la décidabilité du Problème d'Accessibilité avec Compteurs. Tout d'abord, on remarque que les restrictions définies précédemment restent valides lorsque l'on construit le graphe des régions associé :

Proposition 5.17. *Soit $\mathcal{S}$ un TCS. Si $\mathcal{S}$ est une machine à compteurs temporisée (respectivement, un TVASS), alors le système à compteurs $GR(\mathcal{S})$ est une machine à compteurs (respectivement, un VASS).*

Puisque le Problème d'Accessibilité avec Compteurs est décidable pour les VASS [Kos82, May84], alors en utilisant le théorème 5.14, on en déduit que :

Théorème 5.18. *Le Problème d'Accessibilité avec Compteurs est décidable pour les TVASS.*

Les deux définitions 5.15 et 5.16 sont des restrictions syntaxiques ; néanmoins, il est possible de restreindre le comportement d'un TCS (c'est-à-dire, sa sémantique). On dit qu'un couple $(\mathcal{S}, c_0)$ est un *TCS initialisé* (respectivement, un *système à compteurs initialisé*), dans lequel $\mathcal{S}$ est un TCS (respectivement, un système à compteurs) et c_0 la configuration initiale de $ST(\mathcal{S})$ (respectivement, $ST_{cptr}(\mathcal{S})$). Parmi les restrictions possibles sur la sémantique, on peut considérer les TCS (respectivement, les systèmes à compteurs) initialisés et bornés, pour lesquels toutes les valuations de compteurs restent toujours en-dessous d'une certaine borne, dans toutes les exécutions possibles. D'après le théorème 5.14, on en déduit que :

Proposition 5.19. *Si un TCS initialisé $(\mathcal{S}, c_0)$ est borné, alors le système à compteurs initialisé $(GR(\mathcal{S}), c_0')$ est borné, avec $c_0 = (q, \mathbf{x}, \mathbf{y})$ et $c_0' = (q_{\mathbf{x}}, \mathbf{y})$.*

Le Problème d'Accessibilité avec Compteurs est évidemment décidable pour les systèmes à compteurs initialisés et bornés, puisqu'il n'y a qu'un nombre fini de configurations accessibles. On en déduit alors que :

Théorème 5.20. *Le Problème d'Accessibilité avec Compteurs est décidable pour les TCS initialisés et bornés.*

Enfin, on s'intéresse à une autre restriction sémantique, mais cette fois sur les machines à compteurs temporisées. Dans [Iba78], la classe des machines à compteurs *reversal-bornées* est introduite, et a été étendue dans [FS08]. Cette extension mentionne

qu'une machine à compteurs initialisée $(\mathcal{S}, c_0)$ est *k-reversal-b-bornée* pour $k, b \in \mathbb{N}$, si dans toutes les exécutions de $\mathcal{S}$ partant de c_0, chaque valuation de compteur alterne au plus k fois entre des suites non-croissantes et non-décroissantes au-dessus d'une borne b (voir la section 4.1). On étend naturellement cette notion aux machines à compteurs temporisées ; on remarque qu'une machine à compteurs temporisée initialisée est reversal-bornée si elle est k-reversal-b-bornée avec $k, b \in \mathbb{N}$. D'après la proposition 5.13, on déduit que :

Proposition 5.21. *Si une machine à compteurs temporisée initialisée $(\mathcal{S}, c_0)$ est reversal-bornée, alors la machine à compteurs initialisée $(GR(\mathcal{S}), c'_0)$ est aussi reversal-bornée, avec $c_0 = (q, \mathbf{x}, \mathbf{y})$ et $c'_0 = (q_{\mathbf{x}}, \mathbf{y})$.*

Puisque le Problème d'Accessibilité avec Compteurs est décidable pour les machines à compteurs reversal-bornées [FS08], on a que :

Théorème 5.22. *Le Problème d'Accessibilité avec Compteurs est décidable pour les machines à compteurs temporisées reversal-bornées.*

Le tableau suivant résume les résultats de décidabilité que l'on vient d'obtenir :

Modèle	**Graphe des régions**	**P.A.C.**
TCS	SC	Indécidable
TVASS	VASS	Décidable
MCT r.b.	MC r.b.	Décidable
TCS bornés	SC bornés	Décidable

FIG. 5.4: Récapitulatif des résultats de décidabilité sur les TCS

Abbréviations utilisées dans ce tableau : P.A.C. = Problème d'Accessibilité avec Compteurs, SC = Systèmes à Compteurs, MC = Machines à Compteurs, MCT = Machines à Compteurs Temporisées, r.b. = reversal-bornées.

La classe des TVASS est récursive, ce qui est particulièrement intéressant du point de vue de l'implémentabilité de ces résultats. On propose alors un algorithme (page 125) qui résoud le Problème d'Accessibilité avec Compteurs pour cette classe. Cependant, les deux autres classes décidables apparaissent moins intéressantes de ce point de vue, puisqu'il est impossible de décider si un système est reversal-borné ou borné, dans le cas général.

Algorithme 5.1 : Résoud le Problème d'Accessibilité avec Compteurs pour les TVASS.

Entrée : un TVASS $\mathcal{S}$, une configuration $(q, \mathbf{y})$, et une configuration initiale c_0
Sortie : la réponse à "Existe-t-il une valuation $\mathbf{x}$ telle que $(q, \mathbf{x}, \mathbf{y}) \in \mathrm{Reach}(\mathcal{S}, c_0)$?"

```
construire GR(S) = ⟨Γ, →_GR⟩
Pour Tout q'_x ∈ Γ Faire
  Si q' = q Alors
    Si (q_x, y) ∈ Reach_cptr(GR(S), c_0) Alors
      renvoyer Vrai
    Fin Si
  Fin Si
Fin Pour
renvoyer Faux
```

Conclusion relative aux TCS

On a donc défini un nouveau modèle de systèmes combinant compteurs et horloges, et prouvé que le Problème d'Accessibilité avec Compteurs est décidable pour trois de ses sous-classes. D'autres sous-classes devraient être également intéressantes, notamment les TCS plats, en suivant les approches proposées dans [CJ99] ou [BFLS05].

De plus, le théorème 5.14 peut être étendu à d'autres types de données que des compteurs, comme par exemple des piles, des canaux non-fiables, etc.

Une autre direction de recherche consiste à étudier les liens possibles entre les compteurs et les horloges d'un TCS, bien qu'il n'y en ait a priori aucun ; les deux paragraphes suivants donnent quelques pistes de réflexion sur ces liens.

Influences des compteurs sur les horloges. En utilisant un encodage assez simple, on peut simuler une opération permettant d'attendre y_2 unités de temps, c'est-à-dire d'effectuer $x_i := x_i + y_2$, $\forall x_i \in X$, y_2 étant la valeur d'un compteur. Pour ce faire, on a besoin d'une horloge supplémentaire x_1, qui vaudra 0 à la fin de l'opération. De plus, avec un compteur supplémentaire y_1, on peut éviter de modifier la valeur de y_2 en la copiant dans y_1 (sinon y_2 vaudra 0 à la fin de l'opération). La figure 5.5 donne l'encodage réalisant cette opération, en copiant la valeur de y_2 dans y_1.

De plus, on peut effectuer l'opération $x_2 := y_2$ en se basant sur cet encodage : il suffit de remettre l'horloge x_2 à zéro au départ, en ajoutant $x'_2 = 0$ sur la transition sortant de l'état "init". Cependant, cet encodage a pour effet secondaire d'augmenter toutes les horloges de y_2, ce qui montre les limites de ce genre d'encodage.

Simultanéité des opérations. Un autre type de lien entre compteurs et horloges consiste en la simultanéité des opérations sur les transitions. En effet, pour deux relations

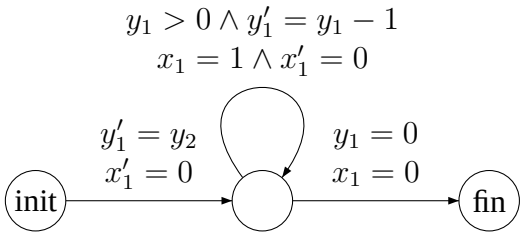


FIG. 5.5: Un TCS simulant l'opération $\mathbf{x}_i := \mathbf{x}_i + \mathbf{y}_2$, $\forall x_i \in X \setminus \{x_1\}$

$r_X \in R_X$ et $r_C \in R_C$, les valuations des horloges pourront être différentes selon que l'on interprète r_X avant r_C ou non. Prenons un exemple : soit $r_X \equiv (x < 4)$ et $r_C \equiv (y < 5 \wedge y' = y + 1)$. Dans tous les cas, après interprétation des deux relations, on sait que $y \leq 5$; par contre, la veleur de x dépend de l'ordre d'exécution. Si l'on exécute une transition étiquetée à la fois par r_X et r_C, alors x sera dans l'intervalle $[0, 4[$ après franchissement de la transition ; si on sépare cette transition en deux, la première étiquetée par r_C et la suivante par r_X, alors le résultat après franchissement de la deuxième transition sera le même. Par contre, à l'inverse, si on exécute d'abord r_X puis r_C, alors x pourra valoir n'importe quel réel positif ! En effet, on pourra avoir un délai entre les deux transitions, d'une durée arbitrairement grande.

D'autres liens existent probablement entre compteurs et horloges avec cette définition de TCS, bien que l'on ait choisi de séparer les variables en deux ensembles disjoints, chacun ayant une syntaxe et une sémantique indépendante de l'autre.

Pour aller plus loin, on pourrait envisager d'autoriser, par exemple, des gardes du type $x > y$, c'est-à-dire de comparer une horloge à un compteur. Cette extension modifie le graphe des régions, puisque les valeurs auxquelles sont comparées les horloges ne sont plus constantes. Ainsi, la région deviendrait de plus en plus petite à mesure que y augmente, et on pourrait donc construire le graphe de régions dynamiquement, en calculant les limites de telles régions lorsqu'elles convergent vers un point fixe.

On pourrait également considérer d'autres opérations, comme affecter une horloge à un compteur ($y := x$). La réciproque pourrait être intéressante à étudier également, en autorisant des opérations du type $x := y$: cela change de la simple remise à zéro généralement autorisée sur les automates temporisés, mais on pourra s'inspirer de la thèse de Patricia Bouyer [Bou02] pour ce genre de mise à jour des horloges avec d'autres valeurs que 0.

Chapitre 6

Machines à choix non-déterministe d'incrément dense

Dans ce chapitre, on étudie un modèle de systèmes à compteurs assez original, introduit par Zhe Dang, Oscar H. Ibarra, Pierluigi San Pietro, et Gaoyan Xie [XDISP03]. Ce modèle a été défini dans cet article, puis étendu dans [DISPX04] avec un ruban d'entrée bidirectionnel contenant des mots réels séparés par des marqueurs. Ces deux articles étaient les seuls étudiant ce modèle, jusqu'à ce que nous décidions, avec Pierluigi San Pietro, de clarifier et de formaliser les définitions, de les étendre, et de compléter les résultats de ces deux articles. Ce chapitre décrit ces travaux, publiés dans [BFSP09].

Une machine à choix non-déterministe d'incrément dense, notée DCM (pour Dense Counter Machine), est une machine de Minsky pouvant en plus incrémenter ou décrémenter ses compteurs d'une valeur réelle choisie de façon non-déterministe sur l'intervalle $]0, 1[$. L'intérêt de telles machines est de pouvoir modéliser des systèmes hybrides dans lesquels on peut faire des choix non-déterministes ne pouvant pas être modélisés facilement par des automates temporisés ou hybrides. Bien entendu, les problèmes non-triviaux restent indécidables sur ce modèle ; cependant, on s'intéresse à des sous-classes pour lesquelles la relation d'accessibilité reste décidable, comme les DCM reversal-bornées.

On s'intéresse également aux DCM purement à choix dense, qui sont des DCM ne pouvant pas incrémenter ni décrémenter de 1. En effet, un compteur à choix dense est par définition plus général qu'un compteur classique (discret). On montre alors que pour les DCM purement à choix dense, même bornées, le problème d'accessibilité d'un état de contrôle est indécidable (même pour quatre compteurs purement à choix dense et bornés par 1).

Afin de modéliser plus facilement les systèmes hybrides, on introduit la capacité de

tester un compteur par rapport à une constante entière (pas seulement 0). Ceci paraît légitime, puisque la plupart des modèles de systèmes à compteurs ont trivialement cette capacité. Dans le cas des DCM reversal-bornées, cette extension pose des problèmes techniques, mais on montre que les résultats de décidabilité restent les mêmes. Une des causes de ces difficultés est que l'encodage intuitif pour simuler un test à une constante $k > 0$ (i.e. plusieurs décréments et tests à 0, suivis par des incréments pour restaurer la valuation originelle du compteur) ne préserve pas le caractère reversal-borné. Toutefois, on contourne ce problème au moyen d'une longue preuve technique, pour étendre un résultat de [XDISP03], en montrant que la relation d'accessibilité d'une DCM avec un compteur à choix dense et un nombre fini de compteurs à choix dense, reversal-bornés, et testables à une constante k, reste définissable par une formule mixte.

Enfin, on caractérise la relation d'accessibilité des DCM reversal-bornées à l'aide de la logique $\mathrm{FO}(\mathbb{R}, \mathbb{Z}, +, <)$. On prouve en effet que toute formule mixte est la relation d'accessibilité d'une DCM reversal-bornée, ce qui complète le résultat réciproque de [XDISP03].

6.1 Motivation

On rappelle qu'une machine de Minsky est un $\mathscr{L}$-Système à Compteurs dans lequel les relations de $\mathscr{L}$ sont de la forme $(x' = x+1)$, $(x' = x = 0)$, $(x > 0 \wedge x' = x-1)$, ou $true$. Bien que le problème d'accessibilité soit indécidable dans ces machines (avec au moins deux compteurs), on souhaite étendre leur expressivité, et ce pour deux raisons.

Premièrement, si une machine de Minsky est reversal-bornée, alors sa relation d'accessibilité devient calculable. Ainsi, on voudrait pouvoir utiliser un modèle plus puissant que les machines de Minsky reversal-bornées, mais qui reste décidable. Ce premier point est détaillé dans les deux sections suivantes (6.2 et 6.3).

Deuxièmement, les machines de Minsky constituent un modèle très basique et fort peu pratique pour modéliser ou exprimer des propriétés de haut niveau. C'est pour cette raison que l'on introduit la possibilité d'utiliser des compteurs réels, et de choisir de façon non-déterministe la valeur des incréments/décréments pour chaque transition. Le restant de cette section est consacré à la discussion de ces deux extensions (compteurs réels et choix non-déterministes).

Afin de manipuler des compteurs réels, on définit les machines de Minsky denses, qui sont des $\mathscr{L}$-Systèmes à Compteurs dans lesquels les relations de $\mathscr{L}$ sont de la forme $(x' = x+r)$, $(x' = x = 0)$, $((x-r > 0 \vee x-r = 0) \wedge x' = x-r)$, ou $true$, étant donné

un ensemble fini de valeurs[1] $r \in \mathbb{Q}_+$. Comme souvent dans les machines de Minsky, on considère que la valuation initiale des compteurs est toujours 0.

Cette première extension n'est pas vraiment plus puissante, puisqu'une machine de Minsky peut simuler une machine de Minsky dense :

Proposition 6.1. *Les machines de Minsky et les machines de Minsky denses sont bisimilaires.*

Démonstration. Un sens est évident, en prenant $r = 1$. L'autre sens est un peu moins direct, mais reste facile. Il suffit de simuler chaque instruction d'une machine de Minsky dense par une machine de Minsky. Il y a quatre instructions à simuler, et deux d'entre elles le sont immédiatement : $true$ et $x' = x = 0$. Pour les deux autres instructions, $(x' = x + r)$ et $((x - r > 0 \vee x - r = 0) \wedge x' = x - r)$, il suffit d'encoder r par un entier. Chaque incrément/décrément $r \in \mathbb{Q}_+$ peut être écrit $\frac{p}{q}$, avec $p, q \in \mathbb{N}$. Puisqu'on connaît tous les r possibles à l'avance (dans la définition syntaxique du système), on peut calculer pour chaque r un $q' \in \mathbb{N}$ tel que $r = \frac{pq'}{q_{ppcm}}$, où q_{ppcm} est le plus petit commun multiple de tous les r. Ainsi, chaque r peut être représenté par un entier non-négatif $r' = pq'$, et les nouvelles valeurs des compteurs seront toutes multipliées par le même facteur q_{ppcm}. À l'aide de cet encodage simple, on peut donc simuler une instruction $x' = x + r$ par une suite de r' instructions $x' = x + 1$. De la même façon, $((x - r > 0 \vee x - r = 0) \wedge x' = x - r)$ peut être simulé par une suite de r' instructions $(x > 0 \wedge x' = x - 1)$. □

Une autre façon d'étendre les machines de Minsky est de permettre, sur chaque transition, un choix non-déterministe de l'incrément/décrément. On appelle cette extension une machine de Minsky à choix dense : on la définit comme un $\mathscr{L}$-Système à Compteurs dans lequel les relations de $\mathscr{L}$ sont de la forme $(x' = x + 1)$, $(x' = x = 0)$, $(x > 0 \wedge x' = x - 1)$, $(x' = x + \Delta)$, $((x - \Delta > 0 \vee x - \Delta = 0) \wedge x' = x - \Delta)$, ou $true$, où Δ symbolise une valeur $\delta \in \mathbb{R}_+$ choisie de façon non-déterministe. Le choix est fait à chaque fois qu'une transition est franchie, de sorte que deux transitions consécutives étiquetées par $x' = x + \Delta$ devraient avoir des valeurs différentes pour Δ : le choix est aléatoire, et on ne connaît pas la valeur choisie (bien que l'on puisse la vérifier a posteriori, à l'aide d'une transition et d'un compteur supplémentaires).

Ici, on ne considère que les exécutions de longueur finie : on montre que la valeur δ peut être choisie sur $]0, 1[$ au lieu de tout $\mathbb{R}_+$:

Proposition 6.2. *Pour des exécutions de longueur finie, toute machine de Minsky à choix dense $\mathcal{M}$ pour laquelle $\delta \in \mathbb{R}_+$ peut être simulée par une machine de Minsky à choix dense $\tilde{\mathcal{M}}$ pour laquelle $\delta \in]0, 1[$.*

[1] Notons que l'on prend ici les valeurs dans $\mathbb{Q}_+$ parce que les propriétés importantes sont (1) le caractère dense, et (2) une représentation effective de tout nombre rationnel (ce qui n'est pas le cas pour les réels, en général).

Idée de la preuve. Toute exécution e de $\mathcal{M}$ peut être simulée par une exécution $\tilde{e}$ de $\tilde{\mathcal{M}}$, éventuellement plus longue, pour laquelle les incréments δ appartiennent à l'intervalle ouvert $]0, 1[$. Sans preuve formelle ni définition formelle des états de contrôle et configurations, on montre que, par exemple, une transition de q vers q', étiquetée par $(x'_1 = x_1 + \Delta), (x'_2 = x_2 + \Delta), (x_3 - \Delta > 0 \vee x_3 - \Delta = 0) \wedge (x'_3 = x_3 - \Delta)$, peut être simulée dans $\tilde{\mathcal{M}}$ (chaque Δ étant remplacé à chaque franchissement d'une transition par un $\delta \in \mathbb{R}_+$).

Premièrement, $\tilde{\mathcal{M}}$ a une transition de q vers un nouvel état de contrôle q'', étiqueté par les même formules, mais avec $0 < \delta < 1$. Deuxièmement, $\tilde{\mathcal{M}}$ contient une transition de q'' à q'' lui-même (i.e. bouclante), elle aussi étiquetée par les mêmes formules, avec $0 < \delta < 1$. Troisièmement, $\tilde{\mathcal{M}}$ contient une transition de q'' vers q' Étiquetée par $x'_1 = x_1$, $x'_2 = x_2$, $x'_3 = x_3$. Par conséquent, une configuration c' ayant pour état de contrôle q' et valuations de compteurs $(x'_1, x'_2, x'_3) \in \mathbb{R}^3_+$ est accessible dans $\mathcal{M}$ depuis une configuration c, d'état de contrôle s et valuations de compteurs $(x_1, x_2, x_3) \in \mathbb{R}^3_+$, si et seulement si c' est accessible dans $\tilde{\mathcal{M}}$ depuis c. □

Ainsi, on ne perd pas en généralité quand on suppose que chaque incrément est dans l'intervalle $]0, 1[$ (du moins, pas tant que l'on considère des exécutions de longueur finie). De plus, un tel incrément borné permet un meilleur contrôle, avec une granularité plus fine. Le fait est que, dans de nombreux systèmes réels, les variables représentent des données physiques et sont donc bornées (par exemple, le niveau d'eau d'un réservoir, qui est une valeur réelle non-négative ne pouvant pas dépasser la capacité du réservoir). Il semble en effet difficile de modéliser ou de vérifier ce genre de comportement avec un Système à Compteurs dont les incréments sont des réels non-bornés.

Enfin, on remarque que permettre des incréments dans l'intervalle $]0, p[$, pour un $p \in \mathbb{N}$ donné, n'apporte aucune expressivité par rapport au cas où $p = 1$. Par exemple, pour incrémenter un compteur x de n'importe quelle valeur δ avec $0 < \delta < p$, il suffit d'appliquer, dans une machine de Minsky à choix dense dont les incréments sont dans $]0, 1[$, une suite d'exactement p transitions (ce qui est possible, puisque p est fixé), chacune de la forme $x' = x + \delta$, pour $0 < \delta < 1$.

Dans la section suivante, on généralise et formalise la définition des machines de Minsky à choix denses dont on vient de montrer l'intérêt.

Exemple : système producteur-consommateur. Un exemple d'application simple de machine à compteurs réels est la version suivante du fameux système producteur-consommateur, décrit dans [XDISP03]. Le système peut être dans l'un de ces trois états : *prod*, *conso*, ou *repos*. Dans l'état *prod*, il fabrique une ressource pouvant être stockée et utilisée plus tard dans l'état *conso*. La resource est un nombre réel représentant une

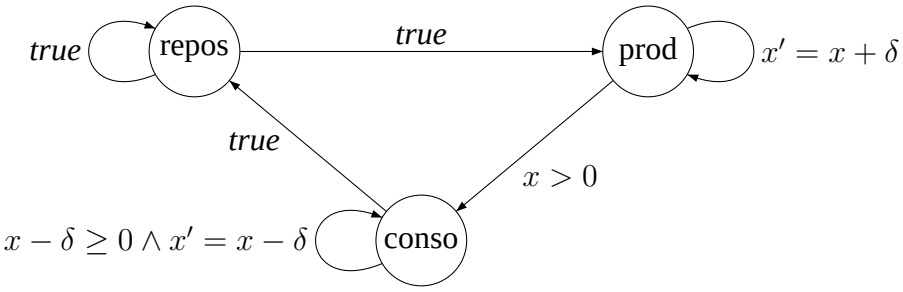


FIG. 6.1: Une DCM modélisant un système producteur-consommateur

grandeur physique : par exemple, de l'essence ou de l'eau. La production peut être utilisée immédiatement, plus tard, ou jamais.

Un tel système peut se modéliser par un automate fini muni d'un compteur à choix dense, comme montré surla figure 6.1. La ressource est additionnée au compteur lorsqu'elle est produite, et soustraite lorsqu'elle est consommée.

Bien que l'on utilise un compteur réel, la variable continue (i.e. la ressource) ne change que de façon discrète, par des incréments ou décréments arbitraires. Toutefois, cette modélisation reste acceptable dans de nombreux cas où la grandeur physique modélisée change de façon continue, puisque les incréments et décréments peuvent être arbitrairement petits.

De plus, puisque le compteur ne peut pas décroître en-deçà de zéro, le système implémente naturellement la contrainte physique imposant que la consommation ne doit jamais excéder la production. D'autres contraintes plus complexes sont décidables, notamment si elles sont exprimables par des contraintes linéaires sur les valeurs de compteurs. Par exemple, une requête décidable est de savoir si la production totale n'excède jamais deux fois la consommation totale.

6.2 Définitions et propriétés des DCM

Dans la section 6.2.1, on définit les DCM, puis on s'intéresse à leurs variations, notamment : k-testables, purement à choix dense, et discrètes. La section 6.2.2 est consacrée à la définition et à l'étude des DCM reversal-bornées.

6.2.1 DCM et k-testabilité

Soit X un ensemble de n variables, appelées *compteurs à choix dense* (ou simplement *compteurs* dans la suite de ce chapitre, sauf précision contraire). Les compteurs à choix dense étaient appelés "compteurs denses" (dense counters) dans [XDISP03] :

nous les rebaptisons afin de clarifier leur fonctionnement et leur rôle. Une *valuation de compteurs* est une fonction qui associe à tout $x_i \in X$ une valeur dans $\mathbb{R}_+$. Puisqu'il n'y a pas vraiment d'ambigüité et que la signification est évidente, nous faisons ici l'amalgame entre compteur(s) et valuation(s) de compteurs, tous deux notés x_i (ou X).

Soit $G = \{(x = 0), (x > 0), true\}$ l'ensemble des gardes. On dit qu'une valuation de compteurs X satisfait une garde $\mathbf{g} \in G^n$, ce que l'on note $X \models \mathbf{g}$, quand la substitution de chaque compteur par sa valeur rend la formule $\mathbf{g}$ vraie (comme d'habitude). Par exemple, si $n = 3$ et $\mathbf{g} = (true, x_2 > 0, x_3 = 0)$, alors $(6, 2, 0) \models \mathbf{g}$ mais $(6, 2, 1) \not\models \mathbf{g}$.

Soit $A = \{1, \Delta\}$ l'ensemble des actions. Intuitivement, 1 représente un incrément ou décrément entier, et Δ représente un incrément ou décrément réel choisi de façon non-déterministe.

Définition 6.3. *Une* machine à choix dense *(abbréviée DCM, pour Dense-choice Counter Machine) avec $n > 0$ compteurs est un couple $\mathcal{M} = \langle Q, T \rangle$ où :*

- *Q est un ensemble fini d'états de contrôle, avec un état $q_{fin} \in Q$ dit* final *;*
- *$T \subseteq Q \times \Sigma \times Q$ est un ensemble fini de transitions, avec $\Sigma = (G \times \mathbb{Z} \times A)^n$*

Une DCM est donc un $\mathscr{L}$-Système à Compteurs avec $\mathscr{L} = (G \times \mathbb{Z} \times A)^n$, dans lequel on distingue un état de contrôle spécifique q_{fin}.

Intuitivement, la composante entière de Σ, $\boldsymbol{\lambda} \in \mathbb{Z}^n$, est un facteur qui détermine si la transition incrémente ou bien décrémente un compteur, et de quelle valeur. Parallèlement, l'action $\mathbf{a} \in A^n$ détermine si l'incrément ou décrément est une valeur entière ou bien réelle.

Par souci de lisibilité, on écrit parfois les transitions de manière plus simple, comme $x > 0 \wedge x := x + 3\delta$, ce qui veut dire que la garde sur le compteur x_i est $g_i = (x > 0)$, que son facteur est $\lambda_i = 3$, et que son action est $a_i = \Delta$.

Remarque : Nos transitions sont équivalentes à celles de [XDISP03, DISPX04], pour lesquelles les auteurs utilisent la notion de *mode*. Les modes `stay`, `unit increment`, `unit decrement`, `fractional increment`, et `fractional decrement` sont, ici, représentés respectivement par $(\lambda_i = 0)$, $(\lambda_i > 0 \wedge a_i = 1)$, $(\lambda_i < 0 \wedge a_i = 1)$, $(\lambda_i > 0 \wedge a_i = \Delta)$, et $(\lambda_i < 0 \wedge a_i = \Delta)$.

Remarque : Les transitions pour lesquelles $\lambda \in \{+1, -1\}^n$ sont juste un cas particulier, et elles peuvent simuler toute combinaison linéaire de la forme $x'_i = x_i + \sum_{j=1}^{m} \lambda_j \delta_j$, pour un entier m et un ensemble de valeurs $\delta_j \in]0, 1[$ donnés.

Comme habituellement en vérification, on interprète une DCM d'abord en spécifiant une valuation initiale à chacun de ses compteurs ainsi qu'un état de contrôle initial, puis

on laisse la machine fonctionner de façon non-déterministe. Le comportement d'une DCM consiste principalement à choisir une transition dont la garde $\mathbf{g} \in G^n$ est satisfaite par la valuation courante des compteurs, et à mettre à jour ces valuations en allant vers le nouvel état de contrôle.

Remarque : Dans une DCM, il y a bien deux sources distinctes de non-déterminisme. La première est l'habituelle, qui consiste à avoir potentiellement plusieurs transitions disponibles à un moment donné. La deuxième, elle, est spécifique à ce modèle, et réside dans le choix explicite de la valeur d'un incrément ou décrément réel. Ainsi, pour une suite de transitions donnée, la première source de non-déterminisme est inhibée, mais la deuxième persiste, faisant que les valuations de compteurs ne sont pas prévisibles.

Définition 6.4. *La sémantique d'une DCM $\mathcal{M} = \langle Q, T\rangle$ est donnée par un système de transitions $TS(\mathcal{M}) = \langle C, \rightarrow\rangle$ où :*

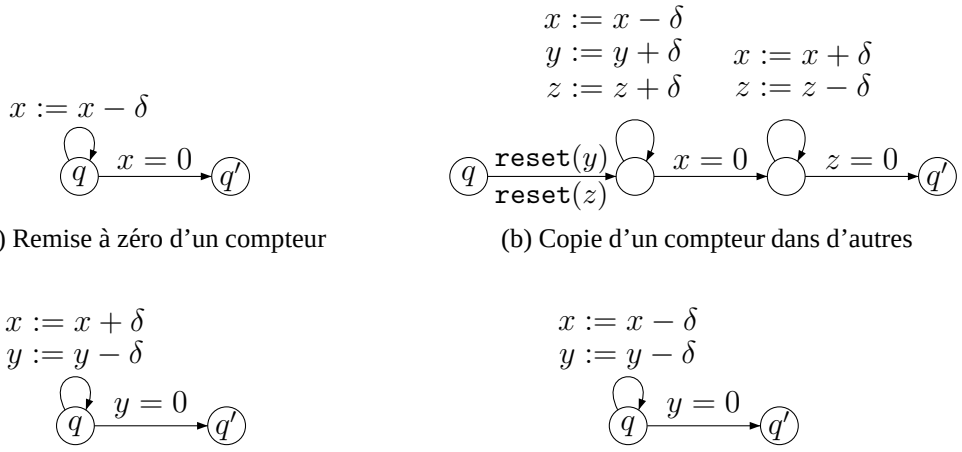
- *$C = Q \times \mathbb{R}_+^n$ est l'ensemble des configurations*
- *$\rightarrow \subseteq C \times \Sigma \times C$ est l'ensemble des transitions, défini par :*
  $(q, X) \xrightarrow{\mathbf{g},\boldsymbol{\lambda},\mathbf{a}} (q', X')$ si et seulement si $(q, (\mathbf{g}, \boldsymbol{\lambda}, \mathbf{a}), q') \in T$ et $\exists \delta \in \mathbb{R}$ tel que $0 < \delta < 1 \wedge X \models \mathbf{g} \wedge X' = X + \boldsymbol{\lambda}\mathbf{u}$, avec $\mathbf{u} = \mathbf{a}[\Delta \leftarrow \delta]$

Pour une DCM $\mathcal{M} = \langle Q, T\rangle$ et son système de transitions $TS(\mathcal{M}) = \langle C, \rightarrow\rangle$, la relation d'accessibilité $\rightsquigarrow_{\mathcal{M}}$ est la fermeture transitive et réflexive $\rightarrow^*$; quand le contexte est clair, on omet l'indice $_{\mathcal{M}}$. Une *exécution* de $\mathcal{M}$ est une séquence $(q^0, X^0) \rightarrow (q^1, X^1) \rightarrow \ldots \rightarrow (q^l, X^l)$, de longueur $l \geq 0$. À cause du non-déterminisme inhérent à une DCM, on ne s'intéresse qu'aux exécutions se terminant dans l'état final $q_{fin} \in Q$. Formellement, une exécution de $\mathcal{M}$ est dite *acceptante* si elle est de la forme $(q, X) \rightarrow^* (q_{fin}, X')$, quels que soient $q \in Q$ et $X, X' \in \mathbb{R}_+^n$; on dit également que $\mathcal{M}$ *accepte* cette exécution. On dit qu'une DCM $\mathcal{M}$ *rejette* une exécution $(q, X) \rightarrow^* (q', X')$, si $q' \in Q$ est un état "puits" mais non-final (c'est-à-dire que l'exécution ne pourrait pas être étendue en une exécution acceptante).

On rappelle que l'ensemble des couples $\Big((q, X), (q', X')\Big) \in C \times C$ tels que $(q, X) \rightsquigarrow (q', X')$ est appelé la *relation d'accessibilité* de $\mathcal{M}$ (parfois appelée *binary reachability*, en anglais).

L'exemple sur la figure 6.1 est une DCM, si l'on supprime la garde $x - \delta \geq 0$ (la machine rejette les exécutions dans lesquelles cette garde n'est pas vérifiée, de toute façon).

Bien qu'une DCM n'ait qu'un ensemble restreint d'opérations possibles sur les compteurs, elle peut effectuer des macro-opérations telles que remettre à zéro la valeur d'un compteur (`reset`), copier la valeur d'un compteur dans un ou plusieurs autres compteurs (`copy`), additionner la valeur d'un compteur à celle d'un autre (`add`), soustraire la valeur d'un compteur à celle d'un autre (`minus`), comparer les valeurs de deux



(a) Remise à zéro d'un compteur

(b) Copie d'un compteur dans d'autres

(c) Addition d'un compteur à un autre

(d) Soustraction d'un compteur à un autre

FIG. 6.2: Encodages des opérations `reset`, `copy`, `add`, et `minus`

compteurs, etc. Voici les encodages de ces quatre premières opérations, que nous utilisons comme des raccourcis syntaxiques.

On représente par $q \xrightarrow{\texttt{reset}(x)} q'$, $q \xrightarrow{\texttt{copy}(x,y)} q'$, $q \xrightarrow{\texttt{add}(x,y)} q'$, et $q \xrightarrow{\texttt{minus}(x,y)} q'$ les DCM sur les Figures 6.2a, 6.2b, 6.2c, et 6.2d (respectivement).

Soit $G'_k = \{(x = i), (x < i), (x > i), true\}_{i \in [0,k]}$, pour un $k \in \mathbb{N}$ donné. Une DCM dont l'ensemble des gardes est inclus dans G'_k est appelée une *k-DCM*. Les compteurs d'une k-DCM sont dits *k-testables*. On souligne le fait que toute DCM est une 0-DCM, et que les compteurs à choix dense sont 0-testables, sauf mention contraire.

Proposition 6.5. *Toute k-DCM peut être simulée par une DCM, pour un $k \in \mathbb{N}$ donné.*

Démonstration. Il y a trois sortes de tests supplémentaires dans les gardes : $x < k$, $x > k$, et $x = k$, pour un $k \in \mathbb{N}$ donné. On montre comment encoder chacun d'eux en utilisant seulement des tests $x = 0$ et $x > 0$.

Un test $x < k$, représenté par une transition $q \xrightarrow{x < k} q'$, peut être simulé par l'encodage suivant :

Notons que pour éviter de modifier la valeur de x, on la copie dans un autre compteur y en utilisant l'encodage de la Figure 6.2b. De manière plus simple, un test $x > k$ (respectivement, $x = k$) peut être simulé par une suite de k décréments de 1, suivie par un test $x > 0$ (respectivement, $x = 0$). On remarque également que le test $x < 0$ ne pourrait jamais être vérifié, puisqu'un compteur ne peut pas prendre de valeurs négatives : la machine rejetterait une telle exécution avant même d'être capable d'effectuer ce test. □

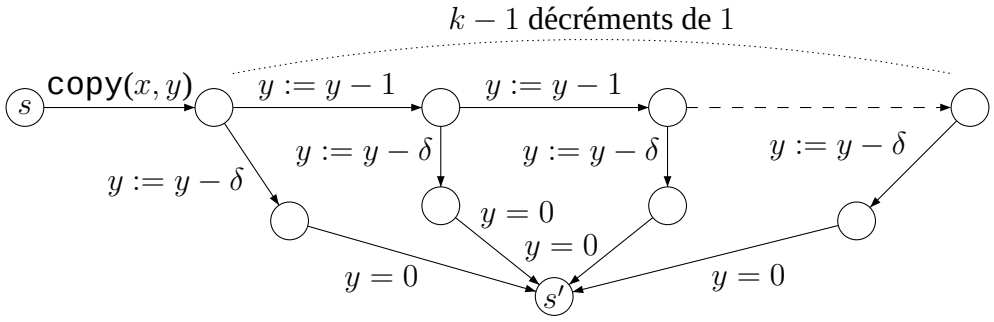


FIG. 6.3: Une DCM simulant un test de la forme $x < k$

Définition 6.6. *Soit $\mathcal{M}$ une DCM. Un compteur x_i de $\mathcal{M}$ est* purement à choix dense *si et seulement si $a_i = \Delta$ dans chaque transition (c'est-à-dire qu'il n'est jamais incrémenté de 1 ni décrémenté de 1). Réciproquement, un compteur x_i est* (purement) discret *si et seulement si $a_i = 1$ dans chaque transition (c'est-à-dire que c'est un compteur classique). Si $\mathcal{M}$ contient seulement des compteurs purement à choix dense, on l'appelle une* DCM pure. *Si $\mathcal{M}$ contient seulement des compteurs discrets, on l'appelle une* machine à compteurs (discrète)*, correspondant à la définition 4.3.*

6.2.2 DCM reversal-bornées

On étend à présent la définition de compteurs reversal-bornés de [Iba78] aux DCM. Soit $\mathcal{M} = \langle Q, T \rangle$ une DCM, $q, q' \in Q$, et $r \in \mathbb{N}$. On appelle *renversement* le passage des facteurs λ_i, entre deux transitions, d'une valeur non-négative à une valeur positive, ou d'une valeur négative ou nulle à une valeur positive, pour un i donné. Sur une exécution de q à q', un compteur x_i est *r-reversal-borné* s'il y a au plus r reversements tout au long de l'exécution, et ce, pour tout i. Un compteur x_i est *reversal-borné* (noté *r.b.*) s'il existe un r tel que x_i est r-reversal-borné sur toute exécution acceptante de $\mathcal{M}$. $\mathcal{M}$ est une machine reversal-bornée à choix dense, notée *r.b. DCM*, si chaque compteur de $\mathcal{M}$ est reversal-borné.

Un compteur qui n'est pas nécessairement reversal-borné est dit *libre*.

Dans ce modèle, on peut effectivement décider si un compteur est r-reversal-borné, en utilisant un état de contrôle pour symboliser le fait que les transitions incrémentent ($\lambda_i > 0$) ou décrémentent ($\lambda_i < 0$) le compteur x_i. Ainsi, on peut utiliser des états de contrôle supplémentaires pour se souvenir de chaque renversement, et rejeter une exécution si elle contient plus de r renversements.

Tout comme dans le cas de compteurs discrets, on peut toujours supposer que $r = 1$. En effet, chaque suite de "incréments, puis décréments" peut être simulée par un comp-

teur 1-reversal-borné, et ainsi, un compteur effectuant r renversements peut être simulé par r compteurs 1-reversal-bornés.

Grâce à [XDISP03], on sait que la relation d'accessibilité d'une DCM reversal-bornée, augmentée d'un compteur libre, peut être définie par une formule mixte, dont la satisfiabilité est décidable (voir la section 6.4 pour des détails sur les formules mixtes). Ce résultat est énoncé dans [XDISP03] comme suit :

Proposition 6.7. *La relation d'accessibilité d'une DCM avec un compteur 0-testable libre et un nombre fini de de compteurs r.b. k-testables est définissable par une formule mixte, étant donné $k \geq 0$.*

Cependant, on a étendu les gardes des DCM pour qu'elles puissent comparer un compteur à une constante entière donnée. Dans la proposition 6.5, on a prouvé que cette extension n'est pas plus puissante dans le cas général ; mais c'est loin d'être évident quand on se limite aux DCM reversal-bornées. En effet, la preuve de la proposition 6.5 utilise un encodage qui ne préserve pas le caractère reversal-borné.

On prouve à présent, dans le théorème 6.7, que cette extension à la k-testabilité n'est en fait pas plus puissante dans le cas des r.b. DCM, là non plus, pour peu que l'on se permette d'utiliser beaucoup plus de compteurs. Mais avant cela, quelques définitons techniques sont requises.

Pour tout réel x, on définit $fr(x) = 0$ si $\lfloor x \rfloor = x$ (i.e., x est un entier), et $fr(x) = 1/2$ sinon. Étant donné un ensemble fini Q (les états de contrôle de $\mathcal{M}$) et un entier $k > 0$, soit $Q' = Q \times (\{0 \dots k\} \times \{0, 1/2\})^n$. Une DCM $\mathcal{M}' = \langle Q', T' \rangle$ munie de n compteurs 0-testables est appelée une *DCM à tests finis*. Une configuration $\langle \ll q, (d_1, f_1), \dots, (d_n, f_n) \gg, X \rangle$ de $\mathcal{M}'$ est *consistante* si, pour tout $1 \leq i \leq n$, soit $(x_i \leq k \wedge d_i = \lfloor x_i \rfloor \wedge f_i = fr(x_i))$ soit $(x_i > k \wedge d_i = k \wedge f_i = 1/2)$ est vrai. Ainsi, dans une configuration consistante, la comparaison d'un compteur à une constante $j \leq k$ donne le même résultat qu'un test sur les composantes d et f de l'état de contrôle.

En général, $\mathcal{M}'$ peut aussi atteindre des configurations inconsistantes. Une exécution de $\mathcal{M}'$ est *consistante* si elle ne passe que par des configurations consistantes.

Maintenant, on a besoin d'un lemme technique que l'on peut énoncer comme suit :

Lemme 6.8. *Pour toute k-DCM $\mathcal{M}$ à n compteurs, il existe une DCM à tests finis $\mathcal{M}'$ à n compteurs telle que : (1) le compteur x_i de $\mathcal{M}'$ est r.b. si le compteur x_i de $\mathcal{M}$ est r.b., (2) toute exécution de $\mathcal{M}$ est aussi une exécution de $\mathcal{M}'$, et (3) une exécution consistante de $\mathcal{M}'$ est aussi une exécution de $\mathcal{M}$.*

Démonstration. Plus formellement, l'énoncé que l'on prouve est le suivant.
Soit $\mathcal{M} = \langle Q, T \rangle$ une DCM munie de n compteurs x, k-testables, pour un $k > 0$ donné. Alors, il existe une DCM à tests finis $\mathcal{M}' = \langle Q', T' \rangle$ à n compteurs 0-testables telle que :

1. si le compteur x_i de $\mathcal{M}$ est r.b. alors le compteur x_i de $\mathcal{M}'$ est lui aussi r.b.
2. pour toute exécution de $\mathcal{M}$ de longueur $l \geq 0$

$$\langle q^1, X^1\rangle \rightarrow^l_{\mathcal{M}} \langle q^l, X^l\rangle,$$

il existe une exécution de longueur l pour $\mathcal{M}'$

$$\langle \ll q^1, (d^1_1, f^1_1), \ldots, (d^1_n, f^1_n) \gg, X^1\rangle \rightarrow^l_{\mathcal{M}'} \langle \ll q^l, (d^l_1, f^l_1), \ldots, (d^l_n, f^l_n) \gg, X^l\rangle.$$

3. pour chaque exécution consistante de $\mathcal{M}'$ de longueur $l \geq 0$

$$\langle \ll q^1, (d^1_1, f^1_1), \ldots, (d^1_n, f^1_n) \gg, X^1\rangle \rightarrow^l_{\mathcal{M}'} \langle \ll q^l, (d^l_1, f^l_1), \ldots, (d^l_n, f^l_n) \gg, X^l\rangle,$$

il existe une exécution de longueur l pour $\mathcal{M}$, de la forme

$$\langle q^1, X^1\rangle \rightarrow^l_{\mathcal{M}} \langle q^l, X^l\rangle.$$

L'idée de la preuve est de construire $\mathcal{M}'$ pour qu'elle réplique le comportement de $\mathcal{M}$, en représentant les variations des valeurs de compteurs par ses états de contrôle. Par simplicité, on ne considère que le cas $n = 1$, mais la preuve peut être facilement généralisée à n'importe quel nombre fini de compteurs. Ainsi, un état de contrôle de Q' est un triplet $\ll q, d, f \gg$, où $q \in Q$, d est un entier dans $0, \ldots, k$ et f vaut soit 0 soit $1/2$. La composante d est utilisée comme un compteur discret allant de 0 à k, supposé représenter $\lfloor x_1 \rfloor$. La composante f, quant à elle, représente la partie décimale (ou fractionnelle) de x_1 : $f = 0$ si $\lfloor x_1 \rfloor = x_1$, et $f = 1/2$ sinon. La définition de T' est telle que toutes les comparaisons d'un compteur x à une constante $0 < j \leq k$ sont éliminées et remplacées par des tests sur d et f (c'est-à-dire sur la structure de contrôle). Par exemple, un test $x > j$ est remplacé par un test $d > j \vee (d = j \wedge f = 1/2)$. Seuls les tests à 0 sont dupliqués dans T'.

Formellement, T' est défini comme suit.

Soit $(q, (\mathbf{g}, \lambda, \mathbf{a}), q') \in T$. On définit g'_1 comme valant $true$ si g_1 vaut $true$, $x_1 < j$, $x_1 = j$, ou $x_1 > j$, et ce pour tout $j > 0$; dans le cas contraire, on définit g'_1 comme valant $x_1 = 0$ ou $x_1 > 0$ si g_1 vaut, respectivement, $x_1 = 0$ ou $x_1 > 0$. Ainsi, g'_1 est obtenu depuis g_1 en éliminant toutes les comparaisons à une constante $j > 0$. Pour chaque $d \in \{0, \ldots, k\}, f \in \{0, 1/2\}$, si l'une des conditions suivantes est vraie :

- $g_1 = true$, ou
- $g_1 = (x_1 < j)$ et $d < j$, ou
- $g_1 = (x_1 = j)$ et $d = j \wedge f = 0$, ou
- $g_1 = (x_1 > j)$ et $d > j \vee (d = j \wedge f = 1/2)$,

alors $(\ll q, d, f \gg, (g, \lambda, a), \ll q', d', f' \gg) \in T'$ pour chaque $d' \in 0, \ldots, k$, $f' \in \{0, 1/2\}$ tel que $\lambda'_1 = \lambda_1 \wedge a'_1 = a_1$ et l'une des cinq conditions suivantes est vraie :

1. $\lambda_1 = 0 \wedge d' = d \wedge f' = f$ (mode `stay`)
2. $\lambda_1 = 1 \wedge a_1 = 1 \wedge \Big((d < k \wedge d' = d+1 \wedge f' = f) \vee (d = k \wedge d' = d \wedge f' = 1/2)\Big)$ (mode `integer increment`)
3. $\lambda_1 = -1 \wedge a_1 = 1 \wedge \Big((0 < d < k \wedge d' = d-1 \wedge f' = f) \vee \Big(d = k \wedge d' = d \wedge (f' = 1/2 \vee f' = 0)\Big)\Big)$ (mode `integer decrement`)
4. $\lambda_1 = 1 \wedge a_1 = \Delta \wedge \Big((f = 0 \wedge d' = d \wedge f' = 1/2) \vee (f = 1/2 \wedge d' = d+1 \wedge f' = 0) \vee (f = 1/2 \wedge d' = d+1 \wedge f' = 1/2) \vee (f = 1/2 \wedge d' = d \wedge f' = 1/2)\Big)$ (mode `fractional increment`)
5. $\lambda_1 = -1 \wedge a_1 = \Delta \wedge \Big((f = 0 \wedge d > 0 \wedge d' = d-1 \wedge f' = 1/2) \vee (f = 1/2 \wedge d > 0 \wedge d' = d-1 \wedge f' = 1/2) \vee (f = 1/2 \wedge d' = d \wedge f' = 0) \vee (f = 1/2 \wedge d' = d \wedge f' = 1/2)\Big)$ (mode `fractional decrement`)

On remarque que dans les cas (4) et (5), il peut y avoir plus d'une alternative qui soit vraie (c'est-à-dire, dans les disjonctions entre parenthèses), ce qui correspond aux choix non-déterministes de $\mathcal{M}'$. De plus, dans le cas (5) (mode `fractional decrement`), si $f = 0 \wedge d = 0$ alors $\mathcal{M}'$ rejette l'exécution, puisqu'il n'y a pas d'alternative possible.

Cette définition implémente l'élimination des tests. Soit $(\ll q, d, f \gg, (g', \lambda, a), \ll q', d', f' \gg) \in T'$. Si le test originel g_1 compare à une constante $j > 0$, alors g'_1 ne nécessite qu'un test sur les composantes d and f, mais pas sur x_1. Sinon, si g_1 est un test à 0, alors g'_1 est un test $x_1 = 0$, mais la structure de contrôle doit également vérifier que d et f valent 0. Similairement, si g_1 vaut $x_1 > 0$, alors g'_1 vaut aussi $x_1 > 0$ et $d > 0 \vee (d = 0 \wedge f = 1/2)$ doit être vérifié. Ceci entraîne que si, durant une exécution, on rencontre un test $x_1 = 0$ pendant que $x_1 = 0 \wedge (d > 0 \vee f = 1/2)$, alors $\mathcal{M}'$ rejette l'exécution.

On montre maintenant que la machine qu'on vient de définir respecte les trois points énoncés par ce lemme.

La condition (1) est immédiate, puisque l'on peut effectivement vérifier si un compteur est r-reversal-borné, pour un $r \geq 0$ donné, en observant quand les transitions incrémentent ($\lambda_i > 0$) ou décrémentent ($\lambda_i < 0$) chaque compteur x_i. Ainsi, on peut utiliser des états de contrôle supplémentaires pour se souvenir de chaque renversement, et rejeter si le nombre de renversements dépasse r.

La condition (2) du lemme est évidente, puisque, par construction, chaque transition dans une exécution de $\mathcal{M}$ peut être dupliquée dans une exécution de $\mathcal{M}'$. Ainsi,

une exécution de $\mathcal{M}$ est également une exécution de $\mathcal{M}'$, en ajoutant les composantes requises dans les états de contrôle.

La condition (3) suit elle aussi, puisque dans une configuration consistante, chaque comparaison d'un compteur à une constante j telle que $0 < j < k$ est équivalente à un test sur l'état de contrôle. Ainsi, une exécution consistante dans $\mathcal{M}'$ peut aussi être dupliquée dans $\mathcal{M}$. □

En général, une exécution d'une DCM à tests finis (telle que ci-dessus) n'est *pas* forcément une exécution de $\mathcal{M}$, puisque $\mathcal{M}'$ n'est pas k-testable à proprement parler, bien qu'elle puisse tester les composantes d et f de ses états de contrôle. En effet, les modes `fractional increment` et `fractional decrement` peuvent mener à une configuration inconsistante, dans laquelle la valeur d'un compteur x_i n'est pas compatible avec la valeur des $i^{èmes}$ composantes de d et f. Par exemple, on pourrait atteindre une configuration dans laquelle $x \leq j$ et $d > j$, étant donné $j > 0$. C'est pourquoi les tests sur d et f peuvent ne pas donner les mêmes résultats que les tests sur la valeur de x, et ainsi l'exécution pourrait être possible dans $\mathcal{M}'$ mais pas dans $\mathcal{M}$.

Lemme 6.9. *Une DCM $\mathcal{M} = \langle Q, T\rangle$ avec un compteur libre 0-testable et n compteurs 1-r.b. k-testables est équivalente à une DCM $\mathcal{M}^0 = \langle Q^0, T^0\rangle$ avec un compteur libre 0-testable et $2n + 2k(n + 1)$ compteurs r.b. 0-testables.*

L'idée de la preuve, détaillée ci-après, est la suivante. On commence par construire une DCM à tests finis $\mathcal{M}'$ comme dans le lemme 6.8 ; elle servira d'intermédiaire. ensuite, on définit $\mathcal{M}^0$ comme ayant ses exécutions séparées en deux phases.

La première phase simule une exécution de $\mathcal{M}'$ sur les n premiers compteurs, en utilisant des tests sur les états de contrôle plutôt que de véritables tests sur les compteurs aux positions 1 à n. Cependant, durant cette phase de simulation, $\mathcal{M}^0$ duplique les valeurs des x_i dans les $n(k + 1)$ premiers compteurs supplémentaires.

La deuxième phase vérifie que l'exécution simulée de $\mathcal{M}'$ est bien consistante, en inspectant les valeurs stockées dans les compteurs supplémentaires, et rejetant l'exécution si et seulement si la simulation n'était pas consistante. C'est-à-dire, elle vérifie que si $\mathcal{M}^0$ entre dans une configuration où $d_i = j \wedge f_i = 1/2$, alors $j < x_i < j + 1$. Remarquons au passage que les compteurs additionnels sont toujours reversal-bornés.

Par conséquent, $\mathcal{M}^0$ simule $\mathcal{M}$ fidèlement.

Au début, la preuve suppose quelques restrictions sur le comportement des compteurs et sur les tests ; ces hypothèses sont ensuite levées, à la fin de la preuve.

Démonstration. Supposons que tous les compteurs r.b. de $\mathcal{M}$ sont en fait 0-reversal, c'est-à-dire qu'ils ne subissent aucun renversement : le compteur ne peut jamais revenir à une valeur qu'il a déjà eue. Supposons aussi que ces compteurs r.b. partent de 0, dans $\mathcal{M}$. Supposons également qu'il n'y a aucun test d'égalité à une constante $j > 0$, c'est-à-dire que seuls les tests $x > j$ ou $x < j$ sont autorisés. Supposons enfin que $\mathcal{M}$ n'a pas

de compteur libre, et n'a donc que n compteurs 1-r.b. k-testables. Toutes ces hypothèses seront supprimées à la fin de la preuve.

Soit $\mathcal{M}' = \langle Q', T' \rangle$ une DCM à tests finis qui vérifie le lemme 6.8.

On considère maintenant $\mathcal{M}^0$.

Soit $Q^0 = Q' \times \{\text{SIMUL}, \text{CHECK}\}$. Si $\mathcal{M}$ démarre dans un état q_0, avec tous ses compteurs initialement nuls, alors $\mathcal{M}^0$ démarre dans l'état $\ll (q_0, (0,0)^n), \text{SIMUL} \gg$, avec tous ses compteurs initialements nuls.

$\mathcal{M}^0$ fonctionne en deux phases : d'abord dans une phase de simulation SIMUL, puis dans une phase de vérification CHECK. De façon correspondante, T^0 est l'union des deux ensembles de transitions T_{SIMUL} et T_{CHECK}.

La phase SIMUL simule une exécution de $\mathcal{M}'$ sur les n premiers compteurs, en utilisant des tests sur la structure de contrôle plutôt que des tests sur les compteurs en position 1 à n. Cependant, durant cette phase, $\mathcal{M}^0$ duplique les valeurs stockées dans x_i dans les $n(k+1)$ premiers compteurs supplémentaires.

La phase CHECK vérifie que l'exécution simulée de $\mathcal{M}'$ est bien consistante, en vérifiant les valeurs des compteurs supplémentaires, et en rejetant l'exécution si et seulement si elle n'est pas consistante (par exemple en vérifiant que si $\mathcal{M}^0$ atteint une configuration dans laquelle $d_i = j \wedge f_i = 1/2$, alors $j < x_i < j+1$). Ainsi, $\mathcal{M}^0$ peut simuler fidèlement $\mathcal{M}$.

Par souci de clarté, on utilise un encodage pour les positions des compteurs ; soit $p(i,j) = n + (i-1) * (k+1) + j + 1$, pour tout $1 \leq i \leq n$, $0 \leq j \leq k$. Ainsi, $p(1,0), p(1,1), \ldots, p(1,k)$ sont les indices des compteurs $x_{n+1}, x_{n+2}, \ldots, x_{n+k+1}$, et $p(2,0), p(2,1), \ldots, p(2,k)$ sont les indices des compteurs $x_{n+k+2}, x_{n+k+3}, \ldots, x_{n+2k+2}$, etc.

T_{SIMUL} est défini comme suit.

1. pour tout $(q'_1, (\mathbf{g}', \boldsymbol{\lambda}, \mathbf{a}), q'_2) \in T'$, la transition $(\ll q'_1, \text{SIMUL} \gg, (\mathbf{g}', \boldsymbol{\lambda}^0, \mathbf{a^0}), \ll q'_2, \text{SIMUL} \gg)$ est dans T_{SIMUL}, si et seulement si, pour i tel que $1 \leq i \leq n$:
   - $\lambda^0_i = \lambda_i, a^0_i = a_i$;
   - $\lambda^0_{p(i,0)} = \cdots = \lambda^0_{p(i,d_i-1)} = 0$ (les compteurs correspondants ne bougent pas) ;
   - si q'_1 ne vérifie pas $d_i = k \wedge f_i = 1/2$, alors $\lambda^0_{p(i,d_i+1)} = \lambda^0_{p(i,d_i+2)} = \lambda^0_{p(i,k+1)} = \lambda_i$, $a_{p(i,d_i+1)} = a_{p(i,d_i+2)} = a_{p(i,k+1)} = a_i$ (les compteurs correspondants font le même mouvement que x_i) ;
   - si q'_1 est tel que $d_i = k \wedge f_i = 1/2$, alors $\lambda^0_{p(i,d_i+1)} = 0$.
2. pour tout $q' \in Q'$, $(\ll q', \text{SIMUL} \gg, (\mathbf{true}, \mathbf{0}, \mathbf{a}), \ll q', \text{CHECK} \gg)$ appartient à T_{SIMUL}, quel que soit $\mathbf{a}$ (où $\mathbf{0}$ (respectivement, $\mathbf{true}$) est le vecteur contenant 0 (respectivement, $true$) dans chaque composante).
3. aucune transition autre que celles définies en (1) et (2) n'appartient à T_{SIMUL}.

La signification des transitions définies en (2) est de faire entrer $\mathcal{M}^0$ dans la phase CHECK, qui sert à vérifier que les tests sur la structure de contrôle utilisés dans la phase SIMUL étaient corrects.

Sans définir T_{CHECK} formellement, on décrit comment $\mathcal{M}^0$ peut vérifier que $j < x_i < j+1$ quand elle atteint une configuration où $d_i = j \wedge f_i = 1/2$, pour tout $i, 1 \leq i \leq n$ et pour tout j, $0 \leq j \leq k$. La valeur de x_i au moment précis où cette configuration est atteinte est encore stockée dans $x_{p(i,j)}$. Puisque le compteur x_i ne peut par effectuer de renversement, alors pour s'assurer que la configuration était consistante avec x_i à ce moment-là, il suffit de s'assurer que $j \leq x_{p(i,j)} < j+1$, et rejeter l'exécution si ce n'est pas le cas.

Soit $z_{i,j}$ la valeur des $x_{p(i,j)}$ quand $\mathcal{M}^0$ arrive dans la phase CHECK. $\mathcal{M}^0$ décrémente $x_{p(i,j)}$ d'exactement j (avec une suite de décréments de 1). Si la machine ne rejette pas, alors $z_{i,j} \geq j$. Dans ce cas, pour s'assurer que $z_{i,j} < j+1$ (i.e. $0 \leq x_{p(i,j)} < 1$ pour la valeur courante de $x_{p(i,j)}$), si $\mathcal{M}^0$ vérifie $x_{i,j+1} = 0$ alors $z_{i,j} = j$. Sinon, $\mathcal{M}^0$ décrémente $x_{p(i,j)}$ de Δ pour le mettre à zéro, et rejette l'exécution si cette remise à zéro non-déterministe échoue. Ainsi, le seul cas où il existe une exécution non-rejetée est quand il existe δ, $0 < \delta < 1$ tel que $x_{p(i,j)} = \delta$. $\mathcal{M}^0$ répète cette procédure pour tout $i, 1 \leq i \leq n$ et pour tout j, $0 \leq j \leq k$. Enfin, M^0 termine son exécution. Il est clair que si $\mathcal{M}^0$ termine son exécution sans la rejeter, c'est que les tests originels de $\mathcal{M}$ ont été devinés correctement par $\mathcal{M}^0$.

L'interdiction d'utiliser des tests d'égalité peut être levée en remarquant qu'un test $x_i = k$ est remplacé dans $\mathcal{M}^0$ par un test $d_i = k \wedge f_i = 0$. Il suffit donc que la machine marque, dans son état de contrôle, la valeur de f_i quand d_i devient égal à j. Si $f_i = 0$, alors la phase CHECK ne devrait s'assurer que de $z_{i,j} = j$, plutôt que de vérifier si $j \leq z_{i,j} < j+1$.

La restriction d'avoir seulement des compteurs 0-reversal peut être éliminée en ajoutant $n(k+1)$ compteurs 1-r-b. 0-testables dans $\mathcal{M}^0$, et en étendant la phase SIMUL à l'utilisation de ces compteurs supplémentaires. Soit $\tilde{p}(i,j)$ la valeur $n + n(k+1) + (i-1)*(k+1) + j + 1$. Chaque compteur $x_{\tilde{p}(i,j)}$ effectue le même mouvement que x_i, avec $0 \leq j \leq k$ et $1 \leq i \leq n$, tant que x_i est dans une phase croissante (c'est-à-dire que $\lambda^0(\tilde{p}(i,j)) = \lambda(i)$). Quand la phase décroissante commence pour x_i, $\mathcal{M}^0$ étant dans un état $\ll q, d_1, \ldots, d_n, f_1, \ldots, f_n \gg$, alors SIMUL agit sur les compteurs aux positions $\tilde{p}(i,j)$, avec λ^0 et a^0 définis comme suit :

- $\lambda^0_{\tilde{p}(i,d_i)} = \lambda^0_{\tilde{p}(i,d_i+1)} = \cdots = \lambda^0_{\tilde{p}(i,k)} = 0$ (les compteurs correspondants ne bougent pas) ;
- si q'_1 ne vérifie pas $d_i = k \wedge f_i = 1/2$, alors $\lambda^0_{\tilde{p}(i,0)} = \lambda^0_{\tilde{p}(i,1)} = \cdots = \lambda^0_{\tilde{p}(i,d_i-1)} = \lambda_i$ et $a^0_{\tilde{p}(i,0)} = a^0_{\tilde{p}(i,1)} = \cdots = a^0_{\tilde{p}(i,d_i-1)} = a_i$ (les compteurs correspondants effectuent le même mouvement que le $i^{ème}$ compteur) ;
- si q'_1 est tel que $d_i = k \wedge f_i = 1/2$, alors $\lambda^0_{\tilde{p}(i,k)} = 0$ (le compteur ne bouge pas).

La phase CHECK pour ces nouveaux compteurs aux positions $\tilde{p}(i,j)$ est exactement la même que pour les $n(k+1)$ compteurs précédents, aux positions $p(i,j)$.

L'obligation pour tous les compteurs de commencer initialisés à 0 peut elle aussi être oubliée, en faisant deviner à $\mathcal{M}^0$, dès le début de l'exécution, les valeurs correctes de chaque d_i et f_i et en initialisant $2n$ compteurs 1-r.b. supplémentaires en y copiant la valeur des n premiers compteurs. Ceci peut être fait en remettant à 0 chaque compteur x_i, $1 \leq i \leq n$, d'abord en décrémentant de 1 puis en finissant par un décrément de Δ ; si $x_i = 0$, c'est que d_i et f_i ont été devinés correctement, et sinon, on rejette l'exécution. Pour ce faire, tout simplement, un compteur x'_i est incrémenté du montant dont x_i est décrémenté pour atteindre 0. Une fois que $x_i = 0$, le test est passé, et $\mathcal{M}^0$ continue l'exécution comme précédemment, cette fois en utilisant x'_i au lieu de x_i et en commençant dans un état ayant les valeurs d_i et f_i venant d'être devinées, plutôt que depuis $d_i = f_i = 0$.

Enfin, la restriction de ne pas avoir de compteur libre 0-testable dans $\mathcal{M}$ peut facilement être supprimée, en ajoutant un compteur libre 0-testable également à $\mathcal{M}^0$. $\mathcal{M}^0$ peut aussi simuler le comportement de ce compteur dans la phase SIMUL, et le laisser inchangé pendant la phase CHECK ; ceci n'affecte en rien les constructions précédentes. □

Puisque les compteurs reversal-bornés peuvent toujours être transformés en un nombre fini (et plus grand) de compteurs 1-r.b., et en utilisant le lemme 6.9, on peut généraliser la proposition 6.5 au cas des compteurs r.b. :

Théorème 6.10. *Toute k-DCM reversal-bornée peut être encodée par une DCM reversal-bornée, étant donné $k \geq 0$.*

Ce théorème généralise immédiatement le résultat principal de [XDISP03], rappelé ici dans la proposition 6.7.

6.3 Résultats de décidabilité et d'indécidabilité

Le tableau suivant résume les résultats concernant les DCM et leurs variations. Les résultats en caractères ***italique gras*** sont nouveaux, et les autres ont été prouvés (ou sont facilement inférables) d'articles précédents, notamment [XDISP03] et [Min67]. Il y a quatre entrées possibles dans ce tableau : " ?" signifie que l'on ne sait pas si le problème d'accessibilité d'un état de contrôle est décidable, "I" s'il est indécidable, "D" s'il est décidable, et "C" si la relation d'accessibilité est calculable et définissable dans une logique décidable. Les "+ r.b." (respectivement, "+ k-test. r.b.") signifient que la machine est étendue avec un nombre fini de compteurs à choix dense reversal-bornés (respectivement, compteurs à choix dense reversal-bornés et k-testables).

Dans le reste de cette section, on prouve deux de ces nouveaux résultats ; les autres nouveaux résultats sont prouvés dans la section précédente ou en sont directement

compteurs		DCM	k-DCM bornée	DCM + r.b.	DCM + k-test. r.b.
purement à choix dense	1	C	***C***	C	***C***
	2	D	?	?	?
	3	?	?	?	?
	4	I	***I***	I	***I***
à choix dense	1	C	***C***	C	***C***
	2	I	I	I	I

FIG. 6.4: Récapitulatif des résultats de décidabilité sur les DCM

inférables. On remarque que les sept problèmes ouverts pourraient être résolus par seulement deux ou trois preuves qui engloberaient les autres. Cependant, les intuitions pour les cas à 1 ou 4 compteurs ne fonctionnent plus dans le cas de 2 ou 3 compteurs, et les techniques de preuves deviennent encore plus complexes dès qu'on utilise des compteurs k-testables.

Indécidabilité pour les compteurs purement à choix dense et bornés.

Étant donnée une DCM $\mathcal{M}$, un compteur x de $\mathcal{M}$ est *b-borné*, $b \geq 0$, si $x \leq b$ durant toute exécution de $\mathcal{M}$. Par exemple, un compteur 1-borné peut prendre des valeurs non-négatives jusqu'à 1. Un compteur est *borné* s'il est b-borné pour un certain $b \geq 0$.

Étant donné $b \geq 0$, si une machine $\mathcal{M}$ a un compteur à choix dense x qui est b-borné et b-testable, alors on peut supposer que $\mathcal{M}$ doit rejeter une exécution non seulement lorsqu'elle essaye de décrémenter x en-dessous de 0, mais aussi quand elle tente d'arriver dans une configuration où $x > b$. En effet, si x n'était pas borné, on pourrait modifier $\mathcal{M}$ pour tester à chaque étape si $x \leq b$, en rejetant si ce n'est pas le cas (ce qui forcerait x à être borné).

Les compteurs bornés *entiers* ont un ensemble fini de valeurs possibles, que l'on peut encoder dans les états de contrôle. Cependant, les compteurs bornés *à choix dense* ont un ensemble infini de valeurs possibles : une DCM avec plusieurs compteurs bornés est un modèle encore très puissant, comme on le montre dans la suite. En général, le problème d'accessibilité d'un état de contrôle est un problème plus simple que de calculer la relation d'accessibilité. Cependant, la proposition suivante montre que même pour l'accessibilité d'un état de contrôle, il suffit d'avoir quatre compteurs 1-bornés pour devenir indécidable.

Proposition 6.11. *Le problème d'accessibilité d'un état de contrôle est indécidable pour les DCM pures et bornées.*

Démonstration. On montre un résultat plus fort : le problème d'accessibilité d'un état de contrôle est indécidable pour une DCM avec seulement quatre compteurs purement à choix dense qui sont 1-bornés et 1-testables. Ce résultat se démontre assez similairement à celui de [XDISP03], montrant qu'il suffit de quatre compteurs purement à choix dense pour simuler une machine de Minsky.

La preuve d'origine est basée sur l'utilisation de deux compteurs pour mémoriser une valeur fixe de δ, choisie au début de l'exécution. Les deux autres compteurs sont alors incrémentés ou décrémentés de cette unique valeur δ : tout entier e est alors encodé par la valeur $e\delta$. Ainsi, ces deux compteurs se comportent comme deux compteurs discrets, sans restriction. Si les quatre compteurs sont 1-bornés, alors ils peuvent encoder seulement des entiers inférieurs ou égaux à $e' = \lfloor 1/\delta \rfloor$. Cependant, e' n'est pas borné, puisque δ est choisi de façon non-déterministe une seule fois au début de l'exécution, et peut être arbitrairement petit : si δ n'est pas suffisamment petit, la DCM rejettera une exécution qui essaierait de faire croître un compteur au-dessus de 1 (par exemple en revenant à la configuration initiale de sorte que δ puisse être choisi de nouveau, jusqu'à ce qu'il soit assez petit).

En outre, dans une machine de Minsky, dans chaque exécution qui termine, les valeurs encodées dans les deux compteurs sont bornées : cette borne dépend de l'exécution, mais elle existe toujours pour une exécution finie. C'est pourquoi l'état final de la DCM est accessible si et seulement si la machine de Minsky qu'elle simule a une exécution qui termine. Le problème d'accessibilité d'un état de contrôle y est donc indécidable. □

Décidabilité avec un compteur k-testable.

La proposition 6.11 n'exclut pas la décidabilité si l'on utilise moins de quatre compteurs, puisque sa preuve est basée sur une DCM pure à quatre compteurs. En particulier, on montre ici que pour une DCM avec un seul compteur, on peut effectivement calculer sa relation d'accessibilité, et ce même si le compteur est k-testable. Cette extension à la k-testabilité est en effet loin d'être évidente, bien qu'elle puisse paraître triviale de prime abord.

En fait, la construction de la preuve de la proposition 6.5 peut être appliquée à des tests de la forme $x > j$ ou $x = j$, pour tout $j \leq k$; cette construction est simulable par une suite de j décréments entiers, suivie par un test $x > 0$ ou $x = 0$, lui-même suivi d'une séquence de j incréments entiers (afin de restaurer la valeur initiale du compteurs). Cependant, on ne peut pas l'appliquer à des tests de la forme $x < j$, puisque ce cas nécessiterait un compteur supplémentaire (inexistant) afin de restaurer la valeur initiale du compteur. Dans la preuve utilisée ici, le compteur doit de plus être borné pour éviter un nombre non-borné de dépassements du seuil k.

Là encore, on utilise la notion de formule mixte, développée dans la section 6.4. Elles constituent une logique décidable, équivalente à $\mathrm{FO}(\mathbb{R}, \mathbb{Z}, +, <)$. De plus, on sait

grâce à la proposition 6.7 que la relation d'accessibilité d'une DCM est définissable par une formule mixte.

Proposition 6.12. *La relation d'accessibilité d'une DCM à un seul compteur k-testable borné est définissable par une formule mixte, pour tout entier $k \geq 0$.*

Démonstration. Soit $\mathcal{M} = \langle Q, T \rangle$ une DCM à un compteur, telle que son unique compteur est b-borné. Notons que puisqu'il n'y a qu'un seul compteur, on ne manipule ici qu'un réel x, plutôt qu'un ensemble X. On ne s'intéresse qu'au cas où $b = k$, puisque si $b < k$ alors tous les tests à un $j > b$ sont faux, et si $b > k$ alors $\mathcal{M}$ n'utilisera simplement pas les tests à $k+1, k+2, \ldots n$.

Soit $\mathcal{M}' = (Q', T')$ la DCM à tests finis avec un compteur libre et 1-testable, telle que définie dans le lemme 6.8, avec $\ll q, d, f \gg \in Q'$ pour tout $q \in Q, d \in \{0 \ldots k\}, f \in \{0, 1/2\}$.

On affirme que pour tout $x^0, x^1 \in \mathbb{R}_+$ et $q^0, q^1 \in Q$, avec $q_{\text{init}}, q_{\text{final}} \in Q'$,

$$\begin{aligned} \langle q^0, x^0 \rangle \rightsquigarrow_{\mathcal{M}} \langle q^1, x^1 \rangle \text{ si, et seulement si,} \\ \langle \ll q_{\text{init}}, \lfloor x^0 \rfloor, fr(x^0) \gg, x^0 \rangle \rightsquigarrow_{\mathcal{M}'} \langle \ll q_{\text{final}}, \lfloor x^1 \rfloor, fr(x^1) \gg, x^1 \rangle \end{aligned} \tag{6.1}$$

La proposition principale s'ensuit immédiatement, puisque la relation (6.1) est descriptible par une formule mixte et donc décidable.

"Seulement si" : garanti par la condition (2) du lemme 6.8.

"Si" : supposons que la formule (6.1) soit vraie.

On doit alors montrer que $\langle q^0, x^0 \rangle \rightsquigarrow_{\mathcal{M}} \langle q^1, x^1 \rangle$. La condition (3) du lemme 6.8 ne s'applique qu'aux exécutions consistantes ; en général, les exécutions de $\mathcal{M}'$ ne sont pas forcément consistantes. Cependant, chaque incrément fractionnel ou décrément fractionnel dans une exécution de $\mathcal{M}'$ est choisi de façon non-déterministe : la valeur de x peut alors être ajustée pour devenir consistante vis-à-vis de d et f. La preuve de cette affirmation nécessite quelques définitions préliminaires.

Une *version consistante* d'une configuration $c = \langle \ll q, d, f \gg, x \rangle$ est n'importe quelle configuration consistante $c' = \langle \ll q, d, f \gg, x' \rangle$, avec $x' \in \mathbb{R}_+$.

Pour toute configuration consistante $c_0 = \langle \ll q, d, f \gg, x^0 \rangle$ et pour toute configuration $c_1 = \langle \ll q_1, d_1, f_1 \gg, x^1 \rangle$,

$$\begin{aligned} \text{si } c_0 \xrightarrow{g', \lambda, a}_{\mathcal{M}'} c_1, \\ \text{alors il existe une version consistante } c_1' \text{ de } c_1 \text{ telle que } c_0 \xrightarrow{g', \lambda, a}_{\mathcal{M}'} c_1', \\ \text{définie par } c_1' = \langle \ll q_1, d_1, f_1 \gg, x'^1 \rangle, \text{ avec } x' \in \mathbb{R}_+. \end{aligned} \tag{6.2}$$

Si $a = 1$, alors c_1 est déjà consistante, par définition de $\mathcal{M}'$. Ainsi, seul un incrément fractionnel ou décrément fractionnel (i.e. si $a = \Delta$ et $\lambda \neq 0$) pourraient mener à une

configuration inconsistante.

Un cas particulier de configuration est une *zéro-conf*, c'est-à-dire toute configuration de $\mathcal{M}'$ de la forme $\langle \ll q, (0,0)^n \gg, \mathbf{0} \rangle$. Dans une DCM à tests finis $\mathcal{M}'$, on peut supposer que chaque zéro-conf est toujours consistante, puisque $\mathcal{M}'$ peut tester si chaque composante de X vaut bien 0 (et rejeter l'exécution, dans le cas contraire).

Supposons d'abord que $x^0 = 0$, c'est-à-dire que c_0 est une zéro-conf. On a donc $d_1 = d$, $f_1 = 1/2$, et $x = \delta$, avec $0 < \delta < 1$. La configuration c_1 est donc déjà consistante.

Supposons à présent que $x^0 > 0$. Ainsi, d_1 peut différer de d d'au plus 1. Remarquons que pour tout état $\ll q, d, f \gg$, $\ll q', d', f' \gg$ de $\mathcal{M}'$, pour tout $(g, \lambda, a) \in \Sigma$, pour tout $x \in R_+$, et $\forall \epsilon \in [-1, 1]$ avec $0 \leq x + \epsilon < k + 1$, si

$$\langle \ll q, d, f \gg, x \rangle \xrightarrow{g', \lambda, a}_{\mathcal{M}'} \langle \ll q', d', f' \gg, x + \epsilon \rangle,$$

alors pour tout $x' \in R_+$ tel que $0 \leq x' + \epsilon < k + 1$, le même mouvement peut être effectué depuis x' :

$$\langle \ll q, d, f \gg, x' \rangle \xrightarrow{g', \lambda, a}_{\mathcal{M}'} \langle \ll q', d', f' \gg, x' + \epsilon \rangle \tag{6.3}$$

Cette propriété (6.3) est évidente, puisque $\mathcal{M}'$ ne peut tester x qu'à zéro, et ne peut donc pas différencier x de x' avant la transition, et peut donc appliquer le même incrément. De plus, elle implique que l'on peut effectuer le même mouvement depuis c_0 en utilisant un incrément (ou décrément) différent : x peut être augmenté (ou diminué) d'une valeur, plus ou moins grande que δ mais restant dans l'intervalle $]0, 1[$, ce qui suffit pour rendre la configuration consistante.

Soit $c_0 \rightarrow_{\mathcal{M}'} c_1 \rightarrow_{\mathcal{M}'} \cdots \rightarrow_{\mathcal{M}'} c_l$ une exécution de $\mathcal{M}'$, avec $l \geq 0$. On prouve maintenant, par induction sur l, que si c_0 est consistante, alors il existe une autre exécution de $\mathcal{M}'$, notée $c_0 \rightarrow_{\mathcal{M}'} c'_1 \rightarrow_{\mathcal{M}'} \cdots \rightarrow_{\mathcal{M}'} c'_l$, dans laquelle chaque c'_i est une version consistante de c_i, pour $1 \leq i \leq l$.

Le cas $l = 0$ est trivial, car $c_0 = c'_0$. Supposons $l > 0$. Par hypothèse d'induction $c_0 \rightarrow_{\mathcal{M}'} c'_1 \rightarrow_{\mathcal{M}'} \cdots \rightarrow_{\mathcal{M}'} c'_{l-1}$, chaque c'_i est une version consistante de c_i, pour $1 \leq i \leq l - 1$. D'après la propriété (6.2), on peut trouver une version consistante c'_l de c_l telle que $c'_{l-1} \rightarrow_{\mathcal{M}'} c'_l$.

D'après la condition (3) du lemme 6.8, toute exécution consistante de $\mathcal{M}'$ est également une exécution de $\mathcal{M}$, ce qui termine la preuve. □

Cette preuve est immédiatement extensible au cas où $\mathcal{M}$ a aussi des compteurs reversal-bornés *discrets*, qui ne sont en aucun cas affectés par la construction utilisée.

Si les compteurs reversal-bornés supplémentaires sont à choix dense, cependant, la décidabilité reste ouverte. Ce résultat nécessiterait une nouvelle preuve, parce que les techniques utilisées pour démontrer la proposition 6.12 et le lemme 6.9 ne semblent pas pouvoir se combiner.

6.4 Caractérisation logique des DCM

Quelques rappels sur les formules mixtes.

On considère ici le langage des formules mixtes, définies dans [Wei99] et adapté de l'arithmétique de Presburger. Ce langage a deux sortes de variables : des variables réelles, notées $x, x', x_1, \ldots$, et des variables entières, notées $y, y', y_1, \ldots$; les variables entières sont donc une sous-sorte des variables réelles. Les constantes sont 0 et 1, les opérations sont $+$, $-$, $\lfloor . \rfloor$, et les relations sont l'égalité $=$, l'ordre $<$, et les congruences $\equiv_d$ pour toute constante $d \in \mathbb{N}$. La définition suivante formalise cet énoncé :

Définition 6.13. *Une* fomule mixte *est définie inductivement comme suit. Une* expression linéaire mixte E *est définie par la grammaire suivante, où x est une variable réelle et y est une variable entière :*

$$E ::= 0 \mid 1 \mid x \mid y \mid E + E \mid E - E \mid \lfloor E \rfloor$$

Une contrainte linéaire mixte C *est définie par la grammaire suivante, où d est un entier positif :*

$$C ::= E = E | E < E | E \equiv_d E$$

Une formule mixte F *est définie par la grammaire suivante, où $x \in \mathbb{R}$ et $y \in \mathbb{Z}$:*

$$F ::= C \mid \neg F \mid F \wedge F \mid \exists x.F \mid \exists y.F$$

La sémantique d'une formule mixte est la même que dans les réels, $\lfloor r \rfloor$ étant la partie entière de son argument réel r, et $r_1 \equiv_d r_2$ si et seulement si $r_1 - r_2 = vd$ pour un entier v.

Typiquement, on utilise des abbréviations, comme par exemple $3x$ pour $x + x + x$, ou en ajoutant des opérateurs communs (comme $\geq$, $\leq$, etc.).

Les formules mixtes sont équivalentes à la célèbre logique additive du premier ordre sur les entiers et réels, notée $\mathrm{FO}(\mathbb{R}, \mathbb{Z}, +, <)$; cette équivalence se remarque aisément, puisque $\lfloor x \rfloor = y$ peut être récrit $\exists x_1 (0 < x_1 \wedge x_1 < 1 \wedge x - x_1 = y)$, et $x_1 \equiv_d x_2$ peut s'écrire $\exists y (x_1 - x_2 = \underbrace{y + \cdots + y}_{d})$, pour un $d > 0$ donné. Cependant, l'avantage principal de la syntaxe plus riche des formules mixtes est qu'elle permet une élimination de quantificateurs, ce qui n'est pas possible directement dans $\mathrm{FO}(\mathbb{R}, \mathbb{Z}, +, <)$, comme le montrent le théorème 3.1 et le corollaire 5.2 de [Wei99].

6.4.1 Formules mixtes et DCM reversal-bornées

On sait que les machines à compteurs discrètes reversal-bornées peuvent définir n'importe quelle formule de Presburger. Puisque la logique de Presburger admet une élimination de quantificateurs effective, la relation d'accessibilité d'une machine à compteurs discrète reversal-bornée peut effectivement définir toute relation de Presburger. On montre ici un résultat similaire, cette fois pour les DCM reversal-bornées, en utilisant les formules mixtes (et l'effectivité de l'élimination de quantificateurs) au lieu des formules de Presburger.

Soit $\mathbf{0}$ le vecteur $(0, \ldots, 0)$ de taille n. Une formule mixte sans quantificateurs $F(z_1, \ldots, z_n)$ avec les variables libres $z_1 \geq 0, \ldots, z_n \geq 0$ est *définissable* par une DCM $\mathcal{M}$ avec au moins n compteurs $x_1, \ldots, x_n$ si $\mathcal{M}$, débutant dans une configuration initiale donnée $\langle q, \mathbf{0}\rangle$, peut atteindre exactement les configurations finales $\langle q_{fin}, X\rangle$ telles que $F(x_1/z_1, \ldots, x_n/z_n)$ est vraie (où x_i/z_i est une substitution de la variable z_i par la valeur x_i).

Proposition 6.14. *Soit $F(x_1, \ldots, x_n, y_1, \ldots, y_p)$ une formule mixte sans quantificateurs, avec $x_1 \geq 0, \ldots, x_n \geq 0, y_1 \geq 0, \ldots, y_p \geq 0$. Alors F est définissable par une DCM reversal-bornée.*

La preuve, détaillée ci-après, fait appel à plusieurs étapes, qui peuvent se comprendre assez facilement. Tout d'abord, on suppose (sans perte de généralité) que F est en forme normale disjonctive ; on la transforme alors en une union d'intersections de formules plus petites, chacune de la forme $E \sim 0$, avec $\sim \in \{>, =, <, \equiv_d, \not\equiv_d\}$. L'idée principale est d'encoder chacune de ces petites formules par une DCM reversal-bornée, dans laquelle il y a n compteurs x qui sont reversal-bornés et à choix dense, p compteurs y qui sont reversal-bornés et discrets, et, si besoin, d'autres compteurs reversal-bornés. On fournit un encodage simple de chacune de ces formules $E \sim 0$ en une DCM reversal-bornée, qui accepte une exécution si et seulement si le fait d'affecter les valuations initiales de compteurs aux variables libres de F la rendent vraie.

Il suffit ensuite de connecter chaque DCM construite comme suit. Chaque DCM possède un état final, que l'on relie par une transition à l'état initial d'une autre DCM ; ces deux DCM forment alors une nouvelle DCM, plus grosse. Pour l'union, on ajoute une transition de l'état final de la première DCM vers l'état initial de chaque DCM encodant une des composantes de l'union (i.e. une intersection de $E \sim 0$). On retrouve donc une DCM dont la sortie est reliée à des suites de DCM, chacune de ces suites encodant une intersection de $E \sim 0$. La dernière composante de chacune de ces intersections est encodée par une DCM dont l'état final mène à un état-puits, qui est l'état final de la DCM globale encodant F. Cet état-puits est accessible si et seulement si F est satisfiable.

Démonstration. Supposons, sans perdre en généralité, que F est en forme normale disjonctive :

$$F = F_1 \vee F_2 \vee \ldots \vee F_m$$

F est donc une disjonction de clauses F_i de la forme :

$$F_i = F_{i_1} \wedge F_{i_2} \wedge \cdots \wedge F_{i_{m_i}}$$

où chaque F_{ij}, dans les variables libres $x_1, \ldots, x_n, y_1, \ldots, y_p$, peut toujours être réduite, en poussant les négations au niveau du symbole relationnel et en utilisant des transformations algébriques élémentaires, à la forme :

$$E \sim 0$$

où $\sim \in \{>, =, <, \equiv_d, \not\equiv_d\}$. Par exemple, si F_{i_j} est $E_1 < E_2$, alors on peut vérifier à la place si $E_1 - E_2 < 0$, etc.

On montre à présent que pour chaque F_{i_j}, il existe une DCM reversal-bornée $\mathcal{M}_{i_j}$, avec des compteurs $x_1, \ldots, x_n$ reversal-bornés et à choix dense, des compteurs $y_1, \ldots, y_p$ reversal-bornés et discrets, et (si cela s'avère nécessaire) d'autres compteurs reversal-bornés $x_{n+1}, \ldots$ et $y_{p+1}, \ldots$, acceptant lorsque la relation F_{i_j} est vérifiée par les valeurs initiales des compteurs $x_1, \ldots, x_n, y_1, \ldots, y_p$.

Ceci entraîne immédiatement que, pour toute clause F_i, il existe une DCM reversal-bornée $\mathcal{M}_i$ qui accepte si $F_{i_1} \wedge F_{i_2} \wedge F_{i_{m_i}}$ est vérifiée par les valeurs initiales de ses compteurs $x_1, \ldots, x_n, y_1, \ldots, y_p$. En fait, puisque F_i est la conjonction de tous les F_{i_j}, $\mathcal{M}_i$ est une DCM reversal-bornée qui commence par faire m_i copies des compteurs $x_1, \ldots, x_n, y_1, \ldots, y_p$, puis qui simule chaque $\mathcal{M}_{i_j}$ en commençant sur chacune de ces copies. $\mathcal{M}_i$ accepte si et seulement si toutes les $\mathcal{M}_{i_j}$ acceptent.

Ainsi, on peut construire une DCM reversal-bornée $\mathcal{M}'$ dont la relation d'accessibilité décrit la relation F : tout d'abord, $\mathcal{M}'$ commence avec tous ses compteurs à zéro. Ensuite, elle effectue des incréments non-déterministes sur chaque compteur, pour deviner des valeurs pour $x_1, \ldots, x_n, y_1, \ldots, y_p$ telles qu'au moins un des F_i est vrai (et donc, F est aussi vraie). Enfin, elle suit l'exécution de $\mathcal{M}_i$ comme décrit précédemment, afin de vérifier que les valeurs devinées sont correctes. Afin de terminer dans une configuration dans laquelle les n premiers compteurs contiennent les valeurs des n variables rendant F vraie, on travaillera sur des copies de ces compteurs.

Afin de montrer que pour tout i, j il existe une DCM reversal-bornée $\mathcal{M}_{i_j}$ définissant F_{i_j}, on commence par montrer, par induction sur la structure de E, que la valeur de E peut être encodée par un compteur reversal-borné à choix dense (contenant la valeur de $|E|$) et un bit de signe ajouté à l'état de contrôle (indiquant le signe de E). On rappelle qu'une valeur de compteur peut toujours être copiée un nombre fixe de fois, en utilisant

l'encodage de la figure 6.2b.

Les cas de base de l'induction sont les cas où E vaut 0, 1, x_i, ou y_j, et sont triviaux.

Supposons à présent que E est de la forme $(E_1 + E_2)$, avec une copie de $|E_1|$ et $|E_2|$ stockée dans deux compteurs idoines reversal-bornés, avec leur signes encodés dans l'état de contrôle. On peut supposer que E_1 et E_2 ont le même signe, puisque si $E_1 \geq 0$ et $E_2 < 0$ alors $E_1 + E_2$ peut s'écrire $E_1 - |E_2|$ (et réciproquement). Il suffit de s'intéresser au cas où E_1 et E_2 sont positifs, puisque s'ils sont tous les deux négatifs il suffira de donner un bit de signe négatif au résultat. L'addition $E_1 + E_2$ peut alors être réalisée en utilisant l'encodage de la figure 6.2c.

Supposons maintenant que E est de la forme $(E_1 - E_2)$, toujours avec des copies de $|E_1|$ et $|E_2|$ ainsi que leur bit de signe. Il suffit de considérer le cas où E_1 et E_2 sont positifs ; les autres cas se ramènent en effet à une addition, comme décrite ci-dessus. On peut également supposer que $E_1 \geq E_2$; si ce n'est pas le cas, i.e. si $E_2 > E_1$, alors la DCM le devinera et calculera $E_2 - E_1$ à la place, en changeant le signe du résultat. Notons que si la DCM se trompe en devinant que $E_1 \geq E_2$ alors qu'en fait $E_2 < E_1$, elle rejettera l'exécution ; ce choix non-déterministe ne peut donc être que correct, si l'exécution est acceptée. Le calcul de $E_1 - E_2$ peut alors être effectué en utilisant l'encodage de la figure 6.2d.

Supposons enfin que E est de la forme $\lfloor E' \rfloor$; alors on peut utiliser l'encodage de la figure 6.5, dans lequel la DCM atteint q' depuis q si et seulement si $x_1 = \lfloor x \rfloor$.

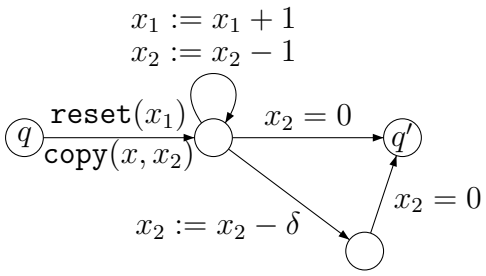


FIG. 6.5: Stockage de $\lfloor x \rfloor$ dans x_1

On vient de montrer comment encoder une expression linéaire mixte E dans une DCM reversal-bornée. Afin de terminer la preuve qu'il existe toujours une DCM reversal-bornée $\mathcal{M}_{i_j}$ définissant F_{i_j}, il suffit de montrer qu'il existe une DCM reversal-bornée $\mathcal{M}$ qui vérifie si $E \sim 0$ (puisque F_{i_j} est sous cette forme).

Puisque la valeur de $|E|$ est stockée dans un compteur reversal-borné à choix dense x, avec un bit de signe dans l'état de contrôle indiquant le signe de E, alors on peut immédiatement tester si $E < 0$ ou $E > 0$. Le cas $E = 0$ est lui aussi trivial, puisque

l'on dispose de gardes $(x = 0)$. Les deux cas restants sont donc les congruences modulo un entier d. On peut alors utiliser l'encodage de la figure 6.6, dont la DCM peut atteindre q' depuis q si et seulement si $E \equiv_d 0$, pour un entier d donné.

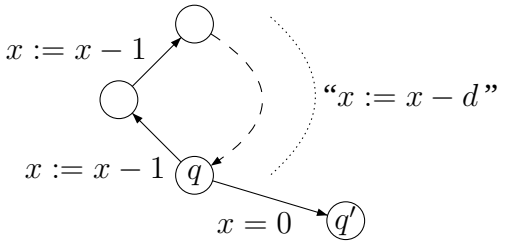


FIG. 6.6: Test de $x \equiv_d 0$

Pour vérifier si $E \not\equiv_d 0$, $\mathcal{M}$ commence par vérifier si $x - \lfloor x \rfloor > 0$, en acceptant si c'est le cas. Si $x - \lfloor x \rfloor = 0$, alors $\mathcal{M}$ devine la constante entière $v \in [0, d]$ telle que $E - v \equiv_d 0$. Ceci peut être effectué comme expliqué ci-dessus.

On a donc donné une preuve constructive qu'il existe une DCM reversal-bornée définissant une formule mixte donnée, sans quantificateurs. □

Puisque l'élimination de quantificateurs dans les formules mixtes est effective, et puisque l'on peut encoder des variables négatives en utilisant un bit de signe dans l'état de contrôle, on peut alors déduire le théorème suivant :

Théorème 6.15. *Toute formule mixte peut être définie par une DCM reversal-bornée.*

Ce théorème est en fait le dual de la proposition 6.7 (issue de [XDISP03]), énonçant que la relation d'accessibilité d'une DCM reversal-bornée (avec un compteur libre supplémentaire) est définissable par une formule mixte. On obtient ainsi une caractérisation exacte des DCM reversal-bornées, et on montre que le problème 1.10 est décidable pour les DCM reversal-bornées et les formules mixtes.

En fait, le théorème 6.15 peut être combiné avec d'autres résultats sur les formules mixtes. Par exemple, on sait que la relation d'accessibilité d'un système à compteurs relationnel plat [CJ98] ou d'un automate temporisé [Dan03] est définissable par une formule mixte. Ainsi, on peut construire une DCM reversal-bornée qui définit exactement la relation d'accessibilité d'un système à compteurs relationnel plat donné ou d'un automate temporisé donné.

6.4.2 DCM et systèmes à compteurs relationnels

Ce modèle de DCM peut sembler exotique : il est en effet syntaxiquement très différent des systèmes à compteurs habituels, notamment à cause de cette notion de

choix non-déterministe du δ. Cependant, on a récemment remarqué qu'une DCM s'apparente fortement aux systèmes à compteurs relationnels de Comon-Jurski (voir section 4.3). En effet, dans le cas $n = 1$ (i.e. un seul compteur), toute k-DCM (et a fortiori, toute DCM) peut s'écrire comme un système à compteurs relationnel (voir définition 4.6) : on donne ici un encodage des instructions des DCM en formules de $\mathscr{L}_{CJ}$. On rappelle que $\mathscr{L}_{CJ}$ est la logique composé de conjonctions finies $F(X, X') = \bigwedge(y \bowtie z + k)$, où $k \in \mathcal{D}$, $\bowtie \in \{\geq, >, =, <, \leq\}$, $y \in X \cup X'$, et $z \in X \cup X' \cup \{0\}$, et que c'est la logique qui étiquette les transitions d'un système à compteurs relationnel.

Les opérations d'une k-DCM sont de la forme $x' = x + 1$, $x' = x - 1$, $x' = x + \delta$, $x' = x - \delta$; ses gardes sont de la forme $x > i$, $x = i$, $x < i$, $true$, pour tout $i \in [0, k]$. Les gardes sont directement expressibles dans $\mathscr{L}_{CJ}$, sans aucune modification, ainsi que les opérations $x' = x + 1$ et $x' = x - 1$. Il ne reste donc que les opérations $x' = x + \delta$ et $x' = x + \delta$ à encoder : c'est là qu'il suffit de remarquer que les deux modèles offrent un choix non-déterministe dans les formules étiquetant les transitions. En effet, l'opération $x' = x + \delta$ d'une DCM peut s'écrire $x' > x \wedge x' < x + 1$, et $x' = x - \delta$ peut s'écrire $x' < x \wedge x' > x - 1$, qui sont toutes les deux des formules de $\mathscr{L}_{CJ}$, et donc des formules étiquetant les transitions d'un système à compteurs relationnel. Il faut bien comprendre que lors d'une exécution d'un système à compteurs relationnel, chaque franchissement d'une transition ne donne qu'une seule valeur bien précise à chaque compteur ; les formules de $\mathscr{L}_{CJ}$ définissent bien des ensembles de valeurs possibles, mais une exécution n'en choisira qu'une, et ce, de façon non-déterministe, tout comme le choix de la valeur d'un δ dans une DCM !

On vient donc de montrer la proposition suivante, signifiant que les k-DCM à un seul compteur ne sont qu'un cas particulier des système à compteurs relationnels :

Proposition 6.16. *Pour toute k-DCM $\mathcal{M}$ à un seul compteur, il existe un système à compteurs relationnel bisimilaire à $\mathcal{M}$.*

Cependant, dans le cas à plusieurs compteurs ($n > 1$), cet encodage ne suffit plus. En effet, on a la possibilité de contraindre le choix non-déterministe dans les DCM, en spécifiant que deux compteurs doivent être modifiés d'une *même* valeur δ, sur une même transition. Par exemple, une formule étiquetant une transition d'une DCM peut avoir la forme $\psi \equiv x'_1 = x_1 + \delta \wedge x'_2 = x_2 + \delta$: cette formule ne semble pas définissable dans $\mathscr{L}_{CJ}$. Elle ne l'est vraisemblablement pas, puisque le non-déterminisme d'un système à compteurs relationnel n'est apparemment pas influençable. On pourrait imaginer une simulation d'une DCM par un système à compteurs relationnel, qui n'accepterait que les exécutions ayant choisi le même incrément/décrément sur les transitions encodant ψ ; mais il faudrait pouvoir tester que les incréments/décréments ont été les mêmes, ce qui ne semble pas possible dans $\mathscr{L}_{CJ}$.

On suppose donc que les DCM et les systèmes à compteurs relationnels sont incomparables, dans le cas général.

Conclusion relative aux DCM

On a donc éclairé les DCM d'une nouvelle lumière, plus formelle, en clarifiant leurs relations avec les systèmes à compteurs classiques (discrets). On remarque alors que certains résultats très simples des systèmes à compteurs classiques restent valables pour les DCM, mais nécessitent des preuves beaucoup plus complexes.

Une première extension des DCM est la k-testabilié, consistant à permettre aux compteurs à choix dense d'être comparés à une constante entière k plus grande que 0. On montre que les compteurs à choix dense ne sont pas plus puissants lorsqu'ils sont k-testables, même dans le cas des DCM reversal-bornés ou des DCM avec un seul compteur borné.

Une autre extension est la caractérisation exacte des DCM reversal-bornées par la fameuse logique additive du premier ordre sur les entiers et réels, de façon similaire à la caractérisation des machines à compteurs reversal-bornées par la logique de Presburger.

On trouve également des résultats ne pouvant pas être étendus des systèmes à compteurs classiques aux DCM. Par exemple, le fait de contraindre les compteurs à choix dense à être bornés ne garantit aucunement la décidabilité de ce modèle.

Modèles de systèmes numériques : conclusion

On a montré que l'accélération dans les systèmes à compteurs de FAST s'étend directement au cas où les compteurs sont mixtes (et non pas seulement entiers). On a ensuite défini un modèle de systèmes à compteurs entiers et à horloges réelles, les TCS, en montrant comment on peut les analyser en se servant de techniques existantes (constructions de régions d'horloges et restriction à des sous-classes pour les compteurs).

On a de plus effectué un travail de clarification, extension, et motivation des DCM, en prolongeant leurs résultats de décidabilité et en fournissant une caractérisation de leur sous-classe reversal-bornée par logique additive mixte. On a également prouvé que les DCM à un compteur sont un cas particulier des systèmes à compteurs relationnels de Comon-Jurski, ce qui relie clairement le modèle "exotique" que sont les DCM à une classe de systèmes bien connue. Il semble cependant que dans le cas général (plusieurs compteurs), ces deux modèles restent incomparables.

Conclusions et perspectives

Bilan de cette thèse

En somme, cette thèse propose une étude des systèmes infinis dans l'optique de leur vérification, essentiellement par model-checking. Les systèmes infinis considérés ici diffèrent selon la logique qui définit les formules étiquetant leurs transitions. Ces différences peuvent être causées par la richesse de cette logique, mais aussi par divers facteurs : le domaine de définition des variables (entier, réel, rationnel), la densité de la sémantique de progression des variables (discrète, continue, hybride), la nature des variables (compteurs, horloges, mixtes), ou encore la présence de non-déterminisme dans la mise à jour des variables.

Comme on l'a vu plusieurs fois au cours de cette thèse, on a besoin de modèles toujours plus puissants, qui restent cependant analysables. À cette fin, on a proposé deux modèles de systèmes parmi les plus expressifs qui soient, en montrant que l'on peut décider leur problème d'accessibilité (voire calculer leur relation d'accessibilité) lorsque l'on se restreint à certaines sous-classes que l'on a identifiées.

Afin d'être en mesure de comparer ces systèmes et de les analyser, on a vu que l'on avait besoin de logiques arithmétiques décidables. Puisque les variables qui nous intéressent sont mixtes, on a considéré des logiques, elles aussi, mixtes. Outre l'intérêt purement théorique de l'étude comparative de ces logiques, celles-ci nous ont servi à caractériser la relation d'accessibilité des DCM reversal-bornées, ainsi qu'à prolonger la théorie de l'accélération de systèmes à compteurs au cas réel (ou mixte).

Notre étude des logiques a également permis de mettre en avant une décomposition des logiques mixtes, en partant d'une idée simple mais intéressante. Cette idée est que pour représenter des ensembles infinis de réels, on peut tirer profit du fait que chaque réel est en fait la somme d'un entier et d'un décimal. L'avantage des entiers est qu'ils permettent d'exprimer une périodicité, et l'avantage des décimaux est qu'ils permettent de définir précisément les ensembles qui seront répétés périodiquement. On retrouve alors le principe de la représentation "base-période", qui consiste à répéter un motif à l'infini, selon un certaine régularité.

On a également implémenté un prototype permettant d'utiliser cette représentation d'ensembles réels (ou mixtes) en pratique. L'intérêt de cette implémentation est triple : premièrement, elle propose un solveur pour des logiques mixtes d'expressivités diverses, selon les plugins GENEPI utilisés pour les parties entière et décimale. Deuxièmement, elle permet de manipuler des ensembles de vecteurs de variables mixtes, ce qui est indispensable pour tout outil de model-checking. Enfin, cette implémentation permet d'étendre l'expressivité de solveurs existants, en les combinant à d'autres solveurs au moyen de plugins GENEPI ; ceci est particulièrement intéressant pour les développeurs

de nouveaux solveurs, qui peuvent alors moduler l'expressivité et comparer l'efficacité de leurs implémentations.

Bien que tous ces résultats se placent dans le cadre du model-checking, l'étude de ces logiques et de ces modèles peut avoir d'autres applications dans divers domaines. En effet, les travaux dans la première partie s'inscrivent également dans le cadre de la logique mathématique, qui possède une motivation théorique intrinsèque. Concernant les modèles de systèmes infinis, l'étude que nous en avons proposée peut intéresser d'autres communautés, comme la simulation, la modélisation, ou encore d'autres techniques de vérification.

Travaux futurs

Dans le monde de la recherche, on peut généralement dire que plus on avance, plus on prend conscience de l'étendue de ce que l'on ignore... Les travaux réalisés dans cette thèse confirment cette idée : à chaque résultat obtenu ici, on pourrait associer une myriade de nouveaux problèmes à résoudre.

Pour terminer cette conclusion, on présente une sélection de problèmes ouverts qui ont apparu au fur et à mesure de nos avancées. Quelques pistes à étudier, plus générales, sont également proposées ici. On groupe ces perspectives en s'inspirant du plan de la thèse.

Logiques du premier ordre. Afin de mieux comprendre les logiques réelles (ou mixtes) qui n'apparaissent pas dans cette thèse, il serait intéressant de leur appliquer l'opérateur de décomposition : on pourrait ainsi obtenir de nouvelles caractérisations, établir de nouveaux liens entre diverses logiques, et obtenir de nouveaux résultats de décidabilité.

Complexité de PDM et RDM. Les logiques PDM et RDM ont été définies dans cette thèse, mais on ne s'est pas intéressé à leur complexité. On a en effet montré qu'elles sont toutes deux décidables, mais il reste à trouver des bornes inférieures et supérieures de la complexité de leur satisfaisabilité.

Prototype IDS. D'après les quelques tests réalisés sur l'outil IDS, construire et manipuler des ensembles avec ce prototype prend plus de temps qu'avec LIRA. Il semble que la représentation par automate binaire ait du mal à manipuler des gros coefficients dans

les contraintes linéaires de départ. En effet, résoudre simplement $1000x = 0$ nécessite trop de mémoire pour IDS alors que LIRA y parvient en au plus quelques secondes ; par contre, dès que l'on remplace cette égalité par une inégalité, les deux outils saturent la mémoire vive de l'ordinateur.

De plus, après avoir effectué quelques essais sur une étude de cas se ramenant à un système de contraintes à résoudre, on a pu identifier un problème dans le prototype IDS. Ce problème vient de la représentation par IDF, qui vise à normaliser les ensembles manipulés pour que leur représentation soit unique.

Techniquement, on construit les ensembles incrémentalement sous forme d'unions finies de sommes "entiers + décimaux", en ajoutant ces sommes à l'union, une par une. D'après les statistiques données par GENEPI pour le temps d'exécution moyen de chaque opération, ainsi que le nombre de fois qu'elles sont utilisées, on se rend compte que les unions sont toujours trop sollicitées et prennent trop de temps à s'exécuter.

Il faudrait donc revoir la méthode de construction des ensembles, quitte à ne pas avoir une forme normale unique basée sur une partition des décimaux (i.e. les IDF, qui sont des fonctions injectives des entiers dans les décimaux).

Accélération mixte. On a montré que le théorème principal de l'accélération implémentée dans FAST s'étend aux cas réel et mixte. Il faudrait à présent poursuivre cette extension de FAST, notamment en s'intéressant à la sélection automatique des circuits à accélérer. L'objectif idéal serait alors de se baser sur la plateforme TAPAS, en utilisant FASTer (la version améliorée de FAST adaptée à GENEPI) couplé à IDS.

Systèmes hétérogènes temporisés. Les résultats obtenus sur les TCS sont en fait extensibles à d'autres types de données que les compteurs. En collaboration avec Sébastien Bardin, on a commencé à utiliser ses travaux sur les systèmes hétérogènes [BF04, Bar05] afin de les combiner avec notre approche sur les TCS [BFS09]. Le modèle de systèmes hétérogènes temporisés, défini pour l'occasion, s'appelle THS (pour "Timed Heterogeneous Systems"). On cherche à montrer que le model-checking de propriétés de sûreté exprimées dans la logique temporelle CTL* ou MITL est décidable sur les THS.

Caractérisation des DCM par PDM. On a caractérisé les DCM reversal-bornées par la logique additive mixte. Par analogie, on pense que la relation d'accessibilité des DCM, en général (i.e. pas nécessairement reversal-bornées), est définissable dans PDM. Intuitivement, les incréments d'une DCM sont naturellement des décimaux, et les seules opérations intervenant sur les transitions sont des additions. Il reste cependant à trouver un moyen de gérer les exécutions qui ne sont pas reversal-bornées (sinon, on pourrait reprendre directement la preuve de la proposition 6.14) : l'idée est alors de bénéficier

de la multiplication décimale. Cependant, il faudrait peut-être se restreindre aux DCM pures, puisque la multiplication entière est interdite.

Toutefois, cette caractérisation ne serait pas complète, dans le sens où il est peu probable que toute formule de PDM soit définissable (au sens utilisé dans la section 6.4.1) par une DCM, et encore moins par une DCM pure.

DCM et (in)décidabilité. Il reste plusieurs résultats de décidabilité inconnus sur les DCM, comment en témoignent les points d'interrogation dans le tableau (figure 6.4). Ces résultats semblent difficiles à obtenir, et les techniques de preuves utilisées sur les résultats similaires ne fonctionnent pas pour les cas qui ne sont pas encore résolus.

Dense-choice Petri Nets. Il serait probablement intéressant d'étudier le modèle des "Dense-choice Petri Nets", c'est-à-dire des DCM dans lesquelles on interdit les tests de la forme $(x = 0)$. En effet, on obtiendrait alors peut-être des résultats de décidabilité pour des cas où les DCM sont indécidables.

DCM à 1 compteur. Les systèmes à compteurs classiques (i.e. discrets) sont encore très étudiés ; en particulier, le cas où l'on se restreint à un seul compteur intéresse encore des chercheurs aujourd'hui [HKOW09]. Le fait de n'autoriser qu'un compteur vient du fait que dès que l'on a deux compteurs, on obtient un machine de Minsky, ce qui nous condamne à l'indécidabilité. Les DCM sont un modèle assez récent et très peu étudié (seulement 3 publications à ce jour, dont la nôtre). On sait que les DCM à un seul compteur sont décidables, tout comme les systèmes à compteurs discrets ; on pourrait alors tenter d'étendre les autres résultats (complexité, couverture, etc.) de ces systèmes discrets vers les DCM à un compteur.

Systèmes plats. Les systèmes (à compteurs) plats ou aplatissables apparaissent souvent comme une sous-classe ayant des propriétés intéressantes (décidabilité, accélération, etc.). Comme le montre [LS05], cette sous-classe est assez naturelle et répandue. Il serait alors intéressant d'étudier les TCS plats ; il semblerait que le graphe de régions d'un TCS plat donne un système à compteurs aplatissable, qui serait alors accélérable dans FAST. De même, on pourrait s'intéresser aux DCM plates (ou aplatissables), en espérant obtenir de meilleurs résultats de décidabilité, voire des propriétés menant à l'accélération.

Langages. Dans l'étude des modèles de cette thèse, on ne s'est presque jamais intéressé aux langages. Étiqueter les transitions d'un système par des symboles permet en effet de pouvoir étudier ces modèles en profondeur, et notamment de les comparer entre eux, ou encore de leur appliquer un algorithme de model-checking. Les deux modèles de systèmes définis dans cette thèse (TCS et DCM) sont encore récents, et on ne sait rien

de leurs propriétés en termes de langages. Dans l'optique d'améliorer les comparaisons d'expressivité de ces modèles, ou de les utiliser pour le model-checking, il sera nécessaire de s'intéresser à leurs langages.

Bibliographie

[AAB00] Aurore Annichini, Eugene Asarin, and Ahmed Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, volume 1855 of *Lecture Notes in Computer Science*, pages 419–434. Springer, 2000.

[ABS01] Aurore Annichini, Ahmed Bouajjani, and Mihaela Sighireanu. TReX : A tool for reachability analysis of complex systems. In *Computer Aided Verification, 13th International Conference, CAV 2001, Paris, France, July 18-22, 2001, Proceedings*, volume 2102 of *Lecture Notes in Computer Science*, pages 368–372. Springer, 2001.

[ACH+95] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1) :3–34, 1995.

[AD90] Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In Mike Paterson, editor, *Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, July 16-20, 1990, Proceedings*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990.

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

[AFH99] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata : A determinizable class of timed automata. *Theoretical Computer Science*, 211(1-2) :253–273, 1999.

[AMM07] Parosh Aziz Abdulla, Pritha Mahata, and Richard Mayr. Dense-timed Petri nets : Checking zenoness, token liveness and boundedness. *Logical Methods in Computer Science*, 3(1), 2007.

[AY07] Jeremy Avigad and Yimu Yin. Quantifier elimination for the reals with a predicate for the powers of two. *Theoretical Computer Science*, 370(1-3) :48–59, 2007.

[Bar05] Sébastien Bardin. *Vers un model checking avec accélération plate de systèmes hétérogènes*. Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, October 2005.

[BBF+01] Béatrice Bérard, Michel Bidoit, Alain Finkel, François Laroussinie, Antoine Petit, Laure Petrucci, and Philippe Schnoebelen. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001.

[BBFL03] Gerd Behrmann, Patricia Bouyer, Emmanuel Fleury, and Kim Guldstrand Larsen. Static guard analysis in timed automata verification. In Hubert Garavel and John Hatcliff, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 9th International Conference, TACAS 2003, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7-11, 2003, Proceedings*, volume 2619 of *Lecture Notes in Computer Science*, pages 254–277. Springer, 2003.

[BBL06] Gerd Behrmann, Patricia Bouyer, and Kim Guldstrand Larsen. Lower and upper bounds in zone-based abstractions of timed automata. *International Journal on Software Tools for Technology Transfer (STTT)*, 8(3) :204–215, 2006.

[BBL09] Bernard Boigelot, Julien Brusten, and Jérôme Leroux. A generalization of semenov's theorem to automata over real numbers. In Renate A. Schmidt, editor, *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, QC, Canada, August 2-7, 2009. Proceedings*, volume 5663 of *Lecture Notes in Computer Science*, pages 469–484. Springer, 2009.

[BBR97] Bernard Boigelot, Louis Bronne, and Stéphane Rassart. An improved reachability analysis method for strongly linear hybrid systems (extended abstract). In Orna Grumberg, editor, *Computer Aided Verification, 9th International Conference, CAV '97, Haifa, Israel, June 22-25, 1997, Proceedings*, volume 1254 of *Lecture Notes in Computer Science*, pages 167–178. Springer, 1997.

[BCGH07] Olivier Bournez, Manuel L. Campagnolo, Daniel S. Graça, and Emmanuel Hainry. Polynomial differential equations compute all real computable functions on computable compact intervals. *Journal of Complexity*, 23(3) :317 – 335, 2007.

[BCH+05] Béatrice Bérard, Franck Cassez, Serge Haddad, Didier Lime, and Olivier H. Roux. Comparison of different semantics for time Petri nets. In Doron A. Peled and Yih-Kuen Tsay, editors, *Proceedings of the*

3rd International Symposium on Automated Technology for Verification and Analysis (ATVA'05), volume 3707 of *Lecture Notes in Computer Science*, pages 293–307, Taipei, Taiwan, October 2005. Springer.

[BDEK07] Bernd Becker, Christian Dax, Jochen Eisinger, and Felix Klaedtke. LIRA : Handling constraints of linear arithmetics over the integers and the reals. In Werner Damm and Holger Hermanns, editors, *Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings*, volume 4590 of *Lecture Notes in Computer Science*, pages 307–310. Springer, 2007.

[BDGP98] Béatrice Bérard, Volker Diekert, Paul Gastin, and Antoine Petit. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36(2) :145–182, November 1998.

[BDM+98] Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. Kronos : A model-checking tool for real-time systems. In *Computer Aided Verification, 10th International Conference, CAV '98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings*, volume 1427 of *Lecture Notes in Computer Science*, pages 546–550. Springer, 1998.

[Bér95] Béatrice Bérard. Untiming timed languages. *Information Processing Letters*, 55(3) :129–135, 1995.

[Bès02] Alexis Bès. A Survey of Arithmetical Definability. In *A tribute to Maurice Boffa, Special issue of the Bulletin of the Belgian Mathematical Society*, pages 1–54. Société Mathématique de Belgique, 2002.

[BF04] Sébastien Bardin and Alain Finkel. Composition of accelerations to verify infinite heterogeneous systems. In Farn Wang, editor, *Automated Technology for Verification and Analysis : Second International Conference, ATVA 2004, Taipei, Taiwan, ROC, October 31-November 3, 2004. Proceedings*, volume 3299 of *Lecture Notes in Computer Science*, pages 248–262. Springer, 2004.

[BFL08] Florent Bouchy, Alain Finkel, and Jérôme Leroux. Decomposition of decidable first-order logics over integers and reals. In Stéphane Demri and Christian S. Jensen, editors, *Proceedings of the 15th International Symposium on Temporal Representation and Reasoning (TIME'08)*, pages 147–155, Montréal, QC, Canada, June 2008. IEEE Computer Society Press.

[BFLP03] Sébastien Bardin, Alain Finkel, Jérôme Leroux, and Laure Petrucci. FAST : Fast Acceleration of Symbolic Transition systems. In Warren A. Hunt Jr. and Fabio Somenzi, editors, *Computer Aided Verification,*

15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings, volume 2725 of *Lecture Notes in Computer Science*, pages 118–121. Springer, 2003.

[BFLP08] Sébastien Bardin, Alain Finkel, Jérôme Leroux, and Laure Petrucci. FAST : Acceleration from theory to practice. *International Journal on Software Tools for Technology Transfer (STTT)*, 10(5) :401–424, October 2008.

[BFLS05] Sébastien Bardin, Alain Finkel, Jérôme Leroux, and Ph. Schnoebelen. Flat acceleration in symbolic model checking. In Doron A. Peled and Yih-Kuen Tsay, editors, *Automated Technology for Verification and Analysis, Third International Symposium, ATVA 2005, Taipei, Taiwan, October 4-7, 2005, Proceedings*, volume 3707 of *Lecture Notes in Computer Science*, pages 474–488. Springer, 2005.

[BFS09] Florent Bouchy, Alain Finkel, and Arnaud Sangnier. Reachability in Timed Counter Systems. In Peter Habermehl and Tomáš Vojnar, editors, *Joint Proceedings of the 8th, 9th and 10th International Workshops on Verification of Infinite State Systems (INFINITY'06,'07,'08)*, volume 239 of *Electronic Notes in Theoretical Computer Science*, pages 167–178. Elsevier Science Publishers, July 2009.

[BFSP09] Florent Bouchy, Alain Finkel, and Pierluigi San Pietro. Dense-choice counter machines revisited. In Azadeh Farzan and Axel Legay, editors, *Proceedings of the 11th International Workshops on Verification of Infinite State Systems (INFINITY'09)*, Electronic Proceedings in Theoretical Computer Science, Bologna, Italy, August 2009. To appear.

[BH06] Bernard Boigelot and Frédéric Herbreteau. The power of hybrid acceleration. In Thomas Ball and Robert B. Jones, editors, *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4144 of *Lecture Notes in Computer Science*, pages 438–451. Springer, 2006.

[BH09] Béatrice Bérard and Serge Haddad. Interrupt timed automata. In Luca de Alfaro, editor, *Proceedings of the 12th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'09)*, volume 5504 of *Lecture Notes in Computer Science*, pages 197–211, York, UK, March 2009. Springer.

[BHMV94] Veronique Bruyère, Georges Hansel, Christian Michaux, and Roger Villemaire. Logic and p-recognizable sets of integers. *Bulletin of the Belgian Mathematical Society*, 1(2) :191–238, 1994.

[BHR08] Patricia Bouyer, Serge Haddad, and Pierre-Alain Reynier. Timed Petri nets and timed automata : On the discriminating power of Zeno

sequences. *Information and Computation*, 206(1) :73–107, January 2008.

[BIL09] Marius Bozga, Radu Iosif, and Yassine Lakhnech. Flat parametric counter automata. *Fundamenta Informaticae*, 91(2) :275–303, 2009.

[BJW01] Bernard Boigelot, Sébastien Jodogne, and Pierre Wolper. On the use of weak automata for deciding linear arithmetic with integer and real variables. In *Automated Reasoning, First International Joint Conference, IJCAR 2001, Siena, Italy, June 18-23, 2001, Proceedings*, volume 2083 of *Lecture Notes in Computer Science*, pages 611–625. Springer, 2001.

[BJW05] Bernard Boigelot, Sébastien Jodogne, and Pierre Wolper. An effective decision procedure for linear arithmetic over the integers and reals. *ACM Transactions on Computational Logic*, 6(3) :614–633, 2005.

[BLL+95] Johan Bengtsson, Kim Guldstrand Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL - a tool suite for automatic verification of real-time systems. In Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors, *Hybrid Systems III : Verification and Control, Proceedings of the DIMACS/SYCON Workshop, October 22-25, 1995, Ruttgers University, New Brunswick, NJ, USA*, volume 1066 of *Lecture Notes in Computer Science*, pages 232–243. Springer, 1995.

[BLR05] Patricia Bouyer, François Laroussinie, and Pierre-Alain Reynier. Diagonal constraints in timed automata : Forward analysis of timed systems. In Paul Pettersson and Wang Yi, editors, *Formal Modeling and Analysis of Timed Systems, Third International Conference, FORMATS 2005, Uppsala, Sweden, September 26-28, 2005, Proceedings*, volume 3829 of *Lecture Notes in Computer Science*, pages 112–126. Springer, 2005.

[BLT90] Tommaso Bolognesi, Ferdinando Lucidi, and Sebastiano Trigila. From timed Petri nets to timed LOTOS. In Luigi Logrippo, Robert L. Probert, and Hasan Ural, editors, *Protocol Specification, Testing and Verification X, Proceedings of the IFIP WG6.1 Tenth International Symposium on Protocol Specification, Testing and Verification, Ottawa, Ontario, Canada, 12-15 June, 1990*, pages 395–408. North-Holland, 1990.

[BM83] Bernard Berthomieu and Miguel Menasche. An enumerative approach for analyzing time Petri nets. In *Information Processing, 9th World Computer Congress, IFIP'83, Paris, France, September 19-23, 1983, Proceedings*, pages 41–46. North-Holland/IFIP, 1983.

[Boi98] Bernard Boigelot. *Symbolic Methods for Exploring Infinite State Spaces*. PhD thesis, Université de Liège, 1998.

[Bou99] Jean-Louis Boulanger. Validation des spécifications du métro automatique. *Recherche - Transports - Sécurité*, 63 :47 – 62, 1999.

[Bou02] Patricia Bouyer. *Modèles et algorithmes pour la vérification des systèmes temporisés*. Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, April 2002.

[Bou04] Patricia Bouyer. Forward analysis of updatable timed automata. *Formal Methods in System Design*, 24(3) :281–320, May 2004.

[BR08] Marc Boyer and Olivier H. Roux. On the compared expressiveness of arc, place and transition time Petri nets. *Fundamenta Informaticae*, 88(3) :225–249, 2008.

[BRW98] Bernard Boigelot, Stéphane Rassart, and Pierre Wolper. On the expressiveness of real and integer arithmetic automata (extended abstract). In Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel, editors, *Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Aalborg, Denmark, July 13-17, 1998, Proceedings*, volume 1443 of *Lecture Notes in Computer Science*, pages 152–163. Springer, 1998.

[Büc60] Julius Richard Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 6, 1960.

[CJ98] Hubert Comon and Yan Jurski. Multiple counters automata, safety analysis and presburger arithmetic. In A. Hu and M. Vardi, editors, *Computer Aided Verification, 10th International Conference, CAV '98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings*, volume 1427 of *Lecture Notes in Computer Science*, pages 268–279, Vancouver, 1998. Springer-Verlag.

[CJ99] Hubert Comon and Yan Jurski. Timed automata and the theory of real numbers. In Jos C. M. Baeten and Sjouke Mauw, editors, *CONCUR '99 : Concurrency Theory, 10th International Conference, Eindhoven, The Netherlands, August 24-27, 1999, Proceedings*, volume 1664 of *Lecture Notes in Computer Science*, pages 242–257, Eindhoven, 1999. Springer-Verlag.

[Cou04] Jean-Michel Couvreur. A BDD-like implementation of an automata package. In Michael Domaratzki, Alexander Okhotin, Kai Salomaa, and Sheng Yu, editors, *Implementation and Application of Automata, 9th International Conference, CIAA 2004, Kingston, Canada, July 22-24, 2004, Revised Selected Papers*, volume 3317 of *Lecture Notes in Computer Science*, pages 310–311. Springer, 2004.

[CP86] Gregory Cherlin and Françoise Point. On extensions of Presburger arithmetic. In *Proceedings of the 4th Easter Model Theory conference, Gross Köris Seminarberichte 86, Humboldt Universität zu Berlin*, pages 17–34, 1986.

[DA87] René David and Hassane Alla. Continuous Petri nets. 8th European Workshop on Application and Theory of Petri Nets, Zaragoza, Spain. Pages 275–294, 1987.

[DA05] René David and Hassane Alla. *Discrete, Continuous, and Hybrid Petri Nets*. Springer, 2005.

[Dan03] Zhe Dang. Pushdown Timed Automata : a Binary Reachability Characterization and Safety Verification. *Theoretical Computer Science*, 302(1-3) :93–121, 2003.

[dFEVRMA00] David de Frutos-Escrig, Valentín Valero Ruiz, and Olga Marroquín Alonso. Decidability of properties of timed-arc Petri nets. In Mogens Nielsen and Dan Simpson, editors, *Application and Theory of Petri Nets 2000, 21st International Conference, ICATPN 2000, Aarhus, Denmark, June 26-30, 2000, Proceeding*, volume 1825 of *Lecture Notes in Computer Science*, pages 187–206. Springer, 2000.

[DFGvD06] Stéphane Demri, Alain Finkel, Valentin Goranko, and Govert van Drimmelen. Towards a model-checker for counter systems. In Susanne Graf and Wenhui Zhang, editors, *Proceedings of the 4th International Symposium on Automated Technology for Verification and Analysis (ATVA'06)*, volume 4218 of *Lecture Notes in Computer Science*, pages 493–507, Beijing, People's Republic of China, October 2006. Springer.

[DFvB05] Christophe Darlot, Alain Finkel, and Laurent van Begin. About FAST and TReX accelerations. *Electronic Notes in Theoretical Computer Science*, 128(6) :87–103, 2005. Proceedings of the Workshop on Structural Operational Semantics (SOS 2004).

[Dil89] David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In Joseph Sifakis, editor, *Automatic Verification Methods for Finite State Systems, International Workshop, Grenoble, France, June 12-14, 1989, Proceedings*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1989.

[DISPX04] Zhe Dang, Oscar H. Ibarra, Pierluigi San Pietro, and Gaoyan Xie. Real-Counter Automata and their Decision Problems. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS 2004 : Foundations of Software Technology and Theoretical Computer Science, 24th International Conference, Chennai, India, December 16-18, 2004, Proceedings*,

volume 3328 of *Lecture Notes in Computer Science*, pages 198–210. Springer, 2004.

[DK96] Isabelle Demondogin and Nick T. Koussoulas. Modeling of hybrid control systems via differential Petri nets. In *4th IEEE Mediterranean Symposium on Control and Automation (MED '96), Chania, Krete. Proceedings*, pages 428–433. IEEE Computer Society Press, 1996.

[DZ98] François Demichelis and Wieslaw Zielonka. Controlled timed automata. In Davide Sangiorgi and Robert de Simone, editors, *CONCUR '98 : Concurrency Theory, 9th International Conference, Nice, France, September 8-11, 1998, Proceedings*, volume 1466 of *Lecture Notes in Computer Science*, pages 455–469. Springer, 1998.

[EK06] Jochen Eisinger and Felix Klaedtke. Don't care words with an application to the automata-based approach for real addition. In Thomas Ball and Robert B. Jones, editors, *Computer Aided Verification, 18th International Conference, CAV'06, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4144 of *Lecture Notes in Computer Science*, pages 67–80. Springer, 2006.

[EK08] Jochen Eisinger and Felix Klaedtke. Don't care words with an application to the automata-based approach for real addition. *Formal Methods in System Design*, 33(1-3) :85–115, 2008.

[FASa] The FAST tool : Fast Analysis of symbolic Transition systems. Available at : http ://www.lsv.ens-cachan.fr/Software/fast/ .

[FASb] The FASTer tool : FAST Extended Release. Available at : http ://altarica.labri.fr/wiki/tools :faster/ .

[FL02] Alain Finkel and Jérôme Leroux. How to compose presburger-accelerations : Applications to broadcast protocols. In Manindra Agrawal and Anil Seth, editors, *Foundations of Software Technology and Theoretical Computer Science, 22th International Conference, FSTTCS 2002, Kanpur, India, December 12-14, 2002, Proceedings*, pages 145–156. Springer, 2002.

[FL08] Alain Finkel and Jérôme Leroux. Presburger functions are piecewise linear. Research Report LSV-08-08, Laboratoire Spécification et Vérification, ENS Cachan, France, March 2008. 9 pages.

[Fou26] Joseph Fourier. Solution d'une question particulière du calcul des inégalités. *Nouveau Bulletin des Sciences par la Société philomatique de Paris*, pages 317–319, 1826.

[Fri98] Laurent Fribourg. A closed-form evaluation for extended timed automata. Research Report LSV-98-2, Laboratoire Spécification et Vérification, ENS Cachan, France, March 1998.

[FS08] Alain Finkel and Arnaud Sangnier. Reversal-bounded counter machines revisited. In Edward Ochmański and Jerzy Tyszkiewicz, editors, *Proceedings of the 33rd International Symposium on Mathematical Foundations of Computer Science (MFCS'08)*, volume 5162 of *Lecture Notes in Computer Science*, pages 323–334, Toruń, Poland, August 2008. Springer.

[GEN] The GENEric Presburger programming Interface (GENEPI). Available at : http ://altarica.labri.fr/wiki/tools :tapas :genepi .

[GS66] Seymour Ginsburg and Edwin H. Spanier. Semigroups, Presburger formulas, and languages. *Pacific Journal of Mathematics*, 16(2) :285–296, 1966.

[GS94] Roberto Gorrieri and Glauco Siliprandi. Real-time system verification using P/T nets. In David L. Dill, editor, *Computer Aided Verification, 6th International Conference, CAV '94, Stanford, California, USA, June 21-23, 1994, Proceedings*, volume 818 of *Lecture Notes in Computer Science*, pages 14–26. Springer, 1994.

[Hen96] Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS'96), New Brunswick, New Jersey, 27-30 July*, pages 278–292. IEEE Computer Society Press, 1996.

[HHWT97] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HyTech : A model checker for hybrid systems. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1-2) :110–122, 1997.

[HJJ+95] Jesper G. Henriksen, Jakob L. Jensen, Michael E. Jørgensen, Nils Klarlund, Robert Paige, Theis Rauhe, and Anders Sandholm. Mona : Monadic second-order logic in practice. In Ed Brinksma, Rance Cleaveland, Kim Guldstrand Larsen, Tiziana Margaria, and Bernhard Steffen, editors, *Tools and Algorithms for Construction and Analysis of Systems, First International Workshop, TACAS '95, Aarhus, Denmark, May 19-20, 1995, Proceedings*, volume 1019 of *Lecture Notes in Computer Science*, pages 89–110. Springer, 1995.

[HKOW09] Christoph Haase, Stephan Kreutzer, Joël Ouaknine, and James Worrell. Reachability in succinct and parametric one-counter automata. In Mario Bravetti and Gianluigi Zavattaro, editors, *Concurrency Theory, 20th International Conference, CONCUR 2009, Bologna, Italy, September 1-4, 2009. Proceedings*, volume 5710 of *Lecture Notes in Computer Science*, pages 369–383. Springer, 2009.

[HMG06] Nicolas Halbwachs, David Merchat, and Laure Gonnord. Some ways to reduce the space dimension in polyhedra computations. *Formal Methods in System Design*, 29(1) :79–95, 2006.

[HP79] John E. Hopcroft and Jean-Jacques Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8 :135–159, 1979.

[HRS06] Serge Haddad, Laura Recalde, and Manuel Silva. On the computational power of timed differentiable Petri nets. In Eugène Asarin and Patricia Bouyer, editors, *Proceedings of the 4th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'06)*, volume 4202 of *Lecture Notes in Computer Science*, pages 230–244, Paris, France, September 2006. Springer.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[Iba78] Oscar H. Ibarra. Reversal-Bounded Multicounter Machines and their Decision Problems. *Journal of the ACM*, 25(1) :116–133, 1978.

[Jac78] Gérard Jacob. La finitude des représentations linéaires des semigroupes est décidable. *Journal of Algebra*, 52(2) :437–459, 1978.

[KM69] Richard M. Karp and Raymond E. Miller. Parallel program schemata. *Journal of Computer System Sciences*, 3(2) :147–195, 1969.

[Kos82] S. Rao Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *Proceedings of the 14th ACM Symposium on Theory of Computing (STOC'82), San Francisco, CA, USA, May 5-7*, pages 267–281. ACM, 1982.

[LAS] The Liège Automata-based Symbolic Handler (LASH) toolset. Available at : http ://www.montefiore.ulg.ac.be/∼boigelot/research/lash/ .

[Ler03] Jérôme Leroux. *Algorithmique de la vérification des systèmes à compteurs. Approximation et accélération. Implémentation de l'outil FAST*. Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, 2003.

[Ler05] Jérôme Leroux. A polynomial time presburger criterion and synthesis for number decision diagrams. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*, pages 147–156. IEEE Computer Society, 2005.

[LIR] The Linear Interger/Real Arithmetic (LIRA) solver. Available at : http ://lira.gforge.avacs.org/ .

[LL98] François Laroussinie and Kim Guldstrand Larsen. CMC : A tool for compositional model-checking of real-time systems. In *Formal Description Techniques and Protocol Specification, Testing and Verification, FORTE XI / PSTV XVIII'98, IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XI) and Protocol Specification, Testing and Verification (PSTV XVIII), 3-6 November, 1998, Paris, France*, volume 135 of *IFIP Conference Proceedings*, pages 439–456. Kluwer, 1998.

[LP09] Jérôme Leroux and Gérald Point. TaPAS : The Talence Presburger Arithmetic Suite. In Stefan Kowalewski and Anna Philippou, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 15th International Conference, TACAS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5505 of *Lecture Notes in Computer Science*, pages 182–185. Springer, 2009.

[LPY97] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1-2) :134–152, 1997.

[LS05] Jérôme Leroux and Grégoire Sutre. Flat counter automata almost everywhere ! In Doron A. Peled and Yih-Kuen Tsay, editors, *Automated Technology for Verification and Analysis, Third International Symposium, ATVA 2005, Taipei, Taiwan, October 4-7. Proceedings*, volume 3707 of *Lecture Notes in Computer Science*, pages 489–503. Springer, 2005.

[LWYP99] Kim G. Larsen, Carsten Weise, Wang Yi, and Justin Pearson. Clock Difference Diagrams. *Nordic Journal of Computing*, 6(3) :271–298, 1999.

[May84] Ernst W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM Journal on Computing*, 13(3) :441–460, 1984.

[Mer74] P.M. Merlin. *A study of the recoverability of computing systems*. PhD thesis, University of California, Irvine, 1974.

[Min61] Marvin L. Minsky. Recursive Unsolvability of Post's Problem of "Tag" and other Topics in Theory of Turing Machines. *The Annals of Mathematics*, 74(3) :437–455, 1961.

[Min67] Marvin L. Minsky. *Computation : Finite and Infinite Machines*. Prentice-Hall, Inc., NJ, USA, 1967.

[Min01] Antoine Miné. The octagon abstract domain. In *Proceedings of the Workshop on Analysis, Slicing, and Transformation (AST'01),*

Held as part of the 8th Working Conference on Reverse Engineering (WCRE'01), 2-5 October 2001, Stuttgart, Germany, IEEE, pages 310–319. IEEE Computer Society Press, 2001.

[Min02] Antoine Miné. A few graph-based relational numerical abstract domains. In Manuel V. Hermenegildo and Germán Puebla, editors, *Static Analysis, 9th International Symposium, SAS 2002, Madrid, Spain, September 17-20, 2002, Proceedings*, volume 2477 of *Lecture Notes in Computer Science*, pages 117–132. Springer, 2002.

[MLAH99] Jesper B. Møller, Jakob Lichtenberg, Henrik Reif Andersen, and Henrik Hulgaard. Difference Decision Diagrams. In Jörg Flum and Mario Rodríguez-Artalejo, editors, *Computer Science Logic, 13th International Workshop, CSL '99, 8th Annual Conference of the EACSL, Madrid, Spain, September 20-25, 1999, Proceedings*, volume 1683 of *Lecture Notes in Computer Science*, pages 111–125. Springer, 1999.

[Mon] The MONA tool. Available at : http ://www.brics.dk/mona/ .

[Mot51] Theodore S. Motzkin. Two consequences of the transposition theorem on linear inequalities. *Econometrica*, 19(2) :184–185, 1951.

[OME] The OMEGA library. Available at : http ://www.cs.umd.edu/projects/omega/ .

[PBCT07] Julien Ponge, Boualem Benatallah, Fabio Casati, and Farouk Toumani. Fine-grained compatibility and replaceability analysis of timed web service protocols. In Christine Parent, Klaus-Dieter Schewe, Veda C. Storey, and Bernhard Thalheim, editors, *Conceptual Modeling - ER 2007, 26th International Conference on Conceptual Modeling, Auckland, New Zealand, November 5-9, 2007, Proceedings*, volume 4801 of *Lecture Notes in Computer Science*, pages 599–614. Springer, 2007.

[Pet62] Carl Adam Petri. *Kommunikation mit Automaten.* Bonn : Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962.

[PH07] Mathias Péron and Nicolas Halbwachs. An abstract domain extending Difference-Bound Matrices with disequality constraints. In Byron Cook and Andreas Podelski, editors, *Verification, Model Checking, and Abstract Interpretation, 8th International Conference, VMCAI 2007, Nice, France, January 14-16, 2007, Proceedings*, volume 4349 of *Lecture Notes in Computer Science*, pages 268–282. Springer, 2007.

[POLa] The New Polka library. Available at : http ://pop-art.inrialpes.fr/people/bjeannet/newpolka/ .

[POLb] The PolyLib tool. Available at : http ://icps.u-strasbg.fr/polylib/ .

[PPL] The Parma Polyhedra Library (PPL). Available at : http ://www.cs.unipr.it/ppl/ .

[Pre91] Mojżesz Presburger. On the completeness of a certain system of arithmetic of whole numbers in which addition occurs as the only operation. *Hist. Philos. Logic*, 12(2) :225–233, 1991. Translated from the German and with commentaries by Dale Jacquette.

[Pug91] William Pugh. The Omega test : a fast and practical integer programming algorithm for dependence analysis. In *SuperComputing(SC)'91, November 18-22, 1991, Albuquerque, NM, USA. Proceedings*, pages 4–13. IEEE Computer Society / ACM, 1991.

[Rab77] Michael O. Rabin. *Decidable theories*, chapter 3, pages 595–627. Handbook of Mathematical Logic. North Holland, 1977. Edited by Jon Barwise.

[Rey07] Pierre-Alain Reynier. *Vérification de systèmes temporisés et distribués : modèles, algorithmes et implémentabilité*. Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, June 2007.

[RS09] Pierre-Alain Reynier and Arnaud Sangnier. Weak Time Petri Nets strike back ! In Mario Bravetti and Gianluigi Zavattaro, editors, *CONCUR 2009 - Concurrency Theory, 20th International Conference, Bologna, Italy, September 1-4, 2009. Proceedings*, volume 5710 of *Lecture Notes in Computer Science*. Springer, 2009.

[San08] Arnaud Sangnier. *Vérification de systèmes avec compteurs et pointeurs*. Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, November 2008.

[Sem79] Alexei L. Semenov. On certain extensions of the arithmetic of addition of natural numbers. *Izvestiya Rossiiskoi Akademii Nauk, Seriya Matematicheskaya*, 43(5) :1175–1195, 1979.

[Sem83] Alexei L. Semenov. Logical theories of one-place functions on the set of natural numbers. *Izvestiya Rossiiskoi Akademii Nauk, Seriya Matematicheskaya*, 47(3) :623–658, 1983.

[SPKM08] P. Vijay Suman, Paritosh K. Pandya, Shankara Narayanan Krishna, and Lakshmi Manasa. Timed automata with integer resets : Language inclusion and expressiveness. In Franck Cassez and Claude Jard, editors, *Formal Modeling and Analysis of Timed Systems, 6th International Conference, FORMATS 2008, Saint Malo, France, September 15-17, 2008. Proceedings*, volume 5215 of *Lecture Notes in Computer Science*, pages 78–92. Springer, 2008.

[SVW87] A. Prasad Sistla, Moshe Y. Vardi, and Pierre Wolper. The complementation problem for büchi automata with appplications to temporal logic. *Theoretical Computer Science*, 49 :217–237, 1987.

[TAF] The SAtaf and PresTAF libraries. Available at : http ://altarica.labri.fr/wiki/tools :sataf-prestaf/ .

[TaP] The Talence Presburger Arithmetic Suite (TaPAS). Available at : http ://altarica.labri.fr/wiki/tools :tapas :start .

[Tar48] Alfred Tarski. A problem concerning the notion of definability. *Journal of Symbolic Logic*, 13(2) :107–111, 1948.

[TRe] TReX : a Tool for REachability analysis of compleX systems. Available at : http ://www.liafa.jussieu.fr/∼sighirea/trex/ .

[Upp] The UppAal tool. Available at : http ://www.uppaal.com/ .

[vdD85] Lou van den Dries. The field of reals with a predicate for the powers of two. *manuscripta mathematica*, 54(1) :187–195, 1985.

[Vil92a] Roger Villemaire. Joining k- and l-recognizable sets of natural numbers. In Alain Finkel and Matthias Jantzen, editors, *STACS'92, 9th Annual Symposium on Theoretical Aspects of Computer Science, Cachan, France, February 13-15, 1992, Proceedings*, volume 577 of *Lecture Notes in Computer Science*, pages 83–94. Springer, 1992.

[Vil92b] Roger Villemaire. The Theory of $\langle \mathbb{N}, +, V_k, V_l \rangle$ is Undecidable. *Theoretical Computer Science*, 106(2) :337–349, 1992.

[VRCGdFE99] Valentín Valero Ruiz, Fernando Cuartero Gomez, and David de Frutos Escrig. On non-decidability of reachability for timed-arc Petri nets. In *Proceedings of the 8th Int. Workshop on Petri Net and Performance Models (PNPM'99), 8-10 October 1999, Zaragoza, Spain*, pages 188–196, 1999.

[WB95] Pierre Wolper and Bernard Boigelot. An automata-theoretic approach to presburger arithmetic constraints (extended abstract). In Alan Mycroft, editor, *Static Analysis, Second International Symposium, SAS'95, Glasgow, UK, September 25-27, 1995, Proceedings*, volume 983 of *Lecture Notes in Computer Science*, pages 21–32. Springer, 1995.

[WB00] Pierre Wolper and Bernard Boigelot. On the construction of automata from linear arithmetic constraints. In Susanne Graf and Michael I. Schwartzbach, editors, *Tools and Algorithms for Construction and Analysis of Systems, 6th International Conference, TACAS 2000, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS 2000, Berlin, Germany, March 25 -*

April 2, 2000, Proceedings, volume 1785 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2000.

[Wei99] Volker Weispfenning. Mixed real-integer linear quantifier elimination. In *Symbolic and Algebraic Computation, International Symposium, ISSAC'99, Vancouver BC, Canada, July 28-31, 1999, Proceedings*, pages 129–136. ACM, 1999.

[XDISP03] Gaoyan Xie, Zhe Dang, Oscar H. Ibarra, and Pierluigi San Pietro. Dense Counter Machines and Verification Problems. In Warren A. Hunt Jr. and Fabio Somenzi, editors, *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings*, volume 2725 of *Lecture Notes in Computer Science*, pages 93–105. Springer, 2003.