République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université 20 Août-1955-SKIKDA
Faculté des Sciences
Département d'Informatique

Mémoire fin d'étude en vue de l'obtention du diplôme

Master en Informatique

Spécialté : Réseaux et Systèmes Distribués (RSD)

thème

Vérification automatique basée-modèles des systèmes temps-réel hétérogènes

Réalisé par :
— Manaa Ryem
— Djamai Maria

Encadré par :
Layadi saïd

Session : Juin 2022

Résumé

Actuellement, les méthodes formelles sont de plus en plus utilisées dans le but d'analyser le comportement des systèmes temps-réels. Ces systèmes font souvent partie des applications critiques (en ce sens qu'une erreur au cours de leur fonctionnement peut entrainer des conséquences graves) dans des domaines névralgiques comme l'avionique, le contrôle de processus industriels ou encore le contrôle de centrales nucléaires. Pour cette raison, ces systèmes doivent être vérifiés avant leur mise en œuvre.

Les méthodes de vérification formelle ou plus simplement les méthodes formelles sont de plus en plus utilisées pour répondre aux exigences auxquelles sont soumises ce type de systèmes. Dans ce travail, nous nous sommes intéressés à la vérification des systèmes temps-réel hétérogènes. Cette vérification nécessite la proposition d'approches formelles qui permettent la construction des outils de vérification. Les méthodes formelles fournissent des techniques efficaces pour vérifier un système donné.

Le modèle proposé, appelé automates temporisés avec temps relatif (R-TA), est une extension des automates temporisés standards. Dans ce modèle, les systèmes temps-réel distribués sont formés par un ensemble de R-TA. Nous supposons que chaque composant est caractérisé par un ensemble d'horloges locales qui évoluent selon une fréquence différente, mais relative par rapport à celles des horloges des autres composants. Pour étudier la sémantique de tels systèmes, nous avons défini un paramètre, appelé "slope", qui est le rapport entre les fréquences des horloges.

Aussi, on a redéfini la relation d'équivalence entre les valuations d'horloges pour aller vers des graphes de régions finis et pouvoir appliquer une approche formelle de vérification telle que model-checking.

Finalement, un outil a été développé pour construire, pour chaque R-TA, l'automate de régions correspondant.

Mots clés : systèmes temps-réel, systèmes hétérogènes, automates temporisés, R-TA, graphe de régions, model-checking.

Remerciements

Nous voulons, avant tout, remercier Dieu le tout puissant pour la force, la volonté mais surtout la santé qu'il nous a données pour entamer et finir ce mémoire.

Tout d'abort, ce travial ne serait pas riche et n'aurait pas pu voir le jour sans l'aide et l'encadrement de monsieur **Saïd Layadi**, on le remercie pour la qualité de son encadrement exceptionnel ,ses conseils précieux,pour son temps, pour sa patience, sa rigueur pour sa disponibilité durant notre préparation de ce memoire .

Pour le grand honneur qu'ils nous font en acceptant de juger ce travail. Nous vous remercions de l'honneur que vous nous avez fait en acceptant de présider notre jury. Nous vous remercions de votre enseignement et nous vous sommes très reconnaissants de bien vouloir porter intérêt à ce travail.

Nous remercions les membres du jury pour leur présence, pour leur lecture attentive de notre mémoire ainsi que pour les remarques qu'ils m'adresseront lors de cette examination afin d'améliorer notre travail.

Notre remerciement s'adresse également à tous nos professeurs pour leur générosité et la grande patience dont ils ont su faire preuve malgré leurs charge académique et professioelle.

Nos profonds remerciements vont également à toutes les personnes qui nous ont aidé et soutenue de prés ou de loin.

Dédicace

Nous dédions ce travail à chacune des membres de nos familles respectives. Les familles **Manaa** et **Yalaoui** et les familles **Djamai** et **Laib**

A nos chers parents pour leur soutien moral tout au long de nos études car nous ne pouvons rien faire sans eux,

A notre encadrant Mr. Layadi Saïd qui était toujours à nos côtés, nous n'oublierons jamais son aide à la réalisation de ce travail,

A nos sœurs Malek, Maram, Nihel, Zaiban, Ramla, Roa et Farah, elles sont les meilleures au monde,

A nos amies Maroua, Amani, Djihane, najet et Rimeh que nous n'oublierons jamais leur amitié et encouragements,

A nos cousins et cousines qui nous a toujours soutenu avec tous les moyens possibles,

A tous ceux que nous aimons.

Table des matières

Table des figures

liste d'abréviations

AT **A**utomate **T**emporisé

R-TA **A**utomate **T**emporisé avec temps **R**elatif

AR **A**utomate de **R**égions

CTAR **C**ontraintes **T**emporelles **A**tomique **R**estrientes

CTAG **C**ontraintes **T**emporelles **A**tomique **G**lobale

GR **G**raphe de **R**égion

Chapitre 1

Introduction générale

Sommaire

1.1 Contexte de travail

1.1.1 L'hétérogénéité dans la vérification des systèmes temps- réel

Récemment, un engouement très fort se fait sentir pour les systèmes temps-réel, ceux-ci gèrent de plus en plus notre quotidien. Leurs caractéristiques les plus importantes sont : la distribution et l'hétérogénéité.

Les réseaux ad-hoc mobiles (MANET) sont un exemple des systèmes distribués complexes, constitués de nœuds mobiles sans-fil qui peuvent dynamiquement s'auto-organiser en topologies de réseaux arbitraires, afin de permettre aux usagers et dispositifs de communiquer entre eux dans de bonnes conditions sur des espaces libres de toute infrastructure de communication préexistante Chlamtac et al. (2003). La flexibilité et la convenance sont les raisons pour lesquelles, leurs applications ont été étendues du domaine traditionnel (militaire) vers une diversité de domaines commerciaux qui traitent des problèmes très spécifiques, par exemple, l'intelligence ambiante Ahola (2002), les réseaux privés Zimmerman (1996) et les services basés sur la localisation Basagni et al. (2000).

Par exemple, avec l'apparition des drones de combat autonomes, la complexité des algorithmes de coordination des drones peut rendre difficile de fournir une assurance à un public concerné que ces unités autonomes coordonnent correctement leurs actions entre eux Sarkar et al. (2007). Avec un modèle formel, il serait possible de fournir et prouver à la fois des algorithmes corrects pour effectuer des tâches complexes. Aujourd'hauid, la vérification des systèmes temps réel hétérogènes est une tàche trés intéressante. Cette vérification nécessite la proposition d'approches formelles qui permettent la construction des outils de vérification. Les méthodes formelles fournissent des techniques efficaces pour vérifier un système donné. Elles utilisent une rigueur mathématique qui aide à prouver la validité du système sous-jacent.

Plusieurs modèles basés-automate ont été largement étudiés au cours des vingt dernières années, capturant une diversité d'aspects des comportements distribués. En outre, être capable de développer des techniques de vérification automatisées une bonne compréhension des modèles les plus simples, comme la plus part des modèles complexes sont construits comme une combinaison de ceux de base.
Les approches qui utilisent la vérification automatique basée modèles, dite model-checking, se sont largement développées ces derniers temps Bérard et al. (2013); Clarke et al. (1999). Cela consiste à modéliser le comportement du système à vérifier et de donner la propriété de correction attendue sous la forme, soit d'une propriété d'accessibilité, d'un automate ou encore d'une formule écrite en logique temporelle. Ensuite, on fait appel à un model-checker pour statuer sur la satisfaction de la propriété attendue par le modèle.
Généralement, le temps est mesuré par des dispositifs physiques, appelés horloges, qui fournissent un comportement presque régulier au cours du temps.
Un modèle de spécification des systèmes temps-réel doit prendre en considération les propriétés temporelles. À cet effet, les horloges sont utilisées de manière explicite dans l'expression des contraintes des systèmes.
En général, il n'y a aucune raison pour supposer que les différents composants temporisés dans les systèmes ont une même référence du temps, autrement dit, qu'ils évoluent selon un rythme similaire.
Les modèles qui utilisent un temps global (c'est-à-dire un même rythme d'évolution) pour tous les composants du système ne reflètent pas réellement les aspects des systèmes distribués.
Nous allons modéliser et vérifier les systèmes temps réel hétérogènes qui sont toujours présents dans les applications de l'Informatique et qui présentent les aspects des systèmes distribués comme les notions de réactivité, mobilité et adaptation du système aux facteurs incertains ou imprévisibles... L'ingénierie des systèmes temps réel hétérogènes (c'est-à-dire spécification, développement, gestion, déploiement...) devient de plus en plus importante, elle s'intéresse aux modèles, méthodes et outils pour leur mise en œuvre. Par conséquent, la vérification de la validité de ces systèmes devient un défi majeur.
Nous allons considérer les systèmes temps réel hétérogènes où chaque composant participant a ses propres horloges physiques avec sa propre fréquence de progression, bien qu'aucune horloge globale ne soit disponible ou souhaitable. Dans de telles circonstances, il est impossible de modéliser un système avec une sémantique sans considérer les fréquences des horloges des composants participants. D'où il est naturel d'étudier ces systèmes en fonction des différentes évolutions du temps.
La majorité des systèmes utilise explicitement des contraintes liées à des aspects temps-réel. Ces contraintes sont de type quantitatif et interprètent les délais ou les durées, comme par exemple les temps de réponse ou les timeouts. La plus part des modèles classiques et les techniques de vérification automatique ont été élargis à ce type de systèmes temps-réel.
En 1990, Alur et Dill ont proposé le modèle des automates temporisés pour décrire le comportement des systèmes temps-réel avec contraintes quantitatives Alur et al. (1993); Alur and Dill (1994). Ces automates possèdent des horloges qui s'incrémentent automatiquement d'une façon continue avec le temps et qui comptent ainsi les délais entre les différentes actions du système modélisé.
Les techniques de vérification ont été élargies pour être compatibles avec ce modèle Alur et al. (1993); Henzinger et al. (1994); Larsen et al. (1995). Des formalismes de spécification,

comme les logiques temporelles, ont aussi été élargis pour pouvoir interpréter des propriétés temps-réel qui caractérisent les systèmes Alur et al. (1993); Alur and Henzinger (1994); Aceto and Laroussinie (2002).
Il existe des travaux qui ont essayé de développer des outils de model-checking Yovine (1997); Larsen et al. (1997b), ces derniers ont été appliqués avec succès sur des cas réels dans le monde industriel Tripakis and Yovine (1998); Bengtsson et al. (2002).
Dans Akshay et al. (2008, 2014), les auteurs ont proposé un modèle pour les systèmes temporisés distribués où chaque composant est un automate temporisé avec un ensemble d'horloges locales qui évoluent avec une fréquence indépendante des horloges des autres composants. Une horloge peut être lue par n'importe quel composant du système, mais elle ne peut être réinitialisée qu'au niveau de l'automate auquel elle appartient. Pour de tels systèmes, deux sémantiques principales ont été discutées à savoir la sémantique universelle et la sémantique existentielle.
La sémantique universelle capture les comportements qui sont maintenus quel que soit le choix des fréquences d'horloges des composants individuels. Ceci est un choix naturel lors de la vérification qu'un système satisfaisant toujours une spécification positive. Cependant, pour vérifier si un système évite une spécification négative, il est préférable d'utiliser la sémantique existentielle dans laquelle on s'intéresse à l'ensemble des comportements que le système peut éventuellement présenter sous un choix de fréquences d'horloges.
La sémantique existentielle décrit toujours un ensemble régulier de comportements. Cependant, dans le cas de la sémantique universelle, le test de vide s'est avéré indécidable.
Comme la sémantique existentielle nous permet de vérifier les spécifications négatives, la sémantique universelle est convenable quand on veut vérifier si un système a un certain bon comportement. Le mot "bon" veut dire un comportement qui est robuste face aux variations d'horloges. Malheureusement, ce problème est indécidable.

1.1.2 Motivations du travail

Dans ce travail, nous adressons le problème d'hétérogénéité des composants des systèmes. L'hétérogénéité est vue comme une différence entre les rythmes d'évolution des composants. Dans différentes applications, un système est constitué de plusieurs éléments qui sont caractérisés par leurs propres fréquences d'horloges.
Par conséquent, nous allons proposer, dans ce travail, une approche pour modéliser et vérifier les systèmes distribués avec fréquences locales et relatives d'horloges. Chaque composant du système sera décrit par un automate temporisé dans lequel toutes les horloges progressent selon la même fréquence. Cependant, les horloges appartenant à différents processus (composants) seront autorisées à évoluer selon des fréquences différentes mais relatives. Nous allons supposer qu'une horloge peut être lue (utilisée) par tous les processus, alors que sa remise à zéro ne peut être effectuée que par le processus auquel elle appartient.
La définition d'une sémantique, associée au modèle proposé ci-dessus, sous forme d'un système de transitions temporisé est impossible parce que le nombre de configurations de ce dernier est infini.
Pour résoudre ce problème, nous allons rappeler les relations d'équivalence sur les valuations d'horloges qui ont été proposées pour agréger les configurations, cela veut dire qu'une classe d'équivalence (appelée région d'horloges) peut représenter un ensemble de configurations Alur

and Dill (1994).
Comme le comportement futur de tout système modélisé est déterminé à tout instant par sa localité actuelle et les valeurs des horloges de tous les processus, nous allons redéfinir de nombreux concepts tels que les régions d'horloges et l'automate de régions en se concentrant sur l'effet de la relativité entre les fréquences d'horloges.
Dans ce contexte, nous allons présenter que le problème de vérifier la satisfaction temporelle entre les composants coordonnés d'un système temps réel hétérogène peut se réduire à une recherche sur un graphe de régions.
Donc, nous allons d'abord étendre les automates temporisés avec temps relative (R-TA) par une caractérisation de vitesses relatives du temps associées aux composants pour faire face à l'hétérogénéité.

1.2 Contributions

Dans ce travail, nous nous sommes intéressés à la modélisation et la vérification des systèmes temps-réel hétérogènes. Le modèle des automates temporisés a été étendu par la notion du temps relatif afin de modéliser de tels systèmes. Nos contributions se résument dans la modélisation des systèmes temps réel hétérogènes en vue de vérifier les propriétés comportementales et temporelles de ce genre de systèmes. La décidabilité de la propriété d'accessibilité a été prouvée par le biais d'une abstraction du comportement du système appelée région. Cette abstraction nécessite la redéfinition des concepts nécessaires.
Nous sommes intéressés par les systèmes temps réel hétérogènes où chaque composant participant a ses propres horloges physiques avec sa propre fréquence de progression, bien qu'aucune horloge globale ne soit disponible ou souhaitable.
Le modèle des automates temporisés (TA) est une structure sémantique conçue comme un système de transitions. Il est utile pour exprimer la causalité et les contraintes temporelles qui doivent être respectées pendant l'exécution des actions.
Nous avons proposé une extension du modèle des automates temporisés avec temp relatif (notée R-TA) pour faire face aux vitesses relatives du temps.
Une structure R-TA représente les comportements potentiellement infinis des systèmes temporisés, ce modèle s'intéresse aux contraintes temporelles imposées par les systèmes temps réel hétérogènes, prenant en compte les vitesses relatives du temps qui différencient les composants en coordination.
Dans un R-TA, les horloges qui appartiennent aux composants différents évoluent d'une façon synchrone mais sous les rythmes relatifs assignés à ces composants. Chaque rythme assigné à un composant p représente la vitesse de p et dépend d'une certaine référence du temps global, de laquelle le temps local de p sera dérivé.
Pour étudier la sémantique de tels systèmes, nous avons défini un paramètre, appelé "slope", qui est le rapport entre les fréquences des horloges. Ce paramètre nous permet d'évaluer et prouver la décidabilité de l'accessibilité dans un R-TA.
À tout point dans le temps, le comportement futur du système modélisé est déterminé par sa localité actuelle et les valeurs des horloges de tous les processus. Pour cela, nous avons redéfini l'algorithme de construction d'un automate de régions en se basant sur l'effet de la relativité entre les fréquences d'horloges.

La sémantique d'un R-TA est donnée par deux nouvelles relations, la relation d'équivalence sur l'ensemble de toutes les valuations d'horloges et la relation du successeur sur ces classes d'équivalence. Ces deux relations nous ont permis de proposer une méthode de construction d'un automate de régions fini pour étudier et prouver la décidabilité. Cet automate de régions est similaire à l'ensemble infini des comportements impliqués par la spécification R-TA.
Nous avons présenté que la construction de l'automate de régions se termine avec un espace fini de régions. Cette construction nous offre la décidabilité pour le test du vide et model-checking.

1.3 Plan du mémoire

En plus de cette première partie qui comporte le chapitre Introduction générale, le reste du manuscrit est organisé comme suit :

Le chapitre 02 présente un état de l'art sur les systèmes hétérogènes, sysytème critique et réactif on générale et des exmple sur les système hétérogéne, et on met aussi un exmple réél d'un système critique de notre stage.

Le chapitre 03 donne ce chapitre le modèle célèbre des automates temporisés. Nous en présentons les notions de base de ce modèle, à savoir la syntaxe, la sémantique, la composition de plusieurs automates temporisés. . . ; et nous insisterons sur l'utilisation de ce modèle dans la vérification formelle avec ses différents types : accessibilité, inclusion de langages, propriétés logiques, tests, équivalences comportementales. . . Ensuite, nous abordons les extensions et les sous classes (les plus importantes) des automates temporisés en mettant l'accent sur trois points pour chaque variante : l'accessibilité d'un état donné, l'expressivité et la concision du modèle. Et on termine ce chapitre par donner quelques concepts sur la pratique de la vérification, notamment les deux approches d'analyse des systèmes en avant et en arrière, les structures DBM, les graphes de zones et une présentation succincte sur l'outil Uppaal.

Le chapitre 04 le chapitre 4 détaille les deux variantes des automates temporisés sur lesquelles nous nous basons dans cette thèse afin de résoudre les problématiques et dépasser les limites citées ci-dessus dans la motivation de ce travail. Nous avons commencé par le modèle des automates temporisés distribués avec évolution indépendante des horloges, ce modèle possède deux sémantiques principales, existentielle et universelle. Et comme l'indécidabilité de l'accessibilité persiste dans le cas de cette dernière sémantique, une troisième sémantique a été proposée pour remédier à ce problème. Concernant la deuxième variante qui est le modèle des automates temporisés avec temp relative (R-TA), nous avons expliqué le principe de la sémantique de maximalité en mettant l'accent sur ses avantages par rapport à la sémantique d'entrelacement. Ensuite, nous avons donné les concepts de base de ce modèle, à savoir la syntaxe, la sémantique et le produit des R-TA.

Le chapitre 05 présenté la partie de la réalisation de notre application et le langage de programmation et les techniques et les outils utilisés.

Et enfin une conclusion générale pour discuter les résultats et proposer quelques perspectives des travaux présentés dans ce manuscrit.

Chapitre 2

Systèmes temps-réel hétérogènes

Sommaire

2.1 Introduction

Dans ce chapitre, nous parleront de système hétérogène en générale puis nous avons donne deux exmple pour ce système le cloud computing et les réseaux ad hoc (HANET)et finalement un exemple réel de note stage.

2.2 Systèmes hétérogènes

Vous obtenez une architecture système hétérogène ou HSA, qui permet an accès a bande passante élevé A la mémoire et des performances d'application élevées à faible consommation d'énergie. HSA définit des interface pour le calcul parallèle utilisant CPU GPU et autres dispositifs programmables et à fonction fixe et prise en charge d'un ensemble diversifié de langage de programmation de haute niveau, créant ainsi la prochaine base de l'information à usage général, cela placera des CPU, les GPU et autre accélérateurs que citoyens égaux sur les plateforme informatique ou les application circuleront antre les type de processeur en fonction de leur charge de travail l'objective est que HSA permettre aux développeurs hétérogène en définissant et en promouvant une approche ouvert basé sur les norme de l'industrie.
pour le calcul la facilité de programmabilité est réalisable en permettant de tirer parti des langages de programmation existants cela proposer une spécification matérielle commune et un large écosystème de support pour permettre aux développeurs de logiciel de fournir plus

facilement de application innovantes qui peuvent tirer d'avantage parti des processeurs moderne aujourd'hui des applications sont tourné vers l'avenir car elles seront compatibles avec les future ISA, ce qui contribue à garantir la compatibilité, l'essentiel est que les applications actuelles seront conçues pour fournisseur de matériel comme AMD peuvent modifier leur matériel pour se différencier des autres sociétés, mais encore une fois maintenir la compatibilité, des application HSA établit les bases idéales pour créer des solution comme apencl c++, Python, java. . .etc.
HSA devient le cadre sur lequel ces langages et technologies de haute niveau existants peuvent être construits, il démocratise le calcul et le rend ouvert à toutes les programmations d'outils pourront tirer parti du temps d'exécution HSA pour créer des outils personnalisés à différente couche de la pile. HSA il est HSA intermédiaire langage qu'est le langage d'assemblage de HSA c'est le virtuel est un qui encapsule de nombre HSA.
HSA protège la pile logicielle de la cause matérielle spécification mais offre toujours des performances via le finaliseur au développeur, cela apporte une écriture une fois exécuté n'importe où multiplateforme la programmabilité, mais maintenant avec les performances la conception d'application pour tirer parti de l'information hétérogène entrainera la prochaine génération de percées et contribuera en fait de manière signification à améliorer l'expérience utilisateur.
La fondation HSA est sur le point de révolution la nouvelle vague et la prochain gérération d'application comme l'interface utilisateur naturelle et les gestes, expérience qui va au-delà de la HD expériences contenu audio vidéo et information continue avec du contenu disponible sur tous vos appareils information reconnaissance biométrique et réalité augmentée.
habituellement, l'étérogénéité dans le contexte de l'information fait référence à différentes architectures de jeux d'instructions (ISA), où le processeur principal en a une et les autres processeurs en ont une autre - généralement très différente - architecture (peut-être plus d'une), pas seulement une microarchitecture différente (le traitement des nombres à virgule flottante en est un cas particulier - généralement pas qualifié d'hétérogène).
Dans le passé, l'informatique hétérogène signifiait que différents ISA devaient être gérés différemment, tandis que dans un exemple moderne, les systèmes à architecture de système hétérogène (HSA) éliminent la différence (pour l'utilisateur) tout en utilisant plusieurs types de processeurs (généralement des CPU et des GPU), généralement sur le même circuit intégré, pour offrir le meilleur des deux monde : le traitement GPU général (en plus des capacités de rendu graphique 3D bien connues du GPU, il peut également effectuer des calculs mathématiquement intensifs sur de très grands ensembles de données), tandis que les processeurs peuvent exécuter le système d'exploitation et effectuer des tâches série traditionnelles.
Le niveau d'hétérogénéité dans les systèmes informatiques modernes augmente progressivement à mesure que la mise à l'échelle des technologies de fabrication permet aux composants autrefois discrets de devenir des parties intégrées d'un système sur puce, ou SoC. en logique pour l'interfaçage avec d'autres appareils (SATA, PCI, Ethernet, USB, RFID, radios, UART et contrôleurs de mémoire), ainsi que des unités fonctionnelles programmables et des accélérateurs matériels (GPU, coprocesseurs de cryptographie, processeurs de réseau programmables, A /V encodeurs/décodeurs, etc.).
Dans le passé, l'informatique hétérogène signifiait que différents ISA devaient être gérés différemment, tandis que dans un exemple moderne, les systèmes à architecture de système

hétérogène (HSA) éliminent la différence (pour l'utilisateur) tout en utilisant plusieurs types de processeurs (généralement des CPU et des GPU), généralement sur le même circuit intégré, pour offrir le meilleur des deux mondes : le traitement GPU général (en plus des capacités de rendu graphique 3D bien connues du GPU, il peut également effectuer des calculs mathématiquement intensifs sur de très grands ensembles de données), tandis que les processeurs peuvent exécuter le système d'exploitation et effectuer des tâches série traditionnelles.
Le niveau d'hétérogénéité dans les systèmes informatiques modernes augmente progressivement à mesure que la mise à l'échelle des technologies de fabrication permet aux composants autrefois discrets de devenir des parties intégrées d'un système sur puce, ou SoC. en logique pour l'interfaçage avec d'autres appareils (SATA, PCI, Ethernet, USB, RFID, radios, UART et contrôleurs de mémoire), ainsi que des unités fonctionnelles programmables et des accélérateurs matériels (GPU, coprocesseurs de cryptographie, processeurs de réseau programmables, A /V encodeurs/décodeurs, etc.).
Les systèmes hétérogènes nous permettent de cibler notre programmation sur l'environnement approprié. La programmable des FPGA doit s'améliorer s'ils doivent faire partie de l'informatique grand public.

2.3 Exemple pour les systèmes hétérogènes

2.3.1 Le cloud computing

Le cloud computing est devenu de plus en plus répendu, offrant aux utilisateurs finaux un accès temporaire à des ressources informatiques évolutives. Au niveau conceptuel, le cloud computing devrait convenir aux utilisateurs d'informatique technique, tels que les scientifiques qui ont souvent besoin d'exécuter des travaux intensifs en calcul dans le cadre de leur travail. En effet, les scientifiques commencent déjà tirer parti des ressources du cloud computing pour exécuter workflows scientifiques.
Les centres de données et de calcul sont souvent limités par la puissance densité et efficacité, ainsi que la densité de calcul. Tandis que fabricants de microprocesseurs et de serveurs à usage général travaillent pour améliorer l'efficacité énergétique, hétérogène les ressources de traitement peuvent fournir un ordre de grandeur ou plus d'amélioration en utilisant ces mesures. Ces améliorations sont susceptibles d'être persistants, car les dispositifs spécialisés peuvent être optimisé pour des types de calculs spécifiques, et cela l'optimisation peut être effectuée pour l'efficacité. Il y a de nombreux exemples de problèmes bien adaptés à des architectures. Des exemples de telles architectures comprennent les processeurs de signaux numériques, les processeurs de paquets réseau, les processeurs graphiques unités de traitement (GPU, également appelés GPGPU, GPU à usage général, dans ce contexte), multiprocesseurs symétriques (SMP) et les processeurs conventionnels.
L'infrastructure cloud d'aujourd'hui, à quelques exceptions notables près (par exemple, SGI Cyclone, R System, GPU Amazon Cluster), se concentre généralement sur le matériel de base, sans contrôle sur architectures cibles en plus de choisir parmi un nombre fixe des tailles de mémoire/CPU. Si les utilisateurs du cloud doivent pouvoir prendre profiter des avantages de performance et d'efficacité de l'informatique hétérogène, le logiciel d'infrastructure cloud doit reconnaître et gérer cette hétérogénéité.
Dans le passé, le calcul en grille et la planification par lots ont tous deux couramment utilisés

pour le calcul à grande échelle. Le cloud computing présente une allocation de ressources différente paradigme que les grilles ou les planificateurs de lots. En particulier, Amazon EC2 est équipé pour gérer de nombreuses allocations de ressources de calcul plus petites, plutôt que quelques requêtes volumineuses comme est normalement le cas avec le calcul en grille. L'introduction d'hétérogénéité permet aux cloud d'être compétitifs avec les systèmes informatiques distribués traditionnels, qui consistent souvent en également différents types d'architectures. Lorsqu'il est combiné avec économies d'échelle, provisionnement dynamique et comparativement des dépenses en capital plus faibles, les avantages de l'hétérogénéité les nuages sont nombreux.
Le cloud computing permet aux utilisateurs individuels d'avoir un accès administratif à une instance de machine virtuelle dédiée. Le la capacité à séparer les utilisateurs est supérieure par rapport à un lot approche de planification, où il est courant pour plusieurs travaux pour partager un seul système d'exploitation. Les avantages de ce ressortent du point de vue de la sécurité ainsi que flexibilité pour les utilisateurs, offrant une variété de systèmes d'exploitation. Certaines implémentations de planification par lots, telles que LSF, dépendent d'une configuration presque identique de nœuds de calcul à travers un cluster, une tâche potentiellement ardue pour le système administrateurs.

2.3.2 Les réseaux ad hoc

Les réseaux ad hoc hétérogènes (HANET) sont des composants importants de l'Internet des objets, qui deviennent une tendance inévitable dans les futures recherches et applications. Au cours des dernières années, l'ad hoc Les réseaux ont été largement utilisés dans de nombreux domaines, en particulier dans la surveillance de l'environnement, le contrôle des armes, le transport intelligent, la ville intelligente et d'autres domaines. Les HANET consistent en des réseaux de capteurs sans fil, réseaux ad hoc intelligents, réseaux de fidélité sans fil, réseaux de télécommunication, réseaux ad hoc véhiculaires, etc...
Les informations numériques et les objets physiques sont intégrés par des méthodes de communication appropriées, créant ainsi de nouvelles applications et de nouveaux services.
Différentes applications utilisent l'indépendant structures de réseau, qui forment une plateforme de réseau hétérogène et augmentent la complexité opérationnelle de la communication entre eux.
Cet article présente une architecture typique de la grande échelle HANETs, et étudie les progrès de la recherche des technologies clés actuelles.
Pour résoudre les problèmes existants, nous suggérons quelques solutions potentielles pour faire face aux défis actuels, tels que l'auto-organisation, les grandes transmissions de données, protection de la vie privée, fusion et traitement de données pour les HANET à grande échelle.
Le réseau ad hoc est un système de réseau temporaire distribué. Il est formé par le lien dynamique des nœuds et ne repose pas sur les infrastructures réseau existantes, telles que routeur, passerelle, alimentation électrique régulière, etc.
Le réseau ad hoc est une sorte de peer to peer réseaux et chaque nœud a des fonctions de collecte de données, de stockage, traitement et transmission. C'est la solution économique pour la communication à courte portée dans certains scénarios particuliers, tels que comme champ de bataille, sauvetage en cas de catastrophe, détection de l'environnement, etc.
Afin d'améliorer la qualité de service (QoS) entre les différentes unités de réseau hétérogènes,

les HANET deviennent le centre de recherche ces dernières années .
Les HANET consistent généralement en des réseaux de capteurs sans fil (WSN), réseaux ad hoc intelligents, réseaux de fidélité sans fil, réseaux de télécommunication , réseaux ad hoc véhiculaires (VANET), etc.
les unités de réseau hétérogènes sont accessibles et interconnectées via les nœuds passerelles. Les WSN comprennent un grand nombre de des nœuds de capteurs spécialisés, qui peuvent mettre en place dynamiquement un réseau de communication auto-organisé. Les utilisateurs peuvent accéder à les données de détection qui contiennent des informations précieuses et fiables collectées par nœuds capteurs. Les réseaux intelligents ad hoc sont largement appliqués entre différents dispositifs de communication temporaire. Avec le développement de l'Internet des objets, hétérogène les réseaux de capteurs sans fil se sont rapidement développés. Il y a deux types de nœuds dans les réseaux hétérogènes de capteurs sans fil, les nœuds réguliers et les super nœuds. Ils ont les mêmes chances d'accéder au réseau, mais ont des fonctions différentes. Les nœuds réguliers sont chargés de surveiller le milieu environnant, d'envoyer et transmettre les données de détection aux super nœuds. Super nœuds collecter principalement les données des nœuds réguliers et connecter les autres types de réseaux par nœuds passerelles. Les données de détection peuvent être envoyé au centre de données basé sur le cloud qui traite les données et gérer le réseau.
Dans la plupart des scénarios sans infrastructures de communication fixes, les terminaux intelligents doivent communiquer entre eux temporairement. Bluetooth est la meilleure solution pour répondre aux communications sans fil à courte portée. Le Bluetooth joue un rôle essentiel dans les réseaux ad hoc intelligents et a de larges applications, telles que l'automobile, les périphériques informatiques, la maison intelligente, etc... .

2.4 Exemple réel de notre stage

2.4.1 programmation

La commande des unités de choc d'air doit fonctioner en mode automatique et en mode manuel
En mode automatique, les unités de choc d'air doivent etre tirés individuellement ou en groupes à intervalles de temps programéées(service de cycle).
Il faut veiller à ce que les unités de choc d'air ne soient pas tirés trop souvent. Vérifier que le KIDS reste protégé de la surchauffe par une couche de clinker pendant le fonctinnement.

Programmation de groupes

chaque unités de choc d'air pour l'entrer du refroidisseur peut étre alloué paralléllemant à différents groupes pour services de cycle.
groupe 1 -groupe de cycle
groupe 2 -groupe de cycle
...
groupe(n-1)-groupe de cycle
Les unités de choc d'air pour les conduite de poussiére du capot de chauffe doivent étrealloúees à sa propre groupe .
groupe n-groupe de cycle pour la conduite de poussière du capot de chauffe .

Temporisation des cycles

Les temps des cycless, c'est-à-dire l'intervalle de temps entre deus tirs d'un groupe de cycle doivent étre programmés.
temps de cycle groupe 1 -xxx min
temps de cycle groupe 2 -xxx min
...
temps de cycle groupe n -xxx min

Ajustement du temps de pilotage de l'électrovanne des unités de choc air

Pour tirer les unités de choc d'air en mode manuel et automatique, les temporisation suivants doivent etre ajusté :

- shoot pulse : temps de pilotage sur lélectrovanne.
- pause pulse : si une unité de choc d'air est tiré, il y a programmée une courte période avant le lancemant de la deuxième unité(protection du refroidisseur).

ATTENTION
Les unités de choc d'air ne doivent pas etre tirés en meme temps, autrement le refroidisseur peut etre endommagé par le jet d'air très violent. En tout cas, le pause pulse doit etre programmé avec la valeur mentinée .
Les valeur suivant doivent etre enregistrées pour le fonctionnemant correct des unités de choc d'air

- shoot pulse :150ms
- pause pulse :500ms

Temps de charge

Le temps de charge doit etre ajusté à 1 min.
Cela signifie, que le min. temps entre deux tirs d'un canon à air est 1 min

2.4.2 Service

Mode manuel

Un bouton de touche "shoot" et pour chaque unité de choc d'air un commutateur devraient etre installés localemant .

- autoriser le signal central"local control" pour le mode manuel
- mettre en marche les unités de choc d'air qui doivent etre tirés
- tirer en pressant le boutton de touche "shoot"

Mode automatique

- programmer les groupes (voir section programmation de groupe).
- ajuster les valeurs de temps de cycle(voir section Temporisation des cycles).
- autoriser le signal central"cycle control" pour le service de cycle .

Les unités de choc d'air seront tirés automatiquement dépendants de groupe programmés et temps de cycle .

2.4.3 Visualisation des valeurs

les valeurs suivantes doivent etre visualisées dans la salle de controle :
remainingtime to shot 1..n
le temps restant au tir des unités de choc d'air du groupe correspondant en min(n-quantité des groupe programmés

2.4.4 Signaux centraux

Sorties digitales du syetème de processus de controle

local control si ce signal est activé les unités de choc d'air peuvent etre tirés en mode locale.
cycle control* si ce signal est activé les unités de choc d'air trvaillenten cycle automatique.
center shoot si "cycle control"est activé les unités de choc d'air présélectionnés seront tirés immédiatement, indépendamment du cycle programmé.

Entrées digitales dans le système de processus de controle

failure(low active) une alarme est déclenché si :

- les interrupteus de proximité des portes du corps supérieur du refroidisseur (si presents) ne sont pas activés.
- l'interrupteur d'urgence est poussé. 'interrupteur à pression (si existant) de l'unité d'entretien indique une insuffisance de pression.
- une faute du tension d'alimentation survient.

shoot(high active)
Ce signal est activé un temps trés bref si un des groupe d'unité de choc d'air a tiré. Il peut etre utilisé pour l'indication dans la salle de control.

2.4.5 Message

emergency
main pressure low l'interrupteur à pression (si existant) a été activé .

Chapitre 3

Automates temporisés : Un modèle pour une vérification formelle et automatique

Sommaire

3.1 Introduction

Un problème très intéressant aujourd'hui est de vérifier les systèmes réactifs, critiques ou embarqués. Les approches qui utilisent la vérification automatique basée modèles, dite model-checking, se sont largement développées ces derniers temps Bérard et al. (2013); Clarke et al. (1999). Cela consiste à modéliser le comportement du système à vérifier et de mettre en place la propriété de correction attendue sous la forme, soit d'une propriété d'accessibilité, d'un automate ou encore d'une formule écrite en logique temporelle. Après, un model-checker peut être utilisé pour statuer sur la satisfaction de la propriété attendue par le modèle.

Une classe importante des systèmes utilise explicitement des contraintes liées à des aspects temps-réel. Ces contraintes sont de type quantitatif et interprètent les délais ou les durées, comme par exemple les temps de réponse ou les timeouts. La plus part des modèles classiques et les techniques de vérification automatique ont été élargis à ce type de systèmes temps-réel. Alur et Dill ont proposé, en 1990, le modèle des automates temporisés pour décrire le comportement des systèmes temps-réel avec contraintes quantitatives Alur et al. (1993); Alur and Dill (1994). Ces automates possèdent des horloges qui s'incrémentent automatiquement d'une façon continue avec le temps et qui comptent ainsi les délais entre les différentes actions du système modélisé. Les techniques de vérification ont été élargies pour être compatibles avec ce modèle Alur et al. (1993); Henzinger et al. (1994); Larsen et al. (1995). Des formalismes de spécification, comme les logiques temporelles, ont aussi été élargis pour pouvoir interpréter des propriétés temps-réel qui caractérisent les systèmes Alur et al. (1993); Alur and Henzinger (1994); Aceto and Laroussinie (2002). Enfin, il existe des travaux qui ont essayé de développer des outils de model-checking Yovine (1997); Larsen et al. (1997b), ces derniers ont été appliqués avec succès sur des cas réels dans le monde industriel Tripakis and Yovine (1998); Bengtsson et al. (2002).

Dans ce chapitre, nous présentons le modèle des automates temporisés. Nous présentons aussi comment obtenir les procédures de décision sur ce type de systèmes ayant un nombre infini d'états. Nous citons aussi les variantes de ce modèle qui se diffèrent dans la formalisation des contraintes, des mises à jour... Des extensions comme les systèmes hybrides linéaires seront aussi mentionnées Alur et al. (1995); Henzinger (2000). Nous décrivons aussi quelques sous-classes des automates temporisés qui implémentent des propriétés particulières. Et nous clôturons ce chapitre par présenter quelques notions algorithmiques et de l'outil de vérification Uppaal Larsen et al. (1997b).

3.2 Syntaxe, sémantique et composition des automates temporisés

Dans les années 1990, les automates temporisés ont été introduits par R. Alur et D. Dill Alur and Dill (1994). Ils ont étendu les automates classiques par des horloges qui s'incrémentent de façon continue et synchrone avec le temps. Deux nouvelles notions ont été ajoutées aux transitions :

— Une garde définie sur la valeur des horloges décrivant l'instant où la transition peut être tirée et une réinitialisation à zéro pour un ensemble d'horloges lors du tirage de la transition.

— Une autre nouvelle notion appelée invariant a été ajoutée aux localités et exprimée sous la forme d'une contrainte sur les horloges, cet invariant peut forcer le système à quitter une localité pour lancer l'exécution d'une action en limitant le temps d'attente dans cette localité.

Le domaine de temps peut être soit l'ensemble des entiers naturels $\mathbb{N}$, soit l'ensemble des rationnels positifs $\mathbb{Q}_{\geq 0}$ ou bien l'ensemble des réels positifs $\mathbb{R}_{\geq 0}$. Dans cette section, nous considérons les réels positifs, cependant, il faut noter qu'il n'y a pas une grande différence dans les résultats si on considère $\mathbb{Q}_{\geq 0}$ ou $\mathbb{N}$. En effet, par la considération de $\mathbb{N}$, la granularité des périodes de progression du temps peut être adoptée pour l'analyse des propriétés du système.

3.2.1 Préliminaires

Dans ce qui suit, nous considérons $\mathbb{R}_{\geq 0}$ l'ensemble des nombres réels non négatifs. Pour $t \in \mathbb{R}_{\geq 0}$, $\lfloor t \rfloor$ et $fract(t)$ se réfèrent respectivement aux parties intégrale et fractionnaire de t, c'est-à-dire t=$\lfloor t \rfloor$+fract(t).
Soit H un ensemble d'horloges à valeur dans $\mathbb{R}_{\geq 0}$. Une valuation v pour H est une fonction $(v : H \to \mathbb{R}_{\geq 0})$ qui associe à chaque horloge x sa valeur $v(x)$. On note $\mathbb{R}^H_{\geq 0}$ l'ensemble des valuations pour H. Étant donné un réel $d \in \mathbb{R}_{\geq 0}$, on note $v + d$ la valuation qui associe à l'horloge x la valeur $v(x) + d$. Si R est un sous-ensemble de H, $v[R]$ représente la valuation v' définie par : $v'(x) = 0$ pour tout $x \in R$ et $v'(x) = v(x)$ pour $x \in H \backslash R$.
On note $C(X)$ l'ensemble des contraintes d'horloges sur X, c'est-à-dire l'ensemble des combinaisons booléennes de contraintes atomiques de la forme $x \bowtie c$ avec $x \in X$, $\bowtie \in \{=, <, \leq, >, \geq\}$ et $c \in \mathbb{R}_{\geq 0}$. On désigne par $C \lessdot (X)$ la restriction de $C(X)$ aux combinaisons positives ne contenant que des contraintes de la forme $x \leq c$ ou $x < c$. Les contraintes d'horloges s'interprètent de manière naturelle sur les valuations d'horloges : une valuation v satisfait une contrainte atomique $x \bowtie c$ lorsque $v(x) \bowtie c$, l'extension aux contraintes quelconques est alors immédiate. Lorsqu'une valuation v satisfait une contrainte C, on écrit $v \models C$.

3.2.2 Modèle des automates temporisés et sa sémantique

On peut définir un automate temporisé d'une manière formelle comme suit :

Définition 3.1

Un automate temporisé A est un 6-uplet (S, s_0, H, I, T, Act) où :
— S est un ensemble fini de localités,
— $s_0 \in S$ est la localité initiale,
— H est un ensemble fini d'horloges,
— $T \subseteq S \times C(H) \times Act \times 2^H \times S$ est un ensemble fini de transitions, $e = (s, G, a, R, s') \in T$ (notée par sG, a, Rs') représente une transition de s vers s' avec une garde G, un ensemble d'horloges R à réinitialiser et une action a qui représente l'étiquette de la transition,
— $I : S \to C_{\lessdot}(H)$ associe un invariant à chaque localité,
— Act est un alphabet d'actions.

Figure 3.1 représente un exemple d'automate temporisé Alur and Dill (1994).

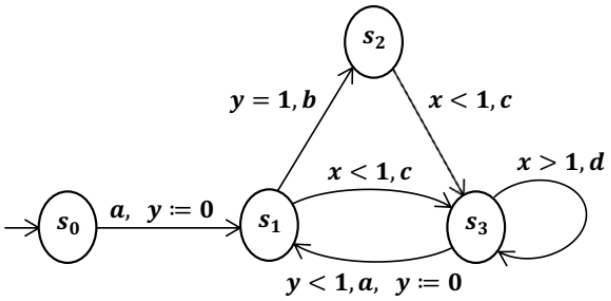


FIGURE 3.1 – Automate temporisé.

Un état ou une configuration d'un automate temporisé est une paire $(s, v) \in S \times \mathbb{R}^H_{\geq 0}$ où s représente la localité courante et v une valuation pour les horloges. La sémantique d'un automate temporisé est donnée par un système de transitions temporisé qui comporte des transitions étiquetées par, soit un élément de Act (une transition d'action) soit des durées réelles (une transition de temps) :

Définition 3.2

Un système de transitions temporisé (STT) est un quadruplet $\mathcal{S} = (Q, q_0, \rightarrow, Act)$ où Q est un ensemble d'états (éventuellement infini), $q_0 \in Q$ est l'état initial et $\rightarrow \subseteq Q \times (Act \cup \mathbb{R}_{\geq 0}) \times Q$ est la relation de transition. La relation $\rightarrow$ doit satisfaire les trois propriétés suivantes :

- si $q0q'$, alors $q = q'$,
- si qdq' et $q'd'q''$ avec $d, d' \in \mathbb{R}_{\geq 0}$, alors $qd''q''$,
- si qdq' avec $d \in \mathbb{R}_{\geq 0}$, alors pour tout $0 \leq d' \leq d$, il existe $q'' \in Q$ tel que $qd'q''$.

Les trois conditions citées plus haut expriment que le temps est continu et déterministe. Elles sont traditionnelles dans les systèmes temporisés, voir par exemple Wang (1990).
De manière standard, une exécution dans un STT est représentée par une suite de transitions successives. Dans $\mathcal{S}$, un état $q \in Q$ est dit atteignable s'il existe une exécution menant de l'état initial q_0 jusqu'à l'état q.

Définition 3.3

Soit $A = (S, s_0, H, I, T, Act)$ un automate temporisé. La sémantique de A est donnée par le STT $\mathcal{S}_A = (Q, q_0, \rightarrow, Act)$ où :

- $Q = S \times \mathbb{R}^H_{\geq 0}$,
- $q_0 = (s_0, v_0)$ avec $v_0(x) = 0$ pour tout $x \in H$,
- la relation de transition $\rightarrow$ peut exprimer deux types de transitions :
  - les transitions d'actions : $(s, v)a(s', v')$ si et seulement s'il existe $sG, a, Rs' \in T$ tel que $v \models G, v' = v[R]$ et $v' \models I(s')$.
  - les transitions de temps : pour $d \in \mathbb{R}_{\geq 0}$, $(s, v)d(s, v+d)$ si seulement si $v+d \models I(s)$.

Informellement, le système de transitions temporisé démarre de l'état initial (localité s_0 et toutes les horloges à zéro), ensuite il y a deux types de transitions à tirer alternativement si les conditions de tir sont satisfaites :

— les transitions d'actions si la valuation d'horloges courante le permet. Ce type de transitions s'accomplit alors instantanément avec une possibilité de remise à zéro de certaines horloges.
— et les transitions de temps qui incrémentent toutes les horloges avec une même durée (les horloges progressent d'une manière synchrone) si l'invariant de la localité courante est toujours respecté.

Une exécution possible de l'automate temporisé de Figure 3.1 est :
$(s_0,(0,0))7.35(s_0,(7.35,7.35))a(s_1,(7.35,0))1(s_1,(8.35,1))b(s_2,(8.35,1))...$
où le couple $(8.35,1)$ résume la valuation v telle que $v(x)=8.35$ et $v(y)=1$.

Une exécution dans un automate temporisé peut aussi être représentée par un mot temporisé, c'est-à-dire une séquence de paires (action,date). Donc, une exécution peut être vue comme suit : $(s_0,v_0,t_0)a_1(s_1,v_1,t_1)a_2...a_n(s_n,v_n,t_n)$ avec $t_i \in \mathbb{R}_{\geq 0}$, $t_0=0$ et $t_{i+1} \geq t_i$ pour tout i. La date t_i coïncide avec la date de l'exécution de l'action a_i.
L'étape $(s_i,v_i,t_i)a_{i+1}(s_{i+1},v_{i+1},t_{i+1})$ s'exprime avec une attente de durée égale à $t_{i+1}-t_i$ puis le lancement de l'action a_{i+1}, la valuation v_{i+1} est donc calculée à partir de $v_i+(t_{i+1}-t_i)$ sans oublier de réinitialiser certaines horloges (selon la transition choisie). Alors, le mot temporisé associé est $(a_1,t_1)(a_2,t_2)...$ Par exemple, le mot temporisé de l'exécution présentée ci-dessus est $(a,7.35)(b,8.35)...$

3.2.3 Composition parallèle d'automates temporisés

Il est possible de donner une définition de manière classique de la composition parallèle d'automates temporisés (ou de STT). Par exemple, on peut construire une synchronisation n-aire avec renommage. Si ce type d'opération est nécessaire pour la modélisation de systèmes, il faut noter que sur le plan théorique, cette composition n'élargie pas l'expressivité du système global modélisé : on peut toujours appliquer un produit synchrone sur l'ensemble des automates temporisés pour avoir un automate résultant ayant le même comportement (selon la bisimulation forte) que la composition parallèle effectuée.

3.3 Décidabilité de la propriété d'accessibilité

Dans cette partie, nous voulons décrire une construction proposée dans Alur and Dill (1994) pour décider si une localité dans un automate temporisé est accessible ou non. Cette construction est alors une abstraction du comportement de l'automate temporisé considéré de telle sorte que le test de l'accessibilité d'une localité dans un automate temporisé se réduit à tester si un état (ou un ensemble d'états) dans un autre automate fini est accessible.
L'idée de cette construction est la génération d'un automate à états fini dans lequel un état peut être une agrégation d'une infinité d'états du système de transitions temporisé. De ce fait, toute transition par action ou par passage de temps dans le système de transitions temporisé trouve son équivalence dans l'automate construit, l'inverse est toujours vrai. Observons que les durées précises d'attente dans les localités ne sont pas forcément les mêmes : donc la relation d'équivalence visée est une relation de bisimulation qui fait abstraction du temps et qui tente à obtenir un nombre fini de classes d'équivalence. Une telle relation est possible

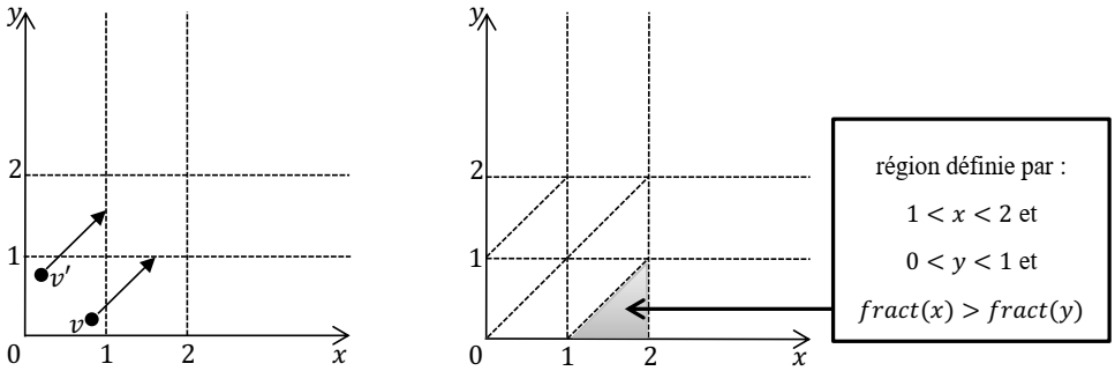
pour les automates temporisés et elle est définie comme suit : deux configurations (s, v) et (s', v') sont équivalentes si $s = s'$ et les deux valuations d'horloges sont équivalentes $v \sim v'$.

Pour toute horloge x, soit C_x la plus grande valeur avec laquelle l'horloge x a été comparée dans l'automate (supposons que cette valeur existe). On dit que deux valuations d'horloges v et v' sont équivalentes, $v \sim v'$, si et seulement si :

- pour toute horloge x :
  - $v(x) > C_x \Leftrightarrow v'(x) > C_x$,
  - si $v(x) \leq C_x$, alors $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$ et ($fract(v(x)) = 0$ si et seulement si $fract(v'(x)) = 0$),
- et pour toute paire d'horloges (x, y) :
  - si $v(x) \leq C_x$ et $v(y) \leq C_y$, alors on a $fract(v(x)) \leq fract(v(y))$ si et seulement si $fract(v'(x)) \leq fract(v'(y))$.

Intuitivement, les deux premières conditions expriment que si une valuation d'horloges satisfait certaines contraintes de l'automate temporisé alors l'autre valuation qui lui est équivalente satisfait les mêmes contraintes. La troisième condition assure le même ordre d'arrivée aux nouvelles valeurs entières quand les différentes horloges s'incrémentent à partir de deux configurations équivalentes. L'équivalence $\sim$ définie ci-dessus concerne les valuations d'horloges, une classe d'équivalence est appelée donc une région d'horloges.

La construction des régions d'horloges est illustrée par Figure **??** où il y a deux horloges x et y et une constante maximale $C_x = C_y = 2$. Si on applique les deux premières conditions de la relation d'équivalence $\sim$ sur cet exemple, on aura la répartition présentée sur Figure **??**(a) qui donne des régions valables pour toutes les contraintes temporelles définies avec des constantes plus petites ou égales à 2. Mais le problème qui se pose c'est qu'on peut trouver deux valuations dans la même région et qui ne sont pas équivalentes vis-à-vis de l'écoulement du temps (voir la troisième condition de la relation d'équivalence $\sim$ ci-dessus). Par exemple, prenons les deux valuations v et v' (voir Figure **??**(a)), si on laisse le temps s'écouler à partir de v, on va d'abord atteindre la valeur entière $x = 1$ avant que l'horloge y atteigne la valeur entière 1, alors qu'à partir de v', on va atteindre la valeur entière $y = 1$ avant que l'horloge x atteigne le 1. Les comportements possibles à partir de v et v' ne sont pas alors les mêmes. La troisième condition de $\sim$ raffine la répartition présentée sur Figure **??**(a) par des lignes diagonales qui représentent l'écoulement du temps et donne la répartition de Figure **??**(b), qui s'avère être une relation de bisimulation avec abstraction du temps.



(a) répartition respectant les deux premières conditions de ∼

(b) répartition respectant les trois conditions de ∼

FIGURE 3.2 – Index de régions pour deux horloges qui ont la même valeur maximale 2.

Avec cette relation d'équivalence, on peut construire un automate fini à partir de l'automate temporisé initial A de la manière suivante : les états (configurations) de l'automate sont les paires (s, α) où s est une localité de l'automate temporisé et α est une région d'horloges construite selon les conditions de la relation d'équivalence α ; les transitions sont $(s, \alpha)a(s', \alpha')$ s'il existe :

- une transition sG, a, Rs' dans l'automate temporisé A,
- une valuation d'horloges $v \in \alpha$ et $t0$ tels que :
  - $v + t \models I(s)$,
  - $v + t \models G$;
  - $(v + t)[R] \models I(s')$ et
  - $(v + t)[R] \in \alpha'$.

L'automate fini $R(A)$ résultant de cette construction associé à l'automate temporisé initial A est appelé l'automate de régions. La propriété essentielle de cet automate de régions est qu'il reconnait exactement le même langage que celui de l'automate temporisé initial, c'est-à-dire, si $R(A)$ reconnaît un mot $a_1a_2...$ alors il existe un mot temporisé $(a_1, t_1)(a_2, t_2)...$ reconnu par l'automate temporisé initial A. Ainsi, le test du vide du langage reconnu par un automate temporisé A ou les problèmes d'accessibilité d'une localité dans A peuvent se réduire à un problème d'accessibilité dans son automate de régions $R(A)$. Cela donne un algorithme pour ces deux problèmes qui PSPACE-complet :

Théorème 3.1 Alur and Dill (1990, 1994)

L'accessibilité d'une localité dans les automates temporisés est un problème PSPACE-complet.

Pour éclaircir ces notions, nous allons construire l'automate de régions à partir de l'automate temporisé de Figure 3.1. Figure 3.3 présente l'automate de régions associé. Sur cet exemple, pour dire que la localité s_3 de A est accessible, il faut prouver l'accessibilité de l'un des états (s_3, α) dans l'automate de régions sur Figure 3.3, où α est une région d'horloges.

Dans l'automate de régions, il existe un chemin menant à un état contenant la localité s_3, qui est $(s_0, x = y = 0)a(s_1, 0 = y < x < 1)c(s_3, 0 < y < x < 1)$. Ce chemin nous apprend qu'il existe une exécution dans l'automate temporisé A de la forme $(s_0, v_0)t_1(s_0, v_0 +$

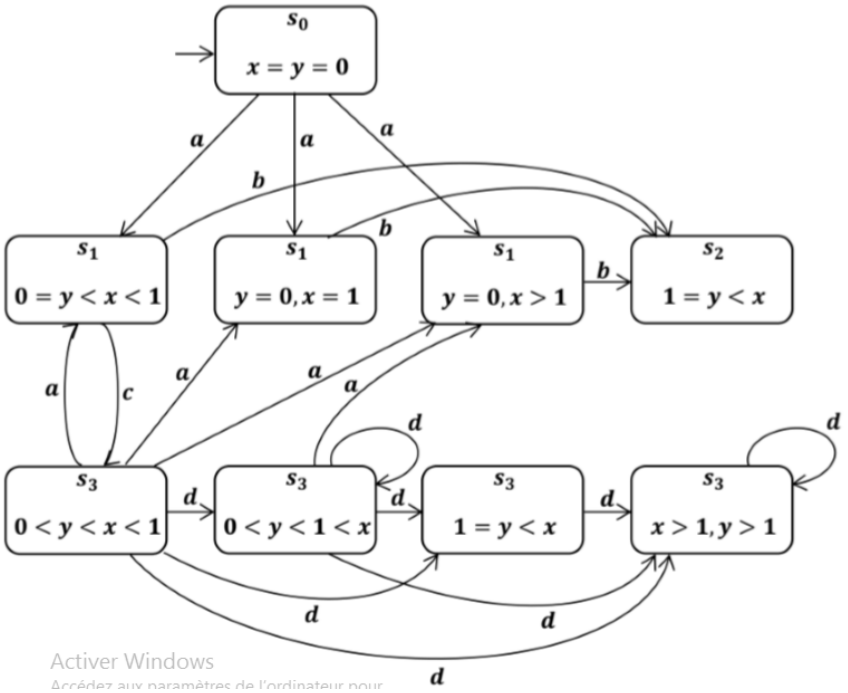


FIGURE 3.3 – Automate de régions associé à l'automate A.

$t_1)a(s_1, v_1)t_2(s_1, v_1 + t_2)c(s_3, v_2)$ qui mène à la localité s_3, où t_1 et t_2 sont des réels positifs. Le calcul de la complexité du problème de l'accessibilité (évoquée dans le théorème précédent) a été fait sur le papier original Alur and Dill (1994).

- Le coté PSPACE-dur parvient de la possibilité de coder, en respectant le mot w, le comportement d'une machine de Turing M qui est bornée linéairement en espace. Donc il y a une possibilité de donner un automate temporisé tel que la valuation d'horloges de cet automate coïncide avec l'état du ruban de M pendant l'exécution. Il faut noter que ce codage peut être effectué avec seulement trois horloges Courcoubetis and Yannakakis (1992).
- Le coté PSPACE-facile parvient de la possibilité de considérer un algorithme non-déterministe qui peut mémoriser l'état symbolique (une localité et une région d'horloges) courant et un état successeur généré de façon non-déterministe.

3.4 Vérification par inclusion de langages temporisés

Dans la section précédente, nous avons montré qu'il est toujours possible d'associer à une exécution dans un automate temporisé un mot temporisé qui est décrit sous forme d'une séquence de paires (a, t) où a représente l'étiquette ou l'action de la transition et t l'instant du temps auquel l'action a a été lancée. Une condition d'acceptation peut être ajoutée au modèle des automates temporisés (localités finales, conditions de Büchi, Müller,...) afin de rendre ce modèle comme une machine qui accepte un langage de mots temporisés. Si les bons comportements d'une spécification d'un système coïncident avec un langage accepté par un automate temporisé, le problème de vérification se ramène alors à étudier la question

suivante sur les langages temporisés : est-ce que l'inclusion du langage du système étudié dans celui de sa spécification est vérifiée ? Dans les systèmes atemporels, on résout ce problème par vérifier que l'intersection entre le langage du système étudié et le complémentaire du langage fourni par la spécification soit vide. Dans le cas des systèmes temporels, on ne peut pas généralement construire un automate temporisé qui reconnait le complémentaire d'un langage accepté par un automate temporisé, ce qui empêche d'appliquer cette méthode. En outre, il a été prouvé dans Alur and Dill (1994) que le problème d'inclusion de langages cité plus haut est indécidable et la vérification de ce genre de propriétés temporisées ne pourra être faite que pour des langages temporisés particuliers, par exemple les langages acceptés par des automates temporisés déterministes.

3.4.1 Logique de temps arborescent

Pour élargir les logiques temporelles par des contraintes quantitatives, on peut

- soit conditionner les modalités classiques de ces logiques par des contraintes,
- soit intégrer des horloges (qui concernent la formule de la propriété) et des opérateurs pour les gérer.

La première possibilité consiste à intégrer une contrainte qui a la forme $\bowtie c$, telle que $\bowtie \in \{=, <, \leq, \geq, >\}$ et $c \in \mathbb{N}$, au niveau de l'opérateur temporel $Until$ noté par $\cup$. Alors on peut dire qu'une formule $\varphi \cup_{<c} \psi$ soit vraie pour une exécution ρ si et seulement s'il existe un état tel que :

- il y amoins de c unités de temps de l'état initial jusqu'à cet état,
- il vérifie ψ et
- tous les états précédents visités pendants l'exécution ρ vérifient φ.

De façon plus générale, il est possible d'intégrer un intervalle $[a; b]$ au niveau d'un opérateur $Until$. Cette manière d'élargir les logiques temporelles est assez classique Koymans (1990). Cette extension a été encore proposée en utilisant CTL pour le temps discret Emerson et al. (1992) et pour le temps dense Alur et al. (1993). Logique Timed CTL (TCTL) a été définit formellement à partir des propositions atomiques (comme des étiquettes sur les états du système à vérifier), des opérateurs booléens $(\wedge, \vee, \neg)$ et des modalités $E\_\cup_{\bowtie c\_}$ et $A\_\cup_{\bowtie c\_}$. Alors on écrit Propriété (1) comme suit :

$$AG(problème \Rightarrow AF_{\leq 5}\ alarme)$$

Une autre approche pour élargir les logiques temporelles par des aspects quantitatifs Alur and Henzinger (1994) consiste à intégrer

- des horloges de formules (l'ensemble de ces horloges est noté H) qui s'incrémentent de façon synchrone avec le temps,
- un opérateur de remise à zéro (in) et
- des contraintes simples $x \bowtie c$ avec $x \in H$.

La remise à zéro suivie d'une contrainte $x \bowtie c$ permet de capter le délai qui sépare les deux états du système. Formellement, une logique $TCTL_h$ a été définie à partir de CTL en ajoutant des contraintes $x \bowtie c$ avec $x \in H$ et l'opérateur $x\ \underline{in}\ \_$. Alors on écrit Propriété (1) comme suit :

$$AG(problème \Rightarrow (x\ \underline{in}\ (AF(x \leq 5 \wedge alarme))))$$

Dans cette formule, l'opérateur $\underline{in}$ remet à zéro l'horloge x quand on rencontre un état qui vérifie "*problème*" et donc on doit vérifier que l'horloge x soit inférieur ou égal à 5 quand on rencontre un état qui vérifie "*alarme*" pour garantir que le délai séparant ces deux états est inférieur ou égal à 5.
Il est clair que tous les opérateurs de $TCTL$ peuvent être exprimés par l'utilisation des horloges de formules. On a l'équivalence suivante pour les formules de $TCTL$ φ et ψ :

$$E(\varphi \cup_{\bowtie c} \psi) \equiv x \ \underline{in} \ E(\varphi \cup (\psi \wedge x \bowtie c))$$

Des propriétés très fines peuvent être exprimées par la logique $TCTL_h$, il a été montré dans le cas du temps dense que cette logique est plus expressive que $TCTL$ Bouyer et al. (2005) : la preuve est que cette formule n'a pas d'équivalent en $TCTL$:

$$x \ \underline{in} \ EF(P_1 \wedge x < 1 \wedge EG(x < 1 \Rightarrow \neg P_2))$$

La formule ci-dessus exprime le fait qu'il est faisable d'atteindre un état vérifiant la propriété P_1 en moins de 1 unité de temps et à partir de cet état il existe une exécution pendant laquelle il n'y a pas d'état vérifiant la propriété P_2 avant que l'horloge x ne soit égale à 1.
On a le résultat suivant :

Théorème 3.2 Alur et al. (1993)

Le model checking des logiques $TCTL$ ou $TCTL_h$ pour les automates temporisés est un problème PSPACE-complet.
Les algorithmes de vérification font l'étiquetage d'un automate de régions étendu avec des horloges particulières qui servent à vérifier les contraintes temporelles dans les formules logiques. Kronos est un outil qui sert à vérifier des formules écrites en $TCTL$ sur des compositions parallèles d'automates temporisés Yovine (1997).

3.4.2 Logique de temps linéaire

Les logiques de temps linéaire peuvent aussi être étendues afin de pouvoir formaliser des propriétés sur les exécutions des automates temporisés. Alors on écrit Propriété (1) comme suit : $G(\textit{problème} \Rightarrow F_{\leq 3} \ \textit{alarme})$. Parmi les logiques temporelles temporisées de temps linéaire, on peut mentionner MTL Koymans (1990); Alur and Henzinger (1993) où la modalité $\cup$ a été étendue par des intervalles, ainsi que MITL Alur et al. (1996) une restriction de MTL qui n'autorise pas la réduction des intervalles à une valeur unique (c'est-à-dire on ne peut pas utiliser la modalité $\cup_{=c}$). Le model ccheking de MTL est un promlème indécidable quand on considère une sémantique pour laquelle l'observation du système est continue : chaque proposition atomique a une valeur de vérité définie sur des intervalles de temps pendant les exécutions. Par contre le model checking est décidable quand on considère une sémantique ponctuelle pour laquelle ces propositions atomiques sont vraies à des instants. Pour MITL, le model checking est un problème EXPSPACE-complet et devient PSPACE-complet si on utilise les modalités "Until" avec les contraintes $< c$ ou $> c$ Alur et al. (1996).
Pour plus de détails sur plusieurs logiques temporelles quantitatives, nous pouvons se référer à Alur and Henzinger (1991); Alur et al. (1993); Alur and Henzinger (1993); Henzinger et al. (1994); Raskin (1999).

3.4.3 Logiques modales avec points fixes

On peut aussi étendre les logiques modales pour énoncer des propriétés temps-réel. Inspirant de la logique d'Hennessy et Milner Hennessy and Milner (1985), on peut décrire le comportement du système par la définition des modalités qui portent sur les étiquettes des transitions du STT. Il y a deux façons pour exprimer la quantification :

— existentielle, écrite sous la forme $< a > \varphi$ pour formaliser "il est possible de tirer une transition a puis de vérifier φ" et

— universelle, $[a]\varphi$ pour formaliser "après toute transition étiquetée par a, φ est vraie".

Pour les transitions de temps on peut utiliser le symbole δ sous la forme $< \delta > \varphi$ pour exprimer qu'il est possible d'attendre un certain temps, sans qu'aucune transition d'action ne soit tirée, jusqu'à ce que φ soit vérifiée. Des opérateurs de point fixe peuvent aussi être utilisés pour formaliser des propriétés qui portent sur des comportements non bornés Larsen (1990); Stirling (2001). Pour mesurer les délais qui séparent les actions du système étudié, des horloges de formule (comme celles de $TCTL_h$) ont été utilisées pour traiter ces aspects quantitatifs.

Le model checking pour les logique modales temporisées avec points fixes est un problème EXPTIME-complet Aceto and Laroussinie (2002).

3.4.4 Automates de test

Un automate de test Aceto et al. (1998) est un automate temporisé T_φ construit pour décrire un scénario d'erreur et que sa vérification peut être effectuée par l'application d'un problème d'accessibilité sur la composition parallèle de l'automate qu'on vérifie avec T_φ. Il est possible aussi d'utiliser une logique modale temporisée pour décrire la propriété de correction et de construire automatiquement l'automate de test associé. Notons que cette approche n'est applicable que sur une catégorie limitée de propriétés Aceto et al. (2003).

3.4.5 Equivalence comportementale et bisimulation

Les équivalences comportementales sont utiles pour comparer des systèmes temporisés. Une classe très utilisée de ces équivalences est la bisimulation forte temporisée. On dit que deux états (s_1, v_1) et (s_2, v_2) sont fortement bisimilaires et on écrit $(s_1, v_1) \approx (s_2, v_2)$ si et seulement si :

— pour toute transition $(s_1, v_1)a(s'_1, v'_1)$ avec $a \in Act$,il existe une transition $(s'_2, v_2)a(s'_2, v_2)$ telle que$(s'_1, s'_1) \approx (s'_2, s'_2)$,et inversemant

— pour toute transition $(s_1, v_1)t(s'_1, v'_1)$ avec $t \in R_{\geq 0}$, il existe une transition $(s_2, v_2)t(s'_2, v'_2)$ telle que $(s'_1, v'_1) \approx (s'_2, v'_2)$, et inversement.

La définition ci-dessus de l'équivalence comportementale est très forte, et en particulier, pour une relation de bisimulation qui respecte les états initiaux et finals, on dit que deux automates temporisés sont fortement bisimilaires s'ils acceptent les mêmes mots temporisés.

Il faut noter que la bisimulation temps-abstrait (utilisée pour la correction de l'automate de régions) est largement moins forte que la bisimulation forte temporisée, parce que dans cette

dernière, les durées d'attente pour quitter un état doivent être les mêmes à partir de deux états équivalents, ce qui n'existe pas dans la bisimulation temps-abstrait.
Prendre une décision si deux automates temporisés sont fortement bisimilaires est un problème EXPTIME-complet Laroussinie and Schnoebelen (2000).

3.5 Extensions des automates temporisés : langage de haut niveau

Avoir des langages de description de haut niveau est très utile pour spécifier plus facilement les systèmes. Pour cela, plusieurs extensions du modèle des automates temporisés ont été considérées. Pendant la présentation de ces extensions, nous nous intéresserons aux questions suivantes :

- l'extension préserve-t-elle la décidabilité de l'accessibilité ?
  si la vérification des systèmes qu'on modélise est notre objectif, préserver la décidabilité du modèle est un point essentiel.
- l'extension ajoute-t-elle de l'expressivité ?
  il est important d'avoir des modèles qui permettent de décrire facilement plus de systèmes.
- l'extension ajoute-t-elle de la concision aux modèles qu'on peut construire ?
  un autre point important est la facilité de la modélisation : un modèle plus petit est généralement plus lisible et plus facile à modifier.

3.5.1 Contraintes diagonales

Les contraintes sur les horloges autorisées dans le modèle des automates temporisés qu'on a présenté sont très simples et permettent de faire seulement une comparaison entre la valeur d'une horloge et une constante. Un autre type de contraintes, appelées contraintes diagonales, a été évoqué dans le papier original Alur and Dill (1994) et qui permettent d'effectuer des comparaisons du type $x - y \bowtie c$ où x et y sont des horloges, $\bowtie$ est un symbole de comparaison et c est une constante entière. Cette extension des automates temporisés à les propriétés suivantes :

- L'accebilité dans les automates temporisés avec contraintes diagonales est un problème PSAPACE-complet Alur and Dill (1994).
- l'utilisation des contraintes diagonales n'augmente pas l'expressivité des automates temporisés Bérard et al. (1998).
- l'utilisation des contraintes diagonales ajoute de la concision Bouyer and Chevalier (2005).

Le problème d'accessibilité de cette extension était déjà prouvé décidable dans le papier original Alur and Dill (1994), il a été aussi résolu par la construction d'un automate de régions, qui raffine celui qu'on a présenté dans Section 3.3 et la complexité reste toujours la même. Pour la deuxième propriété, cette extension a été démontrée plus expressive dans Bérard et al. (1998). Le principe de cette démonstration est de remplacer une à une les contraintes diagonales et de construire un automate temporisé sans contraintes diagonales équivalent selon la relation de bisimulation forte temporisée. Figure 3.4 explique la manière de remplacer une contrainte diagonale (sur cette figure, l'objectif est de retirer la contrainte diagonale

$x - y \leq c$ où c est un entier positif). Le principe de base est comme suit : l'écoulement du temps n'influe pas sur la valeur de vérité d'une contrainte diagonale $x - y \leq c$, mais cette valeur peut être changée si l'une des deux horloges impliquées dans la contrainte diagonale est réinitialisée. Donc, deux copies de l'automate initial ont été construites, la contrainte $x - y \leq c$ est vérifiée dans une copie, par contre dans l'autre, c'est la contrainte $x - y > c$ qui est vérifiée. Quand l'horloge x ou y est réinitialisée, nous allons selon la valeur de l'autre horloge vers l'une ou l'autre des copies. Par exemple, si l'horloge y est réinitialisée, alors si $x \leq c$ on se trouve dans la copie $x - y \leq c$, par contre si $x > c$ on se trouve dans l'autre copie $x - y > c$. Cette transformation décrite ci-dessus donne une taille double de l'automate, par conséquent on a une construction exponentielle pour transformer toutes les contraintes diagonales. Comme les automates temporisés avec contraintes diagonales sont exponentiellement plus concis que les automates temporisés standards Bouyer and Chevalier (2005), on ne peut pas éviter ce coût exponentiel. Ça veut dire qu'on peut modéliser des systèmes avec des automates temporisés avec contraintes diagonales exponentiellement plus petits que le seraient des automates temporisés standards équivalents.

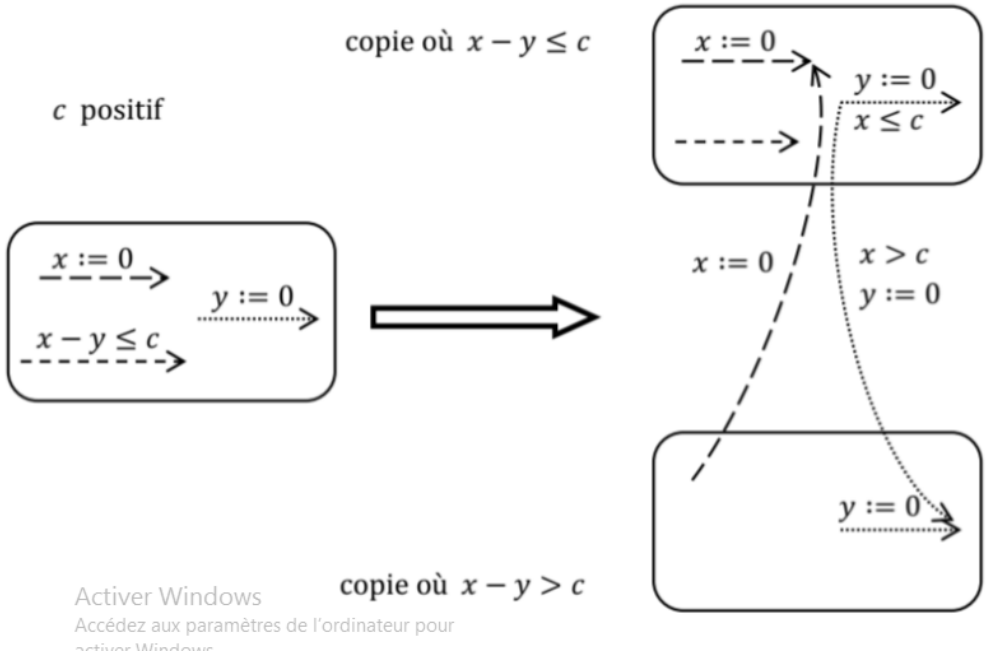


FIGURE 3.4 – Suppression des contraintes diagonales.

3.5.2 Contraintes additives

Les automates temporisés peuvent aussi être étendus par d'autres types de contraintes, dans cette partie nous allons considérer les contraintes dites additives qui ont la forme $x+y \bowtie c$ où x et y sont des horloges, $\bowtie$ est un symbole de comparaison et c est un entier positif. Les auteurs de Bérard and Dufourd (2000) ont étudié cette extension des automates temporisés qui acceptent des langages non reconnus par des automates temporisés standards. Par exemple, l'automate de Figure 3.5 accepte un langage qu'aucun automate temporisé standard ne peut accepter : les actions a sont exécutées aux dates $1/2,\ 3/4,\ 7/8,\ 15/16, ...$

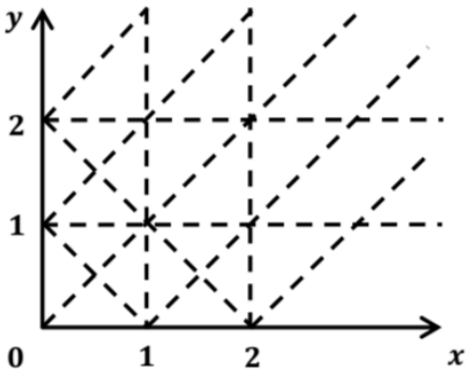


FIGURE 3.6 – Index de régions sur deux horloges pour les automates temporisés avec contraintes additives.

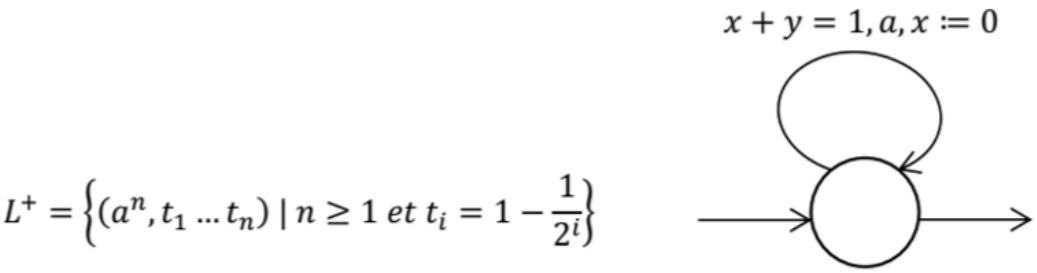


FIGURE 3.5 – Suppression des contraintes diagonales.

Les propriétés satisfaites par ce modèle d'automates temporisés avec contraintes additives sont Bérard and Dufourd (2000) :

- Le problème d'accessibilité dans les automates temporisés avec contraintes additives utilisant deux horloges est décidable.
- Le problème d'accessibilité dans les automates temporisés avec contraintes additives utilisant au moins quatre horloges est indécidable.

La décidabilité de ce modèle dans le cas de deux horloges découle d'une construction étendue d'un automate de régions, l'index de régions utilisé est un raffinement de l'index de régions que nous avons vu dans la section 3.3. Figure 3.6 illustre cette construction. Dans le cas où il y a quatre horloges, l'accessibilité devient indécidable. Dans la preuve de ce dernier cas, le petit automate de Figure 3.5 a été utilisé à plusieurs reprises.

Remarque 3.1

Concernant le cas de trois horloges dans les automates temporisés avec contraintes additives, et comme il est impossible de leur construire un automate de régions fini de la même manière que pour les automates temporisés standard, la preuve de l'accessibilité reste toujours un problème ouvert.

3.5.3 Actions internes

L'élimination des actions internes, dites les ε-transitions, dans les automates finis n'a aucune influence sur le langage reconnu par ces structures, par conséquent ce type d'actions n'ajoute pas de l'expressivité au modèle des automates finis Hopcroft et al. (2001). Par contre dans le cadre temporisé, ces actions silencieuses ajoutent de l'expressivité au modèle et préservent toujours la décidabilité de l'accessibilité (on construit de la même manière l'automate de régions) Bérard et al. (1998).

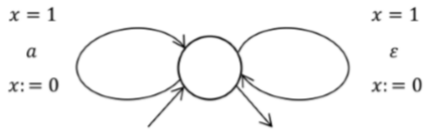


FIGURE 3.7 – Un langage non reconnu par les automates temporisés standards.

L'automate temporisé de Figure 3.7 accepte un langage qui n'est accepté par aucun autre automate temporisé standard. Ce langage comporte les mots temporisés sur une lettre a tels que toutes les dates sont des entiers : à chaque unité de temps cet automate fournit deux possibilités de franchissements, soit la transition étiquetée par a est choisie, soit c'est l'autre qui est étiquetée par ε.

3.5.4 Mises à jour

Dans les automates temporisés classiques, la seule opération permise sur les horloges est la remise à zéro. Il est utile de prendre en charge d'autres opérations un peu plus compliquées sur les horloges. Les deux formes des opérations des mises à jour sont $x :\bowtie c$ et $x :\bowtie y + c$ où x et y sont des horloges, $\bowtie$ est un symbole de comparaison et c est un entier. Par exemple, la mise à jour

- $x :\leq c$ affecte à l'horloge x une valeur inférieure ou égale à la constante c de façon non déterministe,
- $x := y - 1$ affecte à l'horloge x la valeur de l'horloge y décrémentée de 1,
- $x := 0$ correspond aux remises à zéro classiques.

Une étude de ce type d'automates temporisés qui utilise les mises à jour a été faite dans Bouyer et al. (2004). Comme il est possible d'incrémenter, décrémenter ou encore remettre à zéro toute horloge, ces types puissants des mises à jour rendent l'accessibilité du modèle général indécidable. Cependant, l'accessibilité a été montrée décidable pour plusieurs sous-classes de ce modèle, voici quelques résultats pour ces sous-classes présentés dans Bouyer et al. (2004) : l'accessibilité dans les automates temporisés

- avec mises à jour de la forme $x := c$ est décidable.
- avec auto-incrémentation (c-à-d qui utilisent des mises à jour de la forme $x := x + 1$) et sans contrainte diagonale est décidable.
- avec auto-incrémentation et contraintes diagonales est indécidable.
- avec auto-décrémentation (c-à-d qui utilisent des mises à jour de la forme $x := x - 1$) est indécidable.

La preuve de la décidabilité est toujours basée sur une construction d'automates de régions avec des index de régions assez fins et qui raffinent ceux qu'on a vu dans la section 3.3. Figure 3.8 présente un index de régions concernant un automate temporisé qui utilise des contraintes d'horloges diagonales $x - y < 1, y > 1$ et les mises à jour $x := 0, y := 1$. La répartition décrite par des lignes en pointillés gras représente l'index classique de régions. Mais quand on utilise la mise à jour $y := 1$, cet index sera incompatible avec le modèle étudié parce que (entre autres) la mise à jour $y := 1$ rend la région d'horloges grisée sur Figure 3.8 fait un chevauchement de plusieurs régions (sans vérifier une condition de la relation de bisimulation temps-abstrait). Alors il faut répartir encore une fois cet espace de valuations d'horloges en insérant une demi droite $x = 2$ (présentée par une ligne en pointillés fins sur Figure 3.8).
Il faut noter qu'il est possible de transformer les classes d'automates temporisés avec mises à jour qui ont été prouvées décidables en automates temporisés standards avec actions internes équivalents selon une relation de bisimulation faible mais qui préserve les mêmes langages Bouyer et al. (2004). En outre, ces automates avec mises à jour sont aussi exponentiellement plus concis que les automates temporisés standards Bouyer and Chevalier (2005).
En fin, on peut dire que ce type d'opérations sur les horloges appelées "mises à jour" produit des macros très pratiques pour modéliser les systèmes réels. Par exemple on peut citer parmi les systèmes qui se modélisent de manière naturelle avec ce type d'opérations : les problèmes d'ordonnancement Fersman et al. (2002).

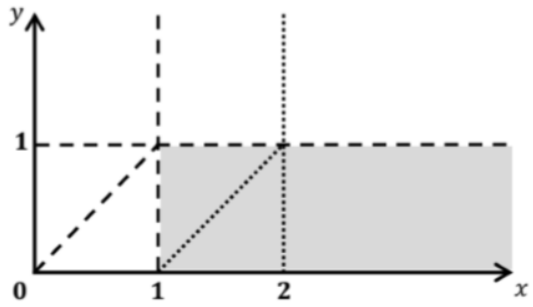


FIGURE 3.8 – Une nouvelle répartition de régions selon les contraintes $\{x - y < 1, y > 1\}$ et les mises à jour $\{x := 0, y := 1\}$.

3.5.5 Automates hybrides linéaires

Les automates hybrides linéaires étendent les automates temporisés classiques par :
- des contraintes linéaires générales, par exemple $2x - y + 3z < 85$,
- des mises à jour plus complexes que les opérations simples de remise à zéro : nous pouvons appliquer des fonctions affines sur les horloges de l'ensemble Z sur chaque transition,
- les horloges ont des pentes différentes selon les états : pour cela les horloges dans cette extension s'appellent variables dynamiques.

On a vu ci-dessus que les extensions des automates temporisés avec les contraintes additives et avec les mises à jour étendues rendent l'ensemble des problèmes de vérification indécidables. Même chose pour les variables avec des pentes différentes : le problème d'accessibilité dans

cette extension, où une variable peut avoir deux pentes différentes, est indécidable. Un survey sur ces notions et même sur des sous-classes décidables des automates hybrides linéaires est disponible dans Henzinger et al. (1998) et Raskin (2005).
En outre, il faut signaler l'importance de la tâche qui consiste à étudier et rechercher des sous-classes des automates hybrides linéaires. Par conséquent, cette tâche nous permet de distinguer :

— des familles décidables comme les automates rectangulaires initialisés Henzinger et al. (1998), ou bien
— des familles sur lesquelles on peut appliquer des optimisations.

En plus, il existe quelques sous-classes qui ont particulièrement une importance dans le monde pratique, par exemple pour la modélisation de protocoles de télécommunication, on peut utiliser la famille des p-automates Bérard and Fribourg (1999), ou bien utiliser la famille des automates à chronomètres, c'est-à-dire, ceux qui ont des variables avec des pentes dans $\{0, 1\}$, pour la modélisation de problèmes d'ordonnancement avec préemption.
Donc concernant le model-checking et pour vérifier des propriétés d'accessibilité dans les automates hybrides linéaires, Il n'existe pas d'algorithmes générales. Mais il y a une possibilité de proposer soit des semi-algorithmes qui peuvent ne terminer pas le calcul, soit des approximations qui assurent la terminaison du calcul mais qui ne donnent pas toujours un résultat (par exemple, la réponse peut être "ne sait pas" ou "le système vérifie l'ensemble vide").
Le point essentiel de ces approches est basé sur l'utilisation des contraintes linéaires afin de déterminer les ensembles d'états du système à vérifier Alur et al. (1995). Les bibliothèques de calcul sur polyèdres ont été utilisées pour implanter des semi-algorithmes qui vérifient l'accessibilité. HyTech est un outil qui peut calculer étape par étape des espaces d'états pour ces automates hybrides linéaires. Comme il peut aussi faire des calculs de points fixes (la terminaison est non sûre). Une description bien détaillée avec des exemples sont disponibles dans Henzinger et al. (1997).

3.6 Les sous-classes des automates temporisés : entre l'expressivité et l'efficacité

Cette partie est réservée pour présenter quelques restrictions sur les automates temporisés. En générale l'idée derrière ces restrictions est de se limiter à des modèles moins expressifs pour assurer certaines propriétés ou bien rendre les algorithmes de décision plus efficaces. Donc l'objectif de cette démarche est de chercher un bon compromis entre une bonne expressivité du modèle et une algorithmique efficace.
Il faut noter que les deux raisons essentielles qui ont donné de l'importance à cette approche de la restriction des automates temporisés sont la composition parallèle (le même problème a été dans le cadre classique atemporel) et de la temporisation (l'intégration des horloges dans le modèle), ces deux aspects ont rendu la tâche de vérification de ces modèles plus compliquée, même théoriquement, la vérification d'une composition parallèle d'automates temporisés a le même degré de complexité que la vérification d'un seul automate temporisé (la complexité est toujours PSPACE). Malgré qu'il existe de bonnes structures de données, appelées les BDD (pour : Binary Decision Diagram), pour coincer l'explosion combinatoire due à la composition parallèle et qu'il existe d'autres structures de données, appelés les DBM

(pour : Difference Bound Matrice), pour faciliter la représentation des contraintes temps-réel, il n'existe pas de structures de données qui nous donnent une solution efficace pour remédier à la combinaison de ces deux types d'explosion.

3.6.1 Automates "event-recording"

Dans ce modèle, chaque lettre a de l'alphabet des actions (qui sont des étiquettes pour les transitions de l'automate temporisé) a sa propre horloge x_a telle que cette horloge est remise à zéro si une transition étiquetée par l'action a est tirée. Ce modèle impose aussi que l'automate ne peut avoir que ce type d'horloges. Par conséquent, pendant le tir de toute transition, il n'y a que l'horloge associée à l'action de cette transition qui sera remise à zéro. Donc la valeur de l'horloge x_a coïncide exactement avec le délai écoulé depuis la dernière exécution de l'action propriétaire a.
Le modèle des automates event-recording a été proposé dans Alur and Dill (1994), ce papier a présenté aussi une autre version qui s'appelle les automates "event-predicting" où les horloges indiquent le temps restant avant le prochain lancement de l'action propriétaire a et non pas le temps écoulé après la dernière exécution de l'action propriétaire a. Des valeurs négatives ont été affectées aux horloges à chaque fois pour pouvoir manipuler cette idée.
Cette sous-classe n'a pas apporté en fait d'améliorations concernant la complexité, donc le test du vide reste toujours PSPACE-complet et même chose pour model-checking. En contrepartie, cette sous-classe d'automates temporisés est déterminisable, les langages de ces modèles sont fermés par complémentaire et par conséquent le test d'inclusion entre deux langages devient décidable.

3.6.2 Automates à une ou deux horloges

L'accessibilité pour les automates temporisés avec plus de trois horloges a été montré dans Courcoubetis and Yannakakis (1992) qu'elle est un problème PSPACE-complet. De cet effet, l'étude des automates avec une ou deux horloges a gagné de l'importance dans le but de développer des algorithmes plus efficaces.
Les automates à une horloge (AT1H) forment une classe particulièrement importante car elle vérifie les propriétés suivantes Laroussinie et al. (2004) :

- l'accessibilité d'une localité est un problème NLOGSPACE-complet, c-à-d elle a la même complexité que celle dans un graphe non temporisé,
- model-checking pour les AT1H qui utilise des formules de $TCTL_{\leq,\geq}$ possède un algorithme en temps polynomial (donc cette restriction de $TCTL$ n'autorise pas les contraintes " $= c$" dans les modalités "$Until$"),
- model-checking de $TCTL$ est un problème PSPACE-complet.

Le principe de l'algorithme en temps polynomial pour model-checking de la restriction $TCTL_{\leq,\geq}$ appliqué sur un automate AT1H A est de déterminer un ensemble d'intervalles $Sat[s,\varphi]$ pour chaque sous-formule φ de la formule globale $TCTL_{\leq,\geq}$ tel que cet ensemble, défini par $Sat[s,\varphi] = \cup_i \langle \alpha_i, \beta_i \rangle$, contient les positions (s,x) qui vérifient φ avec $\langle \in \{[,(\}, \rangle \in \{],)\}$ et $\alpha_i, \beta_i \in \mathbb{N} \cup \{\infty\}$. Par conséquent, il est possible de montrer que le nombre d'intervalles de $Sat[s,\varphi]$ est borné par $2 \cdot |\varphi| \cdot |T_A|$.
Prenons l'exemple $E\varphi \cup_{\leq c} \psi$ pour décrire brièvement la procédure d'étiquetage. Après le calcul de $Sat[s,\varphi]$ et $Sat[s,\psi]$, on peut construire d'une manière simple un automate de

régions avec une taille polynomiale : les configurations sont des paires (s, γ) où γ est un intervalle (région) homogène pour les valeurs de vérité des sous-formules φ et ψ et pour la satisfaction des gardes étiquetant les transitions de A.
À chaque configuration (s, γ) et pour chaque paire (s, x) telle que $x \in \gamma$, une fonction $f_{s,\gamma}$ est associée pour donner la durée minimale pendant la prise d'un chemin vérifiant φ et avant que ψ soit satisfaite. Ces fonctions $f_{s,\gamma}$ ont les trois formes particulières suivantes :

- $f_{s,\gamma}$ est constante sur γ si le plus court chemin qui mène à ψ possède une transition de réinitialisation avant toute transition de temps,
- $f_{s,\gamma}$ est décroissante (avec une pente égale à -1) quand le plus court chemin possède une transition de temps avant toute réinitialisation, ou
- $f_{s,\gamma}$ combine les deux formes citées plus-haut, constante pour le premier intervalle de γ et après décroissante.

Pour définir ces fonctions $f_{s,\gamma}$, il faut d'abord déterminer leurs valeurs aux bornes de γ et ensuite utiliser un algorithme qui calcule le plus court chemin comme l'algorithme de Bellman-Ford Cormen et al. (2001). En fin il faut construire un ensemble d'intervalles qui comportent des positions avec une distance à ψ inférieure à c.
Concernant les propriétés des langages temporisés, les automates temporisés à une horloge offrent la décidabilité pour le test d'inclusion de langages temporisés ne comprenant que les mots finis Ouaknine and Worrell (2004), cependant ce test reste toujours indécidable pour les mots infinis Abdulla et al. (2005).
Pour les automates temporisés à deux horloges, un saut important de complexité a été remarqué : l'accessibilité d'une localité est un problème NP-dur et model-checking de CTL (c'est-à-dire les formules ne contiennent pas de contraintes temps-réel) est dès maintenant PSPACE-complet Laroussinie et al. (2004).

3.6.3 Automates à temps discret

Il est possible d'utiliser les entiers naturels comme domaine de temps au lieu d'utiliser un domaine dense. Pour la majorité des problèmes qu'on a vu dans les automates temporisés et ses restrictions, le changement du domaine de temps n'apporte pas une grande modification sur la complexité des algorithmes de vérification pour ces modèles, l'accessibilité et le model checking de $TCTL$ restent toujours PSPACE-complet. Donc il faut chercher des modèles plus simples pour avoir des algorithmes plus efficaces avec ce changement du temps. Dans la littérature, il existe plusieurs propositions.
En premier lieu, on va considérer des modèles tels qu'une durée d'une unité de temps est associée à chaque transition de l'automate. Plusieurs travaux ont étudié cette notion simple de temporisation. Model-checking de $TCTL$ est polynomial pour cette restriction Emerson et al. (1992) et même pour les modèles où chaque transition dure 0 ou 1 unité de temps Laroussinie et al. (2003). Ce résultat concernant le model-checking pour $TCTL$ est unique. Si on utilise la logique $TCTL_h$ avec horloges de formules, on aura des problèmes PSPACE-complet.
Une extension spontanée du modèle précédent est de considérer des transitions avec des durées entières et obtenir un modèle défini comme des automates à coûts. Dans ce cadre, il est possible d'avoir des algorithmes polynomiaux de model-checking pour $TCTL_{\leq,\geq}$, cependant model-checking pour $TCTL$ est un problème Δ_2^p-complet Laroussinie et al. (2006). Cette extension est implantée dans l'outil TSMV Markey and Schnoebelen (2004) qui est une

version étendue de l'outil NuSMV Cimatti et al. (2000).
À la fin pour avoir des algorithmes polynomiaux pour $TCTL_{\leq,\geq}$, les automates temporisés à une horloge à valeur entière peuvent être utilisés, cependant pour $TCTL$, le problème devient PSPACE-complet Laroussinie et al. (2006).

3.7 La vérification en pratique

Afin de vérifier les automates temporisés, il est possible d'utiliser quelques outils tels qu'Uppaal et Kronos qui implémentent des algorithmes de vérification qu'on va décrire dans cette section. Comme la notion de région, qui regroupe les valuations d'horloges équivalentes, est assez fine et contribue à la croissance de l'inefficacité des algorithmes, la construction de l'automate de régions (que nous avons présenté dans Section 3.3) n'était pas utile pour la pratique. Donc une deuxième représentation symbolique, appelée "zone", de la relation d'équivalence a été utilisée par les outils qui se basent sur des algorithmes à la volée. L'intérêt de l'utilisation des zones en pratique est au détriment de l'abstraction supplémentaire sur les comportements. Pour analyser les systèmes à la volée, on peut distinguer deux familles principales de semi-algorithmes :

- la première famille utilise une analyse en avant : à partir des configurations initiales, cette approche calcule itérativement les successeurs et teste si la configuration qu'on veut atteindre est calculée ou pas.
- La deuxième utilise une analyse en arrière : on peut dire que c'est l'approche duale de la première, parce qu'elle calcule itérativement les prédécesseurs des configurations qu'on veut atteindre et teste si une configuration initiale se trouve parmi l'ensemble des prédécesseurs calculés.

Ces méthodes peuvent être appliquées sur l'ensemble des modèles états-transitions afin d'analyser les différents systèmes, voir Alur et al. (1995) par exemple pour une application sur les automates hybrides.
Avant de présenter ces méthodes d'analyse, on va exposer d'abord une notion très utilisée pour la vérification des systèmes temporisés qui est la représentation symbolique.

3.7.1 Les zones : une représenation symbolique pour l'implémetabilité de la vérification

Afin de vérifier un automate temporisé qui possède un ensemble infini de configurations, il est donc nécessaire d'avoir une représentation symbolique pour pouvoir manipuler ces ensembles infinis de configurations. Les zones sont la représentation symbolique la plus utilisée : chaque zone représente un ensemble de valuations d'horloges qu'on obtient selon une conjonction de contraintes temporelles simples $x \bowtie c$ ou encore $x - y \bowtie c$ où x et y sont des horloges, $\bowtie$ est un symbole de comparaison et c est un entier. Dans les deux approches d'analyses en avant et en arrière, les algorithmes vont manipuler les paires (s, Z) où s est une localité de l'automate temporisé et Z une zone de valuations d'horloges.
Sur les zones, on peut effectuer plusieurs opérations :

- le futur d'une zone Z est défini par $\overrightarrow{Z} = \{v + t | v \in Z \ et \ t \in \mathbb{T}\}$ où $\mathbb{T}$ est un domaine de temps,
- le passé d'une zone Z est défini par $\overleftarrow{Z} = \{v - t | v \in Z \ et \ t \in \mathbb{T}\}$,

— l'intersection de deux zones Z et Z' est définie par $Z \cap Z' = \{v | v \in Z \text{ et } v \in Z'\}$,
— la remise à zéro d'un ensemble d'horloges Y d'une zone Z est définie par $Z[Y] = \{v[Y] | v \in Z\}$ et
— la relâche de Y de Z est définie par $Z[Y]^{-1} = \{v | v[Y] \in Z\}$.

Des formules du 1er ordre sont appliquées sur les zones pour définir ces opérations qui préservent les zones.
Maintenant, nous décrivons la méthode la plus simple qui est l'analyse en arrière. Ensuite nous allons présenter la méthode d'analyse en avant qui est favorisée par la plupart des algorithmes malgré la difficulté de sa mise en œuvre.

3.7.2 L'approche d'analyse en arrière

L'analyse en arrière se fait de la manière suivante : à partir des états qu'on veut atteindre, on calcule les états prédécesseurs en un pas, ensuite ceux en deux pas, ... et à chaque étape on teste si l'état initial est parmi les états calculés. Si c'est le cas, cela veut dire qu'on peut atteindre les états recherchés à partir de l'état initial. Si on a terminé le calcul et ce n'est pas le cas, cela veut dire qu'on ne peut jamais atteindre ces états recherchés. Le principe des méthodes basées sur l'analyse en arrière est illustré sur Figure 3.9.

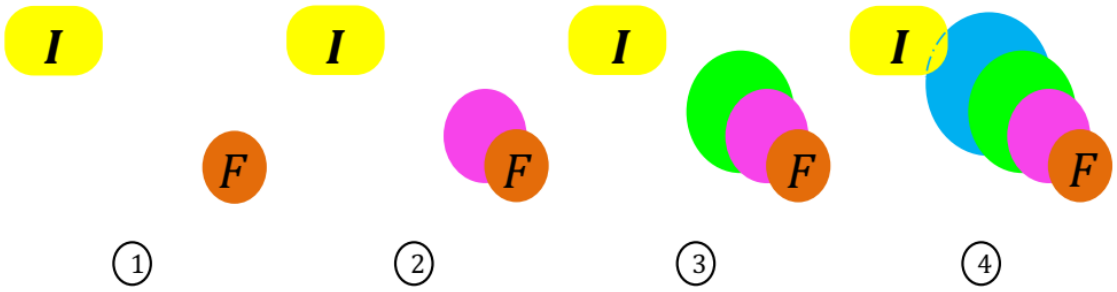


FIGURE 3.9 – Le calcul, pas à pas, des prédécesseurs de l'état final dans l'analyse en arrière.

L'utilisation des zones facilite le calcul d'un pas de l'analyse en arrière. Si on prend une transition de l'automate temporisé $t = sG, a, Ys'$ et si Z' est une zone, alors l'objectif est de calculer l'ensemble des prédécesseurs en un pas de (s', Z') suivant la transition t qui est exactement l'ensemble des configurations (s, v) où v est une valuation d'horloges appartenant à la zone $Z = \overleftarrow{g \cap (Z' \cap Y = 0)[Y]^{-1}}$.
La terminaison du calcul itératif de l'analyse en arrière est une propriété très importante. La preuve de cette propriété est basée sur le fait qu'une zone se compose en réalité de plusieurs régions d'horloges. De ce fait il est facile de montrer que si Z' est une zone, alors la zone Z qu'on vient de décrire est elle-même une union de régions. Comme il a été montré que le nombre de régions associées à un automate temporisé est fini, alors le nombre de paires (s, Z) qu'on est en train de calculer dans une analyse en arrière est lui-même fini.
Pratiquement, l'analyse en avant est implémentée dans la plupart des outils malgré les indéniables avantages de l'analyse en arrière. Les raisons peuvent se résumer en :

— l'analyse en avant peut atteindre uniquement les configurations du système qui ont un sens parce qu'elles sont atteignables par des exécutions réelles.

— l'analyse en arrière ne convient pas avec l'analyse des systèmes qui utilisent des structures de données de haut niveau par exemples les variables entières bornées ou bien les morceaux de programmes C, ... On trouve ces structures de données dans des outils qui implémente l'analyse en avant, par exemple l'outil Uppaal qui va être présenté dans la section 3.8 suivante.

3.7.3 L'approche d'analyse en avant

L'analyse en avant se fait de la manière suivante : à partir de l'état initial, on calcule les états successeurs en un pas, ensuite ceux en deux pas, ... et à chaque étape on teste si les états recherchés sont parmi les états calculés. Si c'est le cas, cela veut dire qu'on peut atteindre certains de ces états recherchés à partir de l'état initial. Si on a terminé le calcul et ce n'est pas le cas, cela veut dire qu'on ne peut jamais atteindre ces états recherchés. Le principe des méthodes basées sur l'analyse en avant est illustré sur Figure 3.10.
L'utilisation des zones facilite le calcul d'un pas de l'analyse en avant. Si on prend une transition de l'automate temporisé $t = sG, a, Ys'$ et si Z est une zone, alors l'objectif est de calculer l'ensemble des successeurs en un pas de (s, Z) suivant la transition t qui est exactement l'ensemble des configurations (s', v') où v' est une valuation d'horloges appartenant à la zone $Z' = (g \cap \overrightarrow{Z})[Y]$.

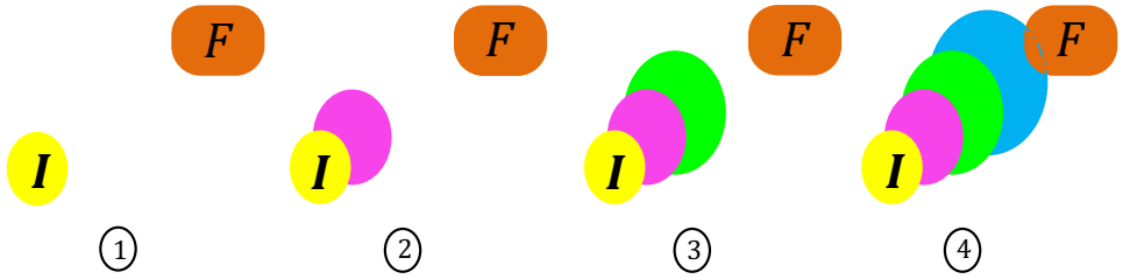


FIGURE 3.10 – Le calcul, pas à pas, des successeurs de l'état initial dans l'analyse en avant.

Généralement le calcul itératif de l'analyse en avant ne se termine pas, contrairement au calcul de l'analyse en arrière. Figure 3.11 et Figure 3.12 présentent un automate temporisé qui illustre cet inconvénient de l'analyse en avant. La valeur de l'horloge x de cet exemple s'incrémente avec une unité de temps pour chaque itération du calcul en avant, par conséquent ce calcul ne va jamais se terminer.
Généralement un opérateur d'abstraction est utilisé à chaque étape du calcul afin de faire face à ce problème de terminaison. Cette opération d'abstraction s'appelle normalisation ou encore extrapolation. Soit k la plus grande constante avec laquelle une horloge de l'automate temporisé a été comparée dans les contraintes. Maintenant soit Z une zone, l'extrapolation de Z par rapport à k est la plus petite zone comprenant Z et qui est définie par des contraintes qui utilisent que des constantes entre $-k$ et $+k$. Cette opération d'extrapolation a comme objectif : Comme toutes les contraintes d'horloges de l'automate temporisé sont bornées par k, alors ce qui est important est de savoir uniquement que la valeur d'une horloge a dépassé la borne k car on n'a pas vraiment besoin de sa valeur exacte. Notons que l'utilisation de cette opération d'extrapolation pendant toute l'analyse en avant assure la terminaison du

calcul itératif parce que le nombre de zones définies par des contraintes qui n'utilisent que des constantes de l'intervalle $[-k, +k]$ est fini.
En outre, cette approche d'analyse en avant peut engendrer un autre problème à chaque itération du calcul : une sur-approximation calculée des états accessibles. Donc ce calcul itératif peut nous retourner des états déclarés comme accessibles par contre en réalité ils ne le sont pas.
Il a été montré dans Bouyer (2004) que cette approche d'analyse en avant est correcte si on considère les automates temporisés sans contraintes diagonales, par contre elle est incorrecte pour la classe des automates avec contraintes diagonales.

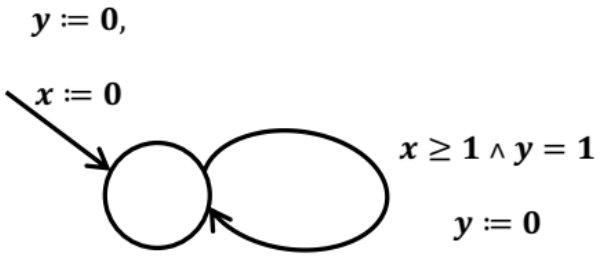


FIGURE 3.11 – Un automate temporisé qui illustre la non terminaison de l'analyse en avant.

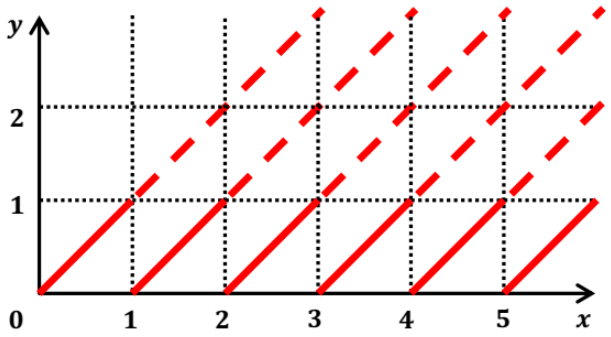


FIGURE 3.12 – Le calcul itératif en avant associé.

3.7.4 Les DBM : une structure de données pour une représentation des zones

Une DBM (Difference Bound Matrice) est une structure de données classique qui représente des systèmes de contraintes de différences Cormen et al. (2001). En outre cette notion de DBM a gagné un intérêt particulier dans la vérification des systèmes temporisés parce qu'elle permet une représentation adéquate d'un concept assez demandé par les outils de vérification qui est les zones. Elle a été proposée dans Berthomieu and Menasche (1983) pour l'analyse des réseaux de Petri temporels, ensuite elle est vivement utilisée pour analyser les automates temporisés Dill (1989).
Soit n la cardinalité de l'ensemble des horloges utilisées, une DBM M est une matrice carrée de taille $(n+1)\times(n+1)$. Généralement, les coefficients de M sont des paires $(m, \prec)$ telles que m est un entier et le symbole $\prec \in \{<, \leq\}$. Soient $M = (m_{i,j})_{0\leq i,j\leq n}$, $\{x_i | 1 \leq i \leq n\}$ l'ensemble

d'horloges et x_0 une horloge fictive qui vaut toujours 0, alors pour chaque contrainte de la forme $x_i - x_j \leq m_{i,j}$ on peut directement acquérir le coefficient $m_{i,j}$ de la DBM. Cependant, le coefficient $m_{i,0} = 7$ dans le cas d'une contrainte de la forme $x_i \leq 7$ parce que cette contrainte peut être facilement réécrite sous la forme $x_i - x_0 \leq 7$.

L'exemple suivant concerne une DBM définie par les contraintes $(x_1 \geq 2) \wedge (x_2 \leq 4) \wedge (x_1 - x_2 \leq 3)$ et illustrée sur Figure 3.13

$$\begin{array}{c} \\ x_0 \\ x_1 \\ x_2 \end{array} \begin{array}{c} \begin{array}{ccc} x_0 & x_1 & x_2 \end{array} \\ \begin{pmatrix} \infty & -2 & \infty \\ \infty & \infty & 3 \\ 4 & \infty & \infty \end{pmatrix} \end{array}$$

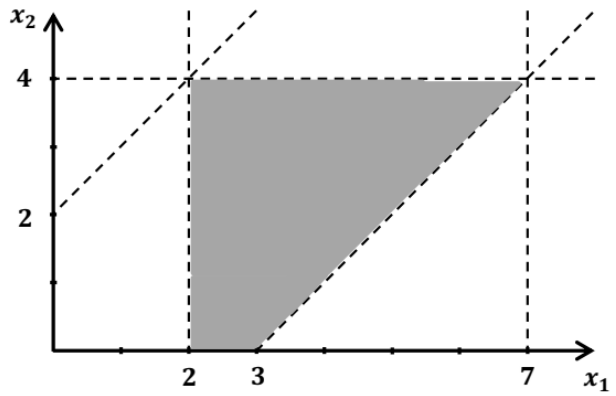


FIGURE 3.13 – Une zone définie par les contraintes $(x_1 \geq 2) \wedge (x_2 \leq 4) \wedge (x_1 - x_2 \leq 3)$.

Si un coefficient est égale à $+\infty$, cela veut dire qu'il n'y a pas une contrainte sur la différence entre les deux horloges liées au coefficient. La valeur du coefficient $m_{0,1} = -2$ est déterminée de la contrainte $x_1 \geq 2$ qui doit être réécrite sous la forme d'une autre contrainte équivalente en utilisant l'horloge fictive $x_0 - x_1 \leq -2$.

Toute DBM caractérise une zone, et toute zone peut être caractérisée par une ou plusieurs DBM. Dans la DBM ci-dessus, si on change la valeur $+\infty$ du coefficient $m_{1,0}$ par 9, cela ne change pas la zone dessinée et on obtient alors une deuxième DBM de la même zone. On peut appliquer un algorithme de plus courts chemins, comme l'algorithme de Floyd-Warshall Cormen et al. (2001), pour fournir des DBM sous une forme normale. Ces algorithmes choisissent les plus fortes contraintes définissant les DBM qui caractérisent les zones. Si on met l'exemple précédent d'une DBM sous forme normale, on aura la DBM équivalente suivante :

$$\begin{pmatrix} 0 & -2 & 0 \\ 7 & 0 & 3 \\ 4 & 2 & 0 \end{pmatrix}$$

Les DBM fournissent la possibilité d'effectuer toutes les opérations requises à l'analyse en avant et l'analyse en arrière, voir Bouyer et al. (2004) pour plus de détails concernant l'utilisation de ces structures de données.
Donc les DBM représentent les structures de données de base qui servent à déterminer les zones pour l'implémentation et la manipulation des ensembles de configurations des automates temporisés. Cependant ces structures ont connu plusieurs améliorations, par exemple la minimisation des DBM Larsen et al. (1997a) et encore l'utilisation des CDD (Clock Difference Diagrams) pour compacter la représentation des unions de DBM Larsen et al. (1999).

3.8 Un exemple d'outils de vérification : Uppaal

Uppaal est un outil développé par Université d'Uppsala (Suède) et Université d'Aalborg (Danemark) Larsen et al. (1997a). Il sert à vérifier les systèmes temporisés. Cet outil est libre à l'adresse http ://www.uppaal.com. Il est conçu pour faire de la vérification sur une variante des automates temporisés. Cette variante est plus large syntaxiquement parce qu'elle autorise une représentation directe des

- contraintes d'urgence : dans le cas d'une transition qui doit être tirée immédiatement, sans attente,
- contraintes d'atomicité : quand on est obligé de tirer instantanément une séquence de transitions,
- etc.

Ces facilités de modélisation améliorent la concision et la lisibilité du modèle original mais n'ajoutent pas en effet de l'expressivité.
L'outil Uppaal offre la possibilité de vérifier un sous-ensemble de $TCTL_h$ qui regroupe surtout les propriétés d'accessibilité, de sûreté et aussi les propriétés de réponse. Pour plus de détails sur l'utilisation de cet outil, un tutoriel est disponible dans l'adresse web mentionnée plus-haut Behrmann et al. (2004).
Uppaal comprend principalement trois modules concernant la construction du modèle, la vérification et la simulation. Ce dernier module permet de voir l'évolution du modèle construit. Une capture d'écran de l'interface de modélisation est illustrée par Figure 3.14.

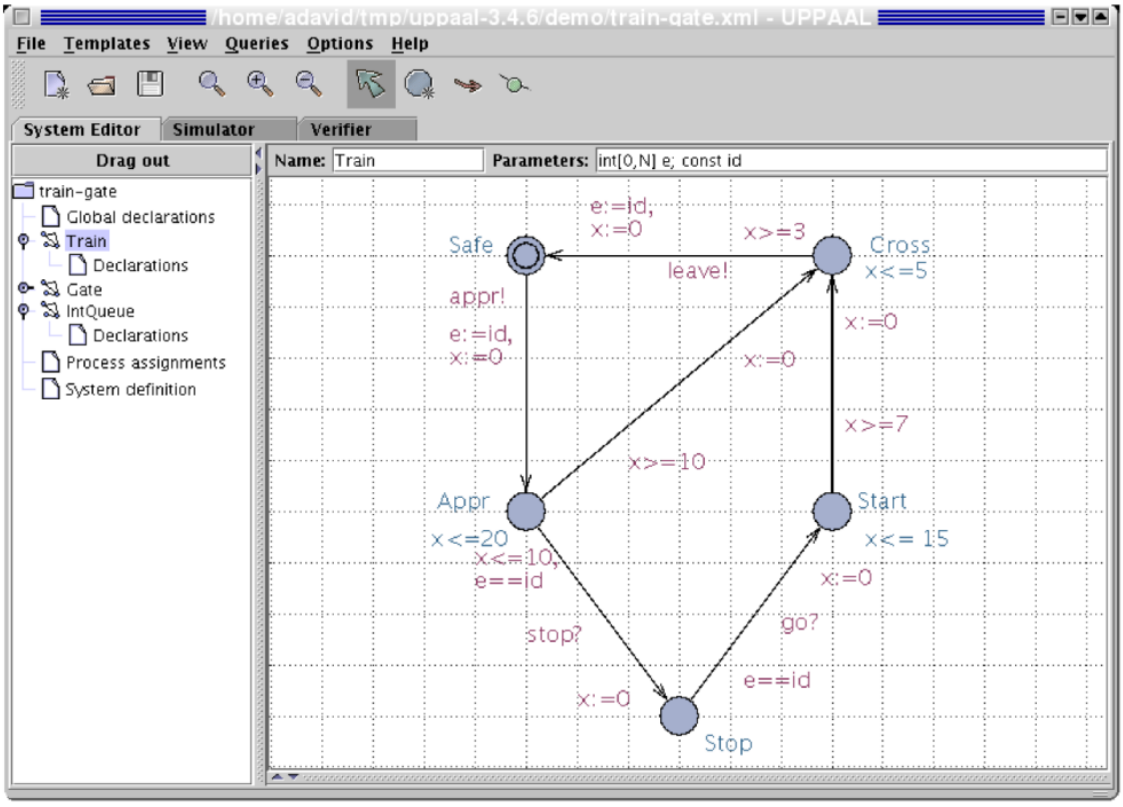


FIGURE 3.14 – Un exemple de modélisation par Uppaal.

Chapitre 4

les automates temporisés avec temps relatif

Sommaire

4.1 Introduction

La vérification des systèmes temps-réel dynamiques, tels que les réseaux ad-hoc et les systèmes ambiants, nécessite la proposition d'approches formelles qui permettent la construction des outils de vérification.
Dans ce chapitre qui est basé sur les résultats publiés dans (Layadi et al. 2016), la question de décidabilité est adressée sur le modèle des automates temporisés dynamiques avec vitesses relatives du temps (R-TA) qui est une variante du modèle classique des automates temporisés.
Nous introduisons un R-TA comme une extension d'un automate temporisé standard.
Dans ce modèle, les systèmes temporisés distribués sont formés par un ensemble d'automates temporisés. Chaque automate est caractérisé par un ensemble d'horloges locales qui évoluent selon une fréquence différente mais relative par rapport à celles des horloges des autres composants.

La contribution principale est la considération du paramètre *slope* (qui est le rapport entre les fréquences des horloges). Ce paramètre nous permet d'étudier une sémantique basée sur l'abstraction des régions pour évaluer et prouver la décidabilité. Notre vie quotidienne est dirigée par des systèmes plus complexes. Leurs caractéristiques les plus importantes sont la distribution, l'hétérogénéité et la dynamicité qui peut être exprimée en termes de mobilité. Les réseaux ad-hoc mobiles (MANET) sont un exemple des systèmes distribués complexes, constitués de nœuds mobiles sans-fil qui peuvent dynamiquement s'auto-organiser en topologies de réseaux arbitraires, afin de permettre aux gens et dispositifs de communiquer entre eux dans de bonnes conditions sur des espaces libres de toute infrastructure de communication préexistante Chlamtac et al. (2003)
La flexibilité et la convenance sont les raisons pour lesquelles, leurs applications ont été étendues du domaine traditionnel (militaire) vers une diversité de domaines commerciaux, par exemple, l'intelligence ambiante (Ahola 2002), les réseaux privés (Zimmerman 1996) et les services basés sur la localisation (Basagni et al.2000).
Une contrainte majeure dans la conception des réseaux ad-hoc scalables est la mobilité des nœuds. Dans ce sens, il est nécessaire de modéliser et vérifier les exigences de la dynamicité. De même pour la conception des solutions efficaces et adaptées aux différentes applications dynamiques qui ressemblent aux MANET.

4.1.1 Intérêt des approches formelles

Chaque exemple cité ci-dessus traite des problèmes très spécifiques, comme l'hétérogénéité et la dynamicité des composants (processus). Par exemple, avec l'apparition des drones de combat autonomes, la complexité des algorithmes de coordination des drones peut rendre difficile de fournir une assurance à un public concerné que ces unités autonomes coordonnent correctement **?**. Avec un modèle formel, il serait possible de fournir et prouver à la fois des algorithmes corrects pour effectuer des tâches complexes. Les méthodes formelles fournissent des techniques efficaces pour vérifier un système donné. Elles utilisent une rigueur mathématique qui aide à prouver la validité du système sous-jacent. Plusieurs modèles basés-automates ont été largement étudiés au cours des vingt dernières années, capturant une diversité d'aspects des comportements distribués. En outre, être capable de développer des techniques de vérification automatisées nécessite une bonne compréhension des modèles les plus simples, comme la plus part des modèles complexes sont construits comme une combinaison de ceux de base. Habituellement, le temps est mesuré par des dispositifs physiques, appelés horloges, qui présentent un comportement presque régulier au cours du temps. Les modèles des systèmes temps-réel doivent prendre en compte les propriétés temporelles. À cet effet, les horloges sont utilisées de manière explicite dans les contraintes des systèmes. Dans ce chapitre, nous adressons d'abord : le problème de la modélisation dans les systèmes temps-réel, en suite : l'hétérogénéité des composants des systèmes. L'hétérogénéité est vue comme une différence entre les fréquences d'horloges. Dans différentes applications, un système est constitué de plusieurs éléments qui sont caractérisés par leurs propres fréquences d'horloges. En général, il n'y a aucune raison pour supposer que les différents composants temporisés dans les systèmes ont une même référence du temps ou bien qu'ils évoluent selon un rythme similaire. Les réseaux d'automates temporisés sont bien connus par l'hypothèse de l'utilisation d'un temps global, comme dans (Larsen et al. 1995a ; Bouyer et al. 2006 ; Rodriguez-Navas Proenza 2013).

Cela ne reflète pas réellement les aspects des systèmes distribués. On propose dans ce chapitre une approche pour modéliser les systèmes distribués avec fréquences locales et relatives d'horloges. Chaque composant du système est décrit par un automate temporisé dans lequel toutes les horloges évoluent selon la même fréquence. Cependant, les horloges appartenant à différents processus (composants) sont autorisées à évoluer selon des fréquences différentes mais relatives. Il faut noter qu'une horloge peut être lue (utilisée) par tous les processus, alors que sa remise à zéro ne peut être effectuée que par le processus auquel elle appartient. Comme le nombre de configurations dans un système de transitions temporisé est infini, la construction de ce système ne sera alors jamais terminée. Les relations d'équivalence sont proposées pour agréger les configurations, cela veut dire qu'une classe d'équivalence (appelée région d'horloges) peut représenter un ensemble de configurations Alur and Dill (1990). À tout point dans le temps, le comportement futur du système modélisé est déterminé par sa localité actuelle et les valeurs des horloges de tous les processus, ce qui motive la redéfinition de nombreux concepts tels que les régions d'horloges et l'automate de régions. On va se concentrer sur l'effet de la relativité entre les fréquences d'horloges.

Hypothèses Les processus sont supposés avoir des fonctions de vitesse du temps, cette fonction caractérise le rythme de chaque processus pendant l'exécution de ses actions. Comme les processus peuvent avoir des rythmes différents, on suppose que ces vitesses sont relatives selon une échelle du temps globale (appelée temps absolu). Par conséquent, pour toute paire de processus p et q, il existe un entier c_{pq} qui est le rapport entre leurs fonctions de vitesse du gtemps à savoir $\tau_q = c_{pq}.\tau_p$.

4.1.2 Travaux connexes

Dans la littérature, plusieurs travaux portent sur la façon de considérer les horloges et le temps **?**. Dans le premier cas, les horloges sont synchronisées et utilisées par tous les processus (lecture et réinitialisation). Dans les autres propositions, les horloges peuvent se dériver d'une certaine quantité du temps δ, en particulier tant que les processus ne communiquent pas entre eux (via des actions de synchronisation). Ces travaux peuvent également être distingués par la prise en compte des langages temporisés ou bien nontemporisés. Dans (Akshay et al. 2008 ; Akshay et al. 2014), un système distribué est modélisé par un réseau d'automates temporisés qui évoluent selon des vitesses différentes. Pour vérifier les bonnes spécifications de chaque système, deux sémantiques sont proposées et bien étudiées. La sémantique universelle capture les comportements qui détiennent sous n'importe quel choix de fréquence d'horloges dans chaque composant du système. En revanche, la sémantique existentielle est proposée pour examiner l'ensemble des comportements que le système peut éventuellement exposer sous un choix de fréquences d'horloges. Les deux sémantiques universelle et existentielle sont considérées naturelles quand on veut vérifier qu'un système satisfait toujours une spécification positive ou bien évite une spécification négative respectivement. Le test du vide est indécidable dans le cas de la sémantique universelle. Une nouvelle sémantique décidable, appelée réactive, a été prouvée qu'elle est incluse dans la sémantique universelle. Elle est obtenue par la construction d'un graphe de régions pour des automates alternatifs.

En jouant sur les vitesses locales du temps (Akshay et al. 2014), on peut, soit borner la dérivation des horloges soit fixer le rapport entre toute paire d'horloges et dans les meilleurs cas, l'indécidabilité persiste. En particulier, la restriction sur les fonctions de vitesses lo-

cales du temps pour avoir des rapports fixes (rationnels) garde toujours l'indécidabilité de la sémantique universelle pour le test du vide. Les résultats présentés dans (Akshay et al. 2014) pour les automates temporisés avec évolution indépendante des horloges nous a donné l'idée d'aller chercher des résultats similaires lorsque l'on considère des automates temporisés avec vitesses relatives du temps. Plus précisément, lorsque le rapport entre toute paire de fonctions de vitesses locales du temps est une constante entière, est ce que le problème de l'accessibilité sera décidable pour les automates temporisés avec vitesses relatives du temps Le résultat principal de ce chapitre répond positivement à cette question, c'est-à-dire, le résultat est inversé et la décidabilité est assurée. Dans la littérature, des concepts similaires sont appliqués sur des technologies de réseaux différentes. Par exemple, dans (Kumar et al. 2014 ; Kumar et al. 2015 ; Misra et al. 2015), les réseaux véhiculaires sont modélisés par des réseaux (collaboratifs) d'automates d'apprentissage (learning automata). Cette approche permet :

— l'optimisation du schéma de routage pour envoyer une information à la destination finale avec un débit maximal et un délai minimal et
— la description d'une solution tolérante aux pannes par la conception d'un protocole de routage.

Dans (Misra, Krishna, et al. 2014 ; Krishna et al. 2013), le modèle des automates d'apprentissage est utilisé pour proposer une approche de qualité de service pour les applications cloud et de développer un système de recommandation basé sur l'analyse des sentiments pour aider les utilisateurs du cloud dans leurs besoins. Le même modèle est utilisé dans les réseaux de grilles intelligentes pour proposer un algorithme qui sélectionne un chemin pour la transmission des données afin d'optimiser la performance de la tolérance aux pannes et la gestion de l'énergie (Misra, Venkata Krishna, et al. 2014). Notre modèle temporisé, appelé automates temporisés avec vitesses relatives du temps r-TA, peut être utilisé pour modéliser et analyser le comportement temporisé des systèmes temps-réel et hétérogènes. Ce modèle permet de vérifier de nombreuses propriétés temporelles telles que la sureté, la vivacité et par dualité l'interblocage.

4.2 les automates temporisés avec temps relatif (R-TA)

Dans cette section, on présente le modèle de base appelé automates temporisés sur lequel on introduit la notion de la relativité entre les fréquences d'horloges. Ceci est réalisé par le produit asynchrone de ces automates.

4.2.1 Modélisation des systèmes temporisés

L'ensemble $\mathcal{C}_Z$ des formules d'horloges sur l'ensemble des horloges Z est donné par la grammaire $\varphi ::= true | x \bowtie c | \neg\varphi | \varphi \wedge \varphi$ où x est une horloge de Z
$\varphi \in \{<, \leq\}$ et c s'étend sur $\mathbb{Q}$. Dans cette grammaire, la formule d'orloges false est représenté par ¬ true et l'expression $\varphi \wedge \varphi$ sera utilisée pour agréger les formules d'horloges simples. A tout instant, la valuation d'une horloge est le temps cumulé depuis la dernière réinitialisation de cette horloge. Cette valuation d'horloges sur Z est un mappage υ :Z→$\mathrm{R}_{\geq 0}$.
Une valuation υ satisfait $\varphi \in \mathrm{C}_Z$ noté $\upsilon \models \varphi$, si φ est évaluée à true on respectant les valeurs données par υ. Pour $R \subseteq$Z,υ [R] dénote la valuation d'horloges définie par υ [R] (x)=0 si

$x \in R$ et υ[R] (x)= υ(x)
sinonυ_0 dénote la valuation d'horloges initiale qui mappe toute horloge de Z à 0.
Pour un tuple non-vide t = (t_1,t_2,...t_n), (t, e) dénote l'ajout d'un élément e au tuple t tel que (t,e) = (t_1,t_2,...,t_n, e) En plus, la fonction concat(s,t) retourne la concaténation de deux tuples non vides s (s_1,s_2,...,s_n)ett= (t_1,t_2,...,t_n)$commesuit$:
concat(s,t)= (...((s,t_1)t_2)...,t_n) = (s_1, ...,s_n,t_1, ...,t_n)
L'opération de substitution sur le tuple t est notée $t[t'_1/t_1]$. Elle consiste en la substitution de l'élément t_1 par t'_1.
Le reste de notations, utilisées dans ce chapitre et qui ne sont pas mentionnées dans cette partie, se trouvent dans la section 3.2.1.
Rappelons dans ce qui suit les notions classiques d'un automate temporisé (Alur Dill 1994) qui constitue la base du modèle R-TA.

Définition 4.1

Un automate temporisé (TA) est un tuple A= (S, ACT, Z, T, I, S_0,F) tel que S, Act et Z son respectivement des ensembles finis de localités, actions et horloges, $T \subseteq S \times$Act$\times$C$_Z$ $\times 2^Z \times$S l'ensemble fini des transitions, I :$S \rightarrow$C$_Z$ assigne un invariant à chaque localité, $s_0 \in$S est la localité initiale et $F \subseteq$S est l'ensemble des localités finales.

4.2.2 Modèle des automates temporisés à temps absolu

On définit le modèle des automates temporisés à temps absolu, qui utilise une fonction de fréquence des horloges, comme suit :

Définition 4.2

Un automate temporisé (TA) est un tuple A= (S, ACT, Z, T, I, S_0,F) tel que S, I et S_0 et F sont définis de la même manière que dans les automates temporisés et

- Z est un ensemble fini d'horloges qui évoluent selon la même fréquence. Cette fréquence dépend d'un certain temps absolu donné par la fonction $\tau : \mathbb{R}_{\geq 0} \rightarrow$R$_{\geq 0}$ avec τ (0)= 0 et τ (t) retourne le temps local dans le TA A à l'instant t du temps absolu
- $T \subseteq S \times Act \times C_Z \times 2^Z \times S$ est un ensemble fini de transitions.On écrit ($S \overset{a,\varphi,R}{\mapsto} S'$) pour représenter (S, a, φ, R, S').

4.2.3 Modèle des automates temporisés avec vitesses relatives du temps

Soit $proc$ un ensemble de processus tel que tout processus est modélisé par un automate temporisé A_p= $(S_p, Act_p, Z_p, T_p, I_p, s_{0p}, F_p)$ où les alphabets Zp sont deux à deux disjoints.
Le produit asynchrone B= (S, Act, Z, T, s_0, F) de ces automates sur $proc$ est défini comme suit :

- $S = \prod_{p \in proc} S_p$,
- $Act = \bigcup_{p \in proc} Act_p$,
- Z= $\bigcup_{p \in proc} Z_p$,
- s_0= $(s_{0p})_{p \in proc}$,

— $F = \prod_{p \in proc} F_p$ et
— $I(s) = \bigwedge_{p \in proc} I_p(s_p)$ pour tout $s \in S$.

En fin, pour $s, s' \in S, a \in Act, \varphi \in \mathcal{C}_Z$ et $R \subseteq Z$ on met $(S, a, \varphi, R, S') \in T$ s'il existe $p \in Proc$ tel que $(s_p, a, \varphi, R, S'_p) \in T_p$ et $s_p = s'_q$ pour tout $q \in proc \backslash \{p\}$

Les horloges de chaque processus évoluent selon une fréquence relative aux fréquences des horloges des autres processus. On définit un automate temporisé avec vitesses relatives du temps $(r - TA)$ comme suit :

définition 4.4

Un automate temporisé avec vitesses relatives du temps $(r-TA)$ sur un ensemble de processus $Proc$ est un tuple $B = (S, Act, Z, T, I, s_0, F)$ qui est un TA à l'exception de l'ensemble fini des horloges Z dans lequel toute horloge progresse selon l'évolution du temps dans le processus auquel elle appartient. Par conséquent, le résultat du produit asynchrone des TA sur un ensemble de processus $Proc$ est un $r - TA$.

4.2.4 sémantique

Soit $B = (S, Act, Z, T, I, s_0, F)$ un $r - ta$ sur un ensemble de processus $proc$ et un tuple de fonctions de vitesses locales du temp τ tel que $\tau = (\tau_p)_{p \in proc}$. Dans ce cas, τ_p est la vitesse locale du temps dans le processus p. Pour une valeur du temps absolu t, la fonction de mappage $\tau : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}^{proc}$ assigne le tuple $(\tau_p(t))_{p \in proc}$ à $\tau(t)$.

La sémantique de B en respectant τ est définie comme un système de transitions $(Q, q_0, \to)$ où $Q \subseteq S \times \mathbb{R}_{\geq 0}^Z$ est l'ensemble des états appelés configurations. Une configuration est une paire $< s \mid v >$ où s est un tuple de localités et v est une valuation d'horloges. À partir de la configuration initiale $q_0 =< s_0, v_0 >$, la relation de transition $\to$ entre les configurations est définie par les règles suivantes :

— $< s \mid v > \xrightarrow{d} < s \mid v \oplus \tau(d) >$
si $\forall d', 0 \leq d' \leq d \Rightarrow (v \oplus \tau(d')) \models I(S)$ pour tout $d \in \mathbb{R}_{\geq 0}$, avec $v \oplus \tau(t) = v(x) + \tau_p(t))_{x \in Zp, and p \in proc}$
— $< s \mid v > \xrightarrow{a} < s[s'_p/s_p] \mid v' >$
si, pour $a \in Act_p$, il existe la transition $s_p \xrightarrow{a, \varphi, R} s'_p$ dans T_p telle que :
— la valuation d'horloges v satisfait la garde φ.
— la valuation v' est obtenue par la réinitialisation de $R(v' = v[R])$ et elle satisfait l'invariant $I(s[s'_p/s_p])$.

4.3 problème à résoudre

Les modèles des systèmes temps-réel doivent prendre en considération les propriétés temporelles. À cet effet, dans les contraintes des systèmes, les horloges sont utilisées de manière explicite. Dans ce chapitre, on étudie l'impact de la relativité entre les fréquences d'horloges des systèmes et notamment sur le comportement temporel des applications distribuées, ainsi la difficulté de considérer cette relativité dans les spécifications des automates temporisés. Comme le nombre de configurations $< s \mid v >$ dans le système de transitions est

infini, la construction de ce dernier est impossible. Dans la littérature (Alur Dill 1994), des relations d'équivalence ont été proposées pour agréger les configurations des systèmes de transitions temporisés telles que chaque classe d'équivalence représente un ensemble de configurations $< s_i \mid v_i >$. Dans ce cas, les relations d'équivalence sont construites en respectant les exécutions. Les classes d'équivalence des valuations d'horloges sont appelées régions d'horloges.

4.3.1 Paramètre *slope* et les régions d'horloges

D'habitude, le temps est mesuré par des dispositifs physiques, appelés horloges, qui offrent un comportement presque régulier au cours du temps. La fonction $v(x)$ donne la valeur de l'horloge x. Pour obtenir les valeurs de la valuation d'horloges x à l'instant du temps absolu t, le tuple des fonctions continues de vitesses $\tau(t)$ est utilisé. Pour une paire d'horloges x et y (qui appartiennent respectivement aux processus p et q, leurs propres fréquences lui feront diverger de la référence du temps absolu avec un certain degré qui est égale au rapport entre leurs propres fréquences. Il représente la pente de la ligne droite sur Figure 4.1. Ce rapport de vitesses est appelé *slope*, tel que $slope_{xy} = \tau_q(t)/\tau_p(t)$. Il représente la pente de la droite de Figure 4.2.

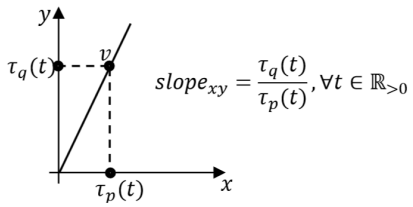


FIGURE 4.1 – Evolution de deux horloges avec différentes fréquences

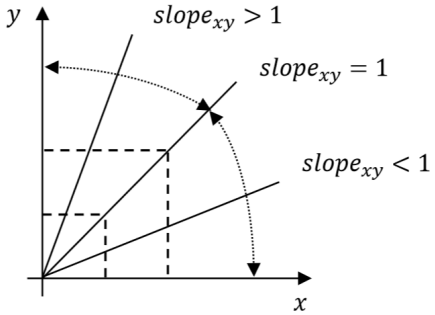


FIGURE 4.2 – Différentes possibilités de l'évolution de deux horloges

Lorsqu'une horloge est remise à zéro, la progression de sa valeur doit reprendre de zéro avec la même fonction de vitesse (voir Figure 4.3). La conception des systèmes devient cohérente si les composants partagent une perception conjointe du temps (Verssimo 1994 ; Lenzen et

al. 2010). Par conséquent, il est important que la perception générale soit consistante. Cette perception prend sa pleine dimension quand on construit le graphe de la sémantique du modèle. À tout moment, le comportement futur du système modélisé est déterminé par sa localité et les valeurs des horloges de tous les processus composant ce système, ceci motive la redéfinition de quelques concepts, comme les régions d'horloges et l'automate de régions. On se concentrera ci-après sur l'effet des fréquences relatives d'horloges.

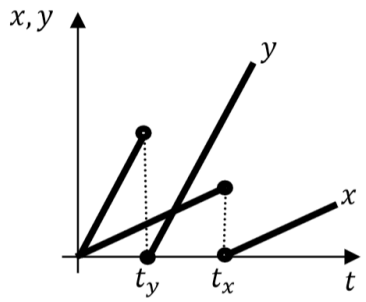


FIGURE 4.3 – Réinitialisation de deux horloges avec différentes fonctions linéaires de vitesses

4.3.2 Classes d'équivalence sur les valuations d'horloges

Soit $B = (S, Act, Z, T, I, s_0, F)$ un r-TA sur un ensemble de processus *proc* et un tuple de fonctions de vitesses τ.

Définition 4.5

Soit x (respectivement y) une horloge qui appartient au processus p (respectivement q)et qui évolue selon la fonction de fréquence τ_p (respectivement τ_q). On définit $slope_{xy} = \tau_q/\tau_p$. Pour les r-TA, nous nous limitons à l'utilisation des constantes entières dans les contraintes temporelles. Cela peut être fait en multipiliant toutes les constantes apparaissant dans les contraintes temporelles par leur plus petit commun multiple des dénominateurs, voir la preuve dans (Alur Dill 1994). Dans le reste de ce chapitre, on suppose que les contraintes temporelles ne comportent que des constantes entières. Comme il n'y a qu'un nombre fini de contraintes temporelles sur toute horloge x, on peut déterminer l'entier le plus grand $c_x \in \mathbb{N}$ avec lequel x a été comparée dans une contrainte temporelle (garde ou invariant) du rd-TA B. Dans ce qui suit, pour chaque paire d'horloges x et y, le paramètre $slope_{xy}$ est supposé être une constante entière quelle que soit la valeur du temps t.

Définition 4.6

Une relation d'équivalence, notée par $\sim$, sur l'ensemble de toutes les valuations d'horloges est définie comme suit :
Deux valuations d'horloges v et v' sont équivalentes, on écrit $v \sim v'$, si seulement si les conditions suivantes sont satisfaites :

— Pour toute $x \in \mathbb{Z}$, soit $\lfloor v(x) \rfloor$ et $\lfloor v'(x) \rfloor$ sont les mêmes ou bien les deux valuations $v(x)$ et $v'(x)$ sont plus grandes que c_x.

— Pour toute $x, y \in \mathbb{Z}$ avec $v(x) \leq c_x, v(y) \leq c_y$ et x (respectivement y) évolue selon la fréquence τ_p (respectivement τ_q) :
  — $c.\frac{1}{slope_{xy}} \leq v(x) \leq (c+1).\frac{1}{slope_{xy}} iff c.\frac{1}{slope_{xy}} \leq v'(x) \leq (c+1).\frac{1}{slope_{xy}} for c \in \mathbb{N}$.
  — $fract(slope_{xy} v(x)) \leq fract(v(y)) iff fract(slope_{xy} v'(x)) \leq fract(v'(y))$.
  Une classe d'équivalence sur les valuations d'horloges induit par $\sim$ est une région d'horloges de B.

Discussion

Dans cette section, nous suivons le même raisonnement que Baier and Katoen (2008) pour discuter comment la définition de la relation d'équivalence permet la répartition de l'espace des valuations d'horloges à des classes d'équivalence. Pour cela, nous considérons les quatre observations suivantes. Ils permettent la construction des conditions qui définissent les classes d'équivalence.

Première observation

Soit v et v' deux valuations d'horloges et soit φ une contrainte temporelle atomique (tous sur l'ensemble des horloges Z). Pour chaque horloge $x \in Z$, la forme de φ peut être soit $x < c$ bien $x \leq c$.

Nous avons $v \models x < c$ quand $\lfloor v(x) \rfloor < c$ Dans ce cas, la partie fractionnaire de $v(x)$ n'est pas importante. Dans l'autre cas, $v \models x \leq c$ quand $\lfloor v(x) \rfloor < c$ ou bien $\lfloor v(x) \rfloor = c$ et $fract(v(x) = 0$ Ainsi, $v \models \varphi$ dépend seulement de la partie entière $\lfloor v(x) \rfloor$ et le fait de savoir si $fract(v(x)) = 0$. Cela conduit à la première observation que deux valuations d'horloges v et v' sont équivalentes si

$\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor et fract(v(x)) = 0 ssi fract(v'(x)) = 0$.

Toutes les valuations d'horloges équivalentes, qui respectent cette observation, satisfont la contrainte temporelle φ à condition que φ soit une formule temporelle atomique de la forme $x < c$ ou bien $x \leq c$.

Exemple (une première répartition)

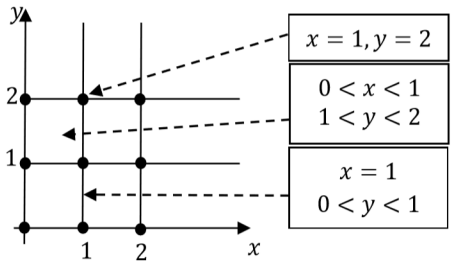


FIGURE 4.4 – Une première répartition

Nous considérons deux horloges v et y qui évoluent à des fréquences différentes, tel que $slope_{xy} = 2$. Les régions d'horloges obtenues par la première observation sont représentées par Figure 4.4.

Deuxième observation

Dans l'exemple ci-dessus et quand on fait passer le temps à partir d'une valuation $v'(v'(x) = 0.1$ et $v'(y) = 0.2)$, la valuation $v"(v"(x) = 0.5$ et $v"(y) = 1)$ est atteinte. Selon la première observation, l'horloge y est entrée dans une nouvelle région, ce qui n'est pas le même cas

pour x. Ceci est induit par le fait que y est deux fois plus rapide que x. Le changement de la région se produit chaque fois que l'horloge x évolue par 0,5, qui est $\frac{1}{slope_{xy}}(slope_{xy} = 2)$. Cela fait référence à une autre observation sur les équivalences de valuations formulée par : pour tout $x, y \in Z$ tel que x (respectivement y) évolue en fonction de τ_p (respectivement τ_q) : $c.\frac{1}{slope_{xy}} \leq v(x) \leq (c+1).\frac{1}{slope_{xy}} ssi c.\frac{1}{slope_{xy}} \leq v'(x) \leq (c+1).\frac{1}{slope_{xy}} pour c \in \mathbb{N}$.

Exemple 4.2(Deuxième répartition) On considère l'exemple ci-dessus. Les régions d'horloges obtenues par la première et la deuxième observation sont représentées par Figure 4.5.

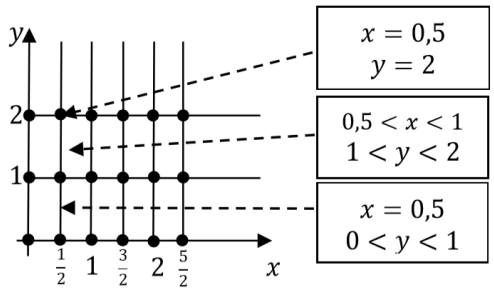


FIGURE 4.5 – Une deuxième répartition

Troisième observation On montre, via un exemple, que la première et la deuxième observation ne sont pas encore suffisantes. Considérant un r-TA B (voir Figure 4.6) avec un ensemble de localités $S = \{s_1, s_2, s_3\}$, un ensemble d'horloges $Z = \{x, y\}$ et un ensemble de transition $T = \{(s_1, \alpha, \varphi_2, \phi, s_2), (s_1, b, \varphi_3, \phi, s_3)\}$ où $\varphi_2 = x \geq 1$ et $\varphi_3 = y \geq 1$.

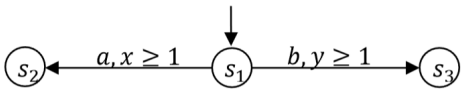


FIGURE 4.6 – r-TA B

Soit $q =< s_1, v >$ une configuration avec $0.5 < v(x) < 1$ et $0 < v(y) < 1$. On fait passer le temps pour savoir quelle transition peut être tirée en premier lieu. L'ordre des parties fractionnaires de $(slope_{xy} v(x))$ et $v(y)$ peut déterminer la transition qui peut être tirée ensuite. Si $fract(slope_{xy} v(x)) < fract(v(y))$, alors l'action b sera sensibilisée avant a. Si $fract(slope_{xy} v(x)) > fract(v(y))$, l'action sera sensibilisée en premier. Lorsque le temps s'écoule, plusieurs régions successeurs sont possibles en fonction de l'ordre entre les parties fractionnaires de la valuation d'horloges(selon le paramètre *slope*). Voir Figure 4.7.

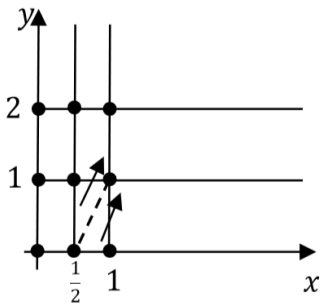


FIGURE 4.7 – passage du temps pour deux valuations d'horloges

Par conséquent, l'ordre de la partie fractionnaire de la valuation d'horloges est important. Alores, on doit ajouter une autre observation formulée par : pour toute $x, y \in Z$ tel que x (respectivement y) évolue en fonction de τ_p (respectivement τ_q) :

$$fract(slope_{xy}v(x)) \leq fract(v(y)) ssi fract(slope_{xy}v'(x)) \leq fract(v'(y))$$

Exemple 4.3 (troisième répartition illustrée par Figure 4.8)

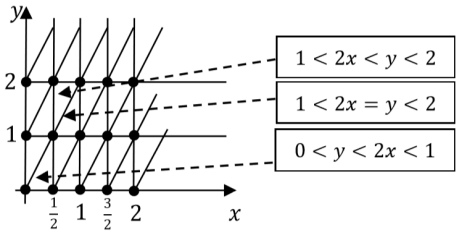


FIGURE 4.8 – Une troisième répartition

Les carrés $\left\{(xyy) \mid c < x < c + \frac{1}{slope_{xy}} \wedge c' < y < c' + 1\right\}$ seront décomposés par cette observation en trois classes d'équivalance : un segment de ligne et deux triangles.
$\left\{(x, y) \mid c < x < c + \frac{1}{slope_{xy}} \wedge c' < y < c' + 1 \wedge slope_{xy}(x - c) < y - c\right\}$,
$\left\{(x, y) \mid c < x < c + \frac{1}{slope_{xy}} \wedge c' < y < c' + 1 \wedge slope_{xy}(x - c) > y - c'\right\}$
et $\left\{(x, y) \mid c < x < c + \frac{1}{slope_{xy}} \wedge c' < y < c' + 1 \wedge slope_{xy}(x - c) = y - c'\right\}$.

Dernière observation

Les observations citées ci-dessus donnent un ensemble dénombrable (mais infini) de régions d'horloges. Pour éviter cet obstacle, nous exploitons la plus grande valeur C_x de chaque horloge x. Donc, on n'applique les observations ci-dessus que pour les valuations d'horloges $v(x) \leq C_x$, pour chaque horloge x, afin de limiter la répartition de l'espace de valuations. Par conséquent, on obtient un nombre fini de classes d'équivalence des valuations d'horloges.

Exemple 4.4 (répartition Finale)

Considérant l'exemple précédant tel que $C_x = 2$ et $C_y = 2$. Les régions d'horloges obtenues par toutes les observations sont représentées par Figure 4.9. Ainsi, on a 15 points relatifs aux

coins (par exemple ,$x = 0.5 \wedge y = 2$]), 38 segments de lignes (par exemple $[0 < 2x = y < 1]$) et 23 régions ouvertes (par exemple $[0 < 2x < y < 1]$).

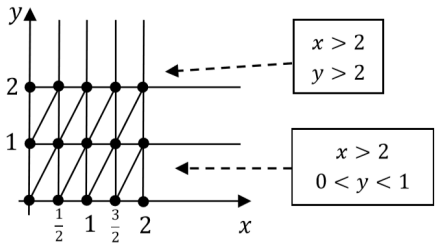


FIGURE 4.9 – Répartition finale

Notez que si deux valuations d'horloges sont équivalentes ($v \sim v'$) alors $v \models \varphi$ si et seulement si $v' \models \varphi$ pour chaque contrainte temporelle φ. Par conséquent, si chaque valuation d'horloge v de la région α satisfait φ, alors on dit que α satisfait φ et on écrit $\alpha \models \varphi$.

4.3.3 Représentation des régions d'horloges

Chaque classe d'équivalence sur les valuations d'horloges peut être spécifiée par un ensemble fini de contraintes temporelles qui la satisfait. La notation $[v]$ représente la région d'horloges à laquelle v appartient. Dans Figure 5.9, on représente par $[0 < y < 2x < 1]$ la région qui contient toutes les valuations d'horloges qui satisfont la contrainte $(0 < y < 2x < 1)$ par exemple la valuation $v(x) = 0.3$ et $v(y) = 0.5$.

Définition 4.7 Pour toute horloge $x \in Z$, on définit $slope_{max(x)}$ comme étant la plus grande valeur de $slope_{xy}$ pour toute $y \in Z$. On utilise Exemple 4.1 pour expliquer cette définition en calculant $slope_{max}$:

- $slope_{max(x)} = max(slope_{xy}, slope_{xx}) = max(\frac{2}{1}, 1) = 2$.
- $slope_{max(y)} = max(slope_{yx}, slope_{yy}) = max(\frac{1}{2}, 1) = 1$. Noter que si x est l'horloge la plusrapide dans H alors $slope_{max(x)} = slope_{xx} = 1$, en outre $\frac{1}{slope_{max(x)}}$ représente la plus petite quantité du temps pendant laquelle x ne peut pas rester dans la même région. Dans Exemple 4.1, l'horloge x change sa région chaque demi unité du temps qui correspond à $\frac{1}{slope_{max(x)}} = \frac{1}{2}$ par contre y fait ce cha,gement chaque unité de temps (sauf pour les régions représentées par des points). La représentation d'une région d'horloges s'accorde avec deux points suivantes :
  - Pour toute horloge x évoluant selon la fréquence τ_p, il y a une contrainte temporelle prise de l'ensemble :
    $\Big\{x = c \mid c = 0, \frac{1}{slope_{xy}}, 2\frac{1}{slope_{xy}}, ..., 1, 1 + \frac{1}{slope_{xy}}, 1 + 2\frac{1}{slope_{xy}}, ..., C_x forally \in H\Big\}$
    $\bigcup\{ \bigwedge_{y \in H}(c - \frac{1}{slope_{xy}} < x < c) \mid c = \frac{1}{slope_{xy}}, 2\frac{1}{slope_{xy}}, .., 1, 1 + \frac{1}{slope_{xy}}, 1 + 2\frac{1}{slope_{xy}}, ..,$
    C_x for each $y \quad \bigcup\{x > C_x\}$. (1)
  - pour toute paire d'horloges x et y qui évoluent respectivement selon τ_p et τ_q tel que $c < x < c + \frac{1}{slope_{max(x)}}$ et $d < y < d + \frac{1}{slope_{max(y)}}$ apparaissent dans (1) pour certaines valeurs de c et d, que la valeur de $slope_{xy}(x - c)$ soit plus petite que, égale à ou plus grande que $y - d$.

4.3.4 Successeurs des régions d'horloges

Dans ce qui suit, nous introduisons la relation du successeur sur l'ensemble des régions d'horloges. Lorsque le temps avance depuis n'importe quelle valuation d'horloges v dans une région α, on atteindra tous ses successeurs α'. Formellement, on dit que α' est un successeur de la région α s'il y a v dans α, v' dans $\alpha', t \in \mathbb{R}_{>\not\vdash}$ de telle sorte que $v' = v \oplus \tau(t)$ avec $v \oplus \tau(t) = (v(x) + \tau_p(t))_{\pi^{-1}(x)=p}$. Par exemple dans Figure 4.9, les cinq successeurs de la région $\alpha = [(1.5 < x < 2), (1 < y < 2x - 2)]$ sont : elle même, $[(x = 2), (1 < y < 2)], [(x > 2), (1 < y < 2), (1 < y < 2)], [(x > 2), (y = 2)]$ et $[(x > 2), (y > 2)]$. Ces régions sont celles couvertes par une ligne tracée, d'un point quelconque en α, en parallèle à la ligne $y = slope_{xy}x = 2x$ (avec une direction vers le haut).

Construction des successeurs des régions d'horloges Pour calculer un successeur d'une région α, on doit donner :

- **étape 1.** Pour chaque horloge x, une contrainte de la forme $(x = c)$, $c < x < c + \frac{1}{slope_{max(x)}})$ ou bien $(x > c_x)$ et
- **étape 2.** Pour toute paire x et y de telle sorte que $(c < x < c + \frac{1}{slope_{max(x)}}$ et $(d < y < d + \frac{1}{slope_{max(y)}})$ apparaissent dans l'étape 1, une relation d'ordre entre $slope_{xy}(x - c)$ et $y - d$.

Pour calculer les successeurs possibles, trois cas sont distingués :

Premier Cas

Chaque horloge x dans la région α satisfait la contrainte $(x > c_x)$ alors α n'a qu'un successeur, qui est elle-même. C'est le cas de la région $[(x > 2), (y > 2)]$ dans Figure 5.9

Deuxième Cas

Ce cas considéré quand il y a au moins, dans la région α, une horloge x qui satisfait la contrainte $x = c$ pour certain $c \leq c_x$. L'ensemble Z_0 contient toutes les horloges apparaissant dans une forme de contrainte similaire à celle de x. La région d'horloges αsera immédiatement changée lorsque le temps avance, parce que la partie fractionnaire de chaque horloge dans Z_0 devient différente de 0. Les régions d'horloges α et β ont les mêmes successeurs où β est spécifiée par :

1. Un ensemble de contraintes temporelles qui peut être donné comme suit :
   (a) Pour toute horloge $x \in Z_0$:
      i. Siα satisfait $(x = c_x)$ alors β satisfait $(x > c_x)$;
      ii. Si α satisfait $(x = c)$ alors β satisfait $(c < x < c + \frac{1}{slope_{max(x)}})$.
   (b) Pour toute horloge $x \notin Z_0$, la contrainte temporelle dans α reste la même que celle de β.
2. La relation d'ordre entre $slope_{xy}(x - c)$ et $y - d$ de toute paire d'horloges x et y dans α est la même que celle dans β, tel que les deux conditions $x < c_x$ et $y < c_y$ sont vérifiées dans la région α

Par exemple dans Figure 5.9, les successeurs de la région $[(x = 0), (0 < y < 1)]$ sont les mêmes successeurs de la région $[0 < 2x < y < 1]$.

Troisième Cas

Si le premier et le deuxième cas ne s'appliquent pas, alors soit Z_0 l'ensemble des horloges x pour lesquelles la région α satisfait les deux contraintes $c < x < c + \frac{1}{slope_{max(x)}}$ et $slope_{xy}(x - c) \geq y - d$ pour toutes les horloges y pour lesquelles la région α satisfait $d < y < d + \frac{1}{slope_{max(y)}}$. Ainsi, quand le temps avance, les horloges de Z_0 prennent les valeurs $c + \frac{1}{slope_{max(x)}}$. Par conséquent, les successeurs de la région α sont α, β et tous les successeurs de β qui est spécifiée par :

1. Un ensemble de contraintes temporelles qui peut être donnée comme suit :
   (a) Pour toute horloge $x \in Z_0$, si α satisfait $(c < x < c + \frac{1}{slope_{max(x)}})$ alors β satisfait $(x = c + \frac{1}{slope_{max(x)}})$;
   (b) Pour toute horloge $x \notin Z_0$, la contrainte temporelle de α reste la même que celle dans β
2. Pour toute paire f'horloges x et y tel que $(c < x < c + \frac{1}{slope_{max(x)}}$ et $(d < y < d + \frac{1}{slope_{max(x)}}$ apparaissent dans (1.b), la relation d'ordre entre $slope_{xy}(x - c)$ et $y - d$ dans α reste la même que celle dans β Par exemple dans Figure 5.9, les successeurs de la région $[0 < 2x < y < 1]$ incluent elle-même, $[(0 < x < 0.5), (y = 1)]$ et tous les successeurs de $[(0 < x < 0.5), (y = 1)]$.

4.4 Automate de régions

Les définitions précédentes concernant la relation d'équivalence sur l'ensemble de toutes les valuations d'horloges et la relation du successeur sur ces classes d'équivalence permettent la définition de l'automate de régions.
L'automate de régions $R(B)$ du R-TA est formé par un ensemble de configurations et une relation de transition sur cet ensemble. Toute configuration $< s, \alpha >$ enregistre une localité de l'automate Bt une région d'horloges α des valeurs actuelles des horloges. La configuration initiale de $R(B)$ est $< s_0, [v_0] >$ telle que s_0 et la localoté initiale de B et $[V_0]$ est la région des valuations initiales des horloges qui mappe chacune des horloges de Z à 0. La relation de transition de $R(B)$ relie $< s_0, [v_0] >$ et $< s'_0, [v_{0]>}$ec l'étiquette et seulement s'il existe une transition étiquetée par α partir d'une localité s avec une valuation d'horloges $\upsilon \in \alpha$ vers une localité s' avec une valuation d'horloges $\upsilon' \in \alpha'$ ans le R-TA (B) Pour une région d'horloges α et un ensemble d'horloges $R \subseteq Z$ la région $\alpha[R]$ dénote la réinitialisation de toutes les horloges de R dans toute valuation υ de α Formellement $\forall \upsilon \in \alpha, \forall x \in Z$:
— si $x \in R$ alors $\upsilon(s) = 0$ dans $\alpha[R]$.
— soit $\upsilon(x)$ dans $\alpha[R]$ est le même que dans α.

Définition 4.8

Soit B= $(S, Act, Z, T, I, S_0, F)$ R-TA sur un ensemble de processus *proc* un tuple de fonctions de vitesses locales du temps τ L'automate de régions$R(B)$ est un automate sur l'alphabet Act tel que :

— Les configurations de $R(B)$ sont de la forme $< s|\alpha >$où s est une localité de B et α est une région d'horloges.
— La configuration initiale est de la forme $< s_o|[\upsilon_0] >$ où $\upsilon_0(x) = 0$ pour tout $x \in Z$.
— Une transition de $R(B)$, de la configuration $< s|\alpha >$ vers $< s'|\alpha' >$, est étiqueti par $\alpha \in Act$ si et seulement s'il existe une transition $(s, \alpha, \varphi, R, s')$ dans T et une région d'horloges a'' qui satisfait
  — a'' est un successeur de la région α
  — $a'' \models \varphi$
  — $a' = a''[R]$

Théorème 4.1

Soit B un R-TA et soit $C_x \in \mathbb{R}$ plus grande constante qui apparait dans les contraintes temporelles de B définies sur l'horloge x Alors, le nombre de régions d'horloges **nbr** de l'automate de régions $R(B)$ une borne inférieure et une autre borne supérieure comme suit :

$$\mathbf{nbr} \geq |Z|! * \prod_{x \in Z}(c_x * slop_{max(x)})$$

$$\mathbf{nbr} \geq |Z|! * 2^{|Z|-1} * \prod_{x \in Z}(2(c_x * slop_{max(x)} + 2)$$

Preuve 4.1

Considérant une autre représentation des régions d'horloges pour déterminer les bornes inférieures et supérieures. Notez qu'il existe une relation un-à-un entre cette nouvelle représentation de la région et la région elle-même. Les bornes sont dérivées en utilisant cette représentation. Soit Z un ensemble d'horloges et valuation d'horloges sur Z On représente chaque région d'horloges α un tuple $< interv, permut, pred >$ $oùinterv$ une famille d'intervalles permutation d'un sous-ensemble d'horloges de Z, et $pred \subseteq z$ est un ensemble d'horloges tel que :
— $interv = (interv_x)_{x \in Z}$ est une famille d'intervalles où

$$interv_x \in \left\{ \begin{array}{l} [0,0],]0, \frac{1}{slope_{max(x)}}] \\ [\frac{1}{slope_{max(x)}}, \frac{1}{slope_{max(x)}}] \\ ]\frac{1}{slope_{max(x)}}, \frac{2}{slope_{max(x)}}[\\ ...,]c_x - \frac{1}{slope_{max(x)}}, c_x[\end{array} \right\}$$

tel que $(x) \in interv_x$ horloge $x \in Z$ et les valuations d'horloges υ à la région α.
— Soit Z_{open}emble des horloges $x \in Z$ pour lesquelles $interv$ est un isntervalle ouvert, Salors :

$$Z_{open} = \left\{ \begin{array}{l} x \in Z|interv_x \in \\ ]0, \frac{1}{slope_{max(x)}}], \\ ]\frac{1}{slope_{max(x)}}, \frac{2}{slope_{max(x)}}[, \\ ...,]c_x - \frac{1}{slope_{max(x)}}, c_x[, \\ ]c_x, \infty] \end{array} \right\}$$

$permul = \{x_{i1}, ..., x_{ik}\}$ est une permutation de $Z_open = \{x_1, ..., x_k\}$ tel que pour toute valuation d'horloges la région α les horloges de Z_open sont arrangées selon : $i_h < i_j$

implique $slop_{x_{i_h}x_{i_j}(x-x_h-c)} \leq (x_{i_j} - d)$ *où* c (resp.c) est la bonde gauche du $interv_{x_{ij}}$ (resp.$interv_{x_{ij}}$).

— $pred \subseteq Z_open$ contient tout les horloges rtutes les horloges x_{i_j} de Z_open tel que pour tout valuation d'horloge de la région α : $slop_{x_{i_h}x_{i_{j-1}}}(x_{ij} - c) = x_{i_{j-1}} - d$ Ainsi, on a assuré une relation un-à-un entre les triplets $< interv, permut, pred >$ et les régions d'horloges. La borne supérieure pour le nombre de régions d'horloges est atteinte par la combinaison des affirmations suivantes qu'il y a :

— Exactement $\prod_{x \in Z}(2(c_x * slop_{max(x)} + 2)$ familles d'intervalles $interv$ différentes,

— Au maximum $|Z_{open}|! \leq |Z|!$ permutations différentes sur et Z_open et

— Au maximum choix différents pour $2^{|Z_open|-1} \leq 2^{|Z|-1}$ choix différent pour $pred \subseteq Z \backslash \{x_1\}$

La borne inférieure est atteinte lorsque chaque valuation d'horloges (x) partient à un intervalle ouvert $interv_x$ (mais le dernier intervalle $]c_x, \infty[$) et a une partie fractionnaire différente par rapport aux autres horloges. Dans ce cas $pred = \emptyset$ et

$$Z_{open} = \left\{ \begin{array}{l}]0, \frac{1}{slope_{max(x)}}],]\frac{1}{slope_{max(x)}}, \frac{2}{slope_{max(x)}}[, \\ ...,]c_x - \frac{1}{slope_{max(x)}}, c_x[, \end{array} \right\}$$

Par conséquent, il y a exactement $\prod_{x \in Z}(c_x * slop_{max(x)})$ possibilités pour maximum | Z |! permutations différentes, la borne inférieure est trouvée.

Corollaire 4.1

[terminaison] Pour un ensemble de processus *proc* et un tuple de fonctions de vitesses locales du temps τ la construction d'un automate de régions $R(B)$ Pour tout r-TA se termine et a un espace fini de régions.

Preuve 4.2

Le nombre de régions d'horloges du R-TA (B)est borné, donc prouver la finitude de la construction de l'automate de régions dérive immédiatement du Théorème 5.1.

Théorème 4.2 (décidabilité)

Etant donné un R-TA (B)sur un ensemble *proc*est un tuple τ le test du vide et model-checking sont décidables.

Preuve 4.3

Comme la construction de l'automate de régions du R-TA (B) est possible, le test du vide et model checking sont prouvés décidable.

4.5 Algorithme de transformation d'un R-TA vers un automate de régions

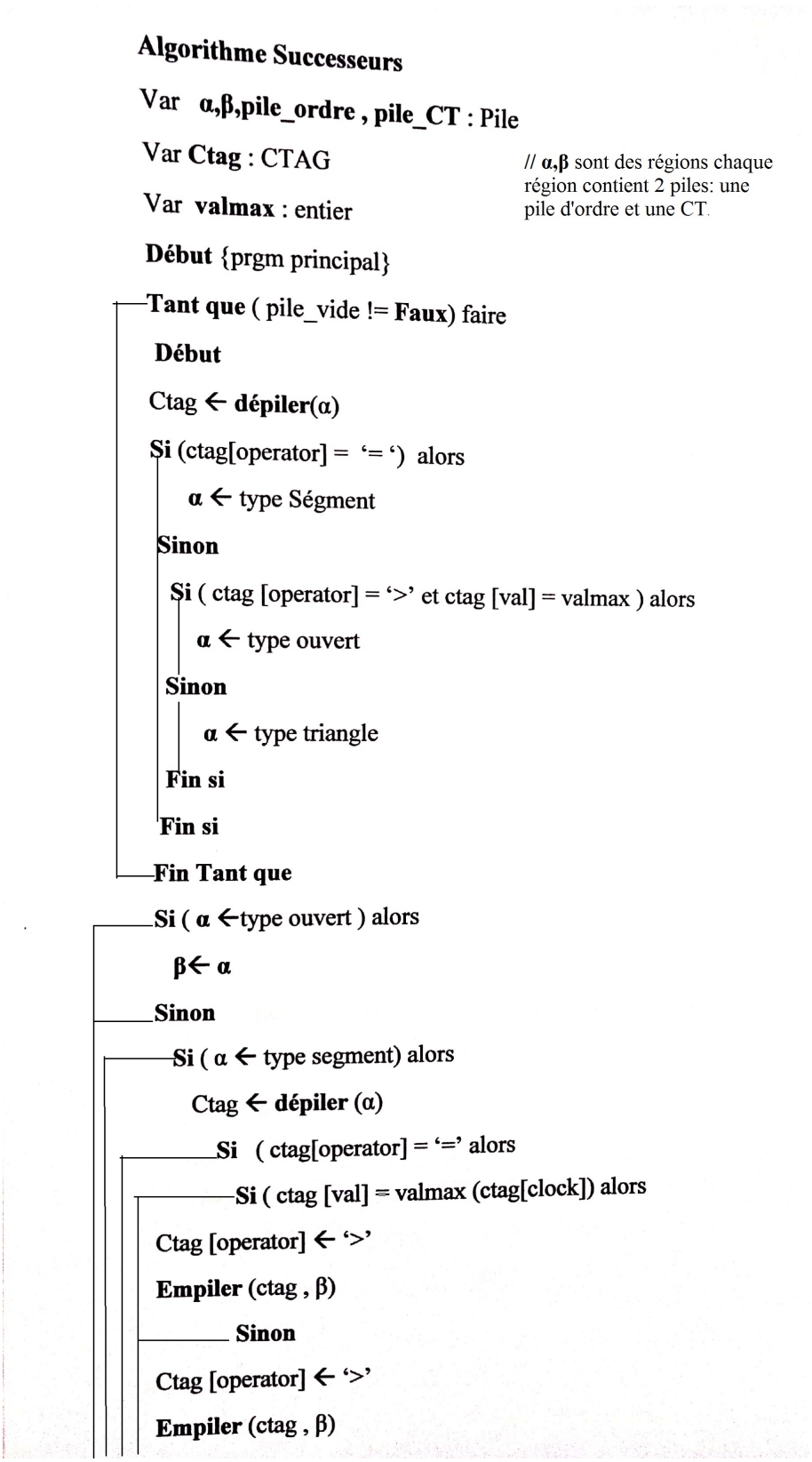


FIGURE 4.10 – Partrie 01 de l'alghorithme de transformation

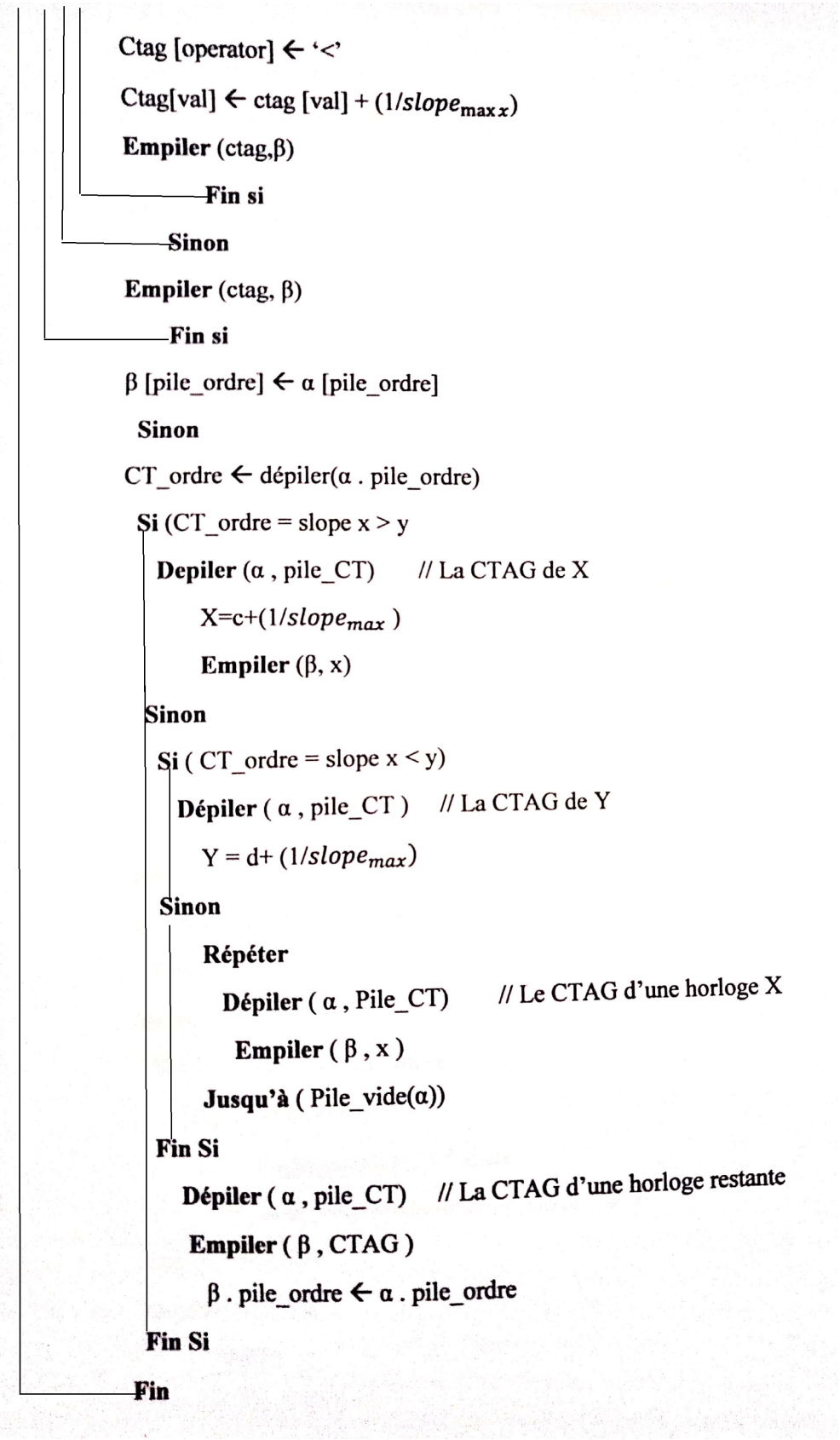


FIGURE 4.11 – Partie 02 de l'alghorithme de transformation

4.6 Conclusion

Dans ce chapitre, une approche pour l'analyse des systèmes temporisés a été proposée où les horloges évoluent avec des fréquences relatives. La preuve de la décidabilité a été basée sur une abstraction, appelée région, du comportement du système, pour cela, les concepts nécessaires sont redéfinis. Comme suite de cette contribution, il reste à étudier la puissance d'expressivité des automates temporisés avec des vitesses relatives du temps et d'étudier encore une fois l'effet des autres relations entre ces fréquences d'horloges. Cela peut être réalisé en explorant les différentes formes et valeurs du paramètre *slope* autre direction consiste à explorer les capacités de ce modèle pour décrire la sémantique des spécifications des systèmes ambiants. Enfin, l'application de l'approche model-checking peut demander une autre extension de ces résultats.

Chapitre 5

Conception et implémentation

Sommaire

5.1 Introduction

Dans ce chapitre, nous allons présenter les diagrammes de classes pour la modélisation du R-TA et l'automate de régions ainsi que le langage de développement java avec les environnements de développement nécessaires pour le développement et la création de notre application.

5.2 Diagrammes de classes

Le diagramme de classes est un schéma utilisé en génie logiciel pour présenter les classes et les interfaces des systèmes ainsi que leurs relations. Ce diagramme fait partie de la partie statique d'UML, ne s'intéressant pas aux aspects temporels et dynamiques.
Une classe décrit les responsabilités, le comportement et le type d'un ensemble d'objets. Les éléments de cet ensemble sont les instances de la classe.
Une classe est un ensemble de fonctions et de données (attributs) qui sont liées ensemble par un champ sémantique. Les classes sont utilisées dans la programmation orientée objet. Elles permettent de modéliser un programme et ainsi de découper une tâche complexe en plusieurs petits travaux simples.

Les classes peuvent être reliées grâce au mécanisme d'héritage qui permet de mettre en évidence des relations de parenté. D'autres relations sont possibles entre des classes, représentées par un arc spécifique dans le diagramme de classes.
Elles sont finalement instanciées pour créer des objets (une classe est un moule à objet : elle décrit les caractéristiques des objets, les objets contiennent leurs valeurs propres pour chacune de ces caractéristiques lorsqu'ils sont instanciés).

5.2.1 Diagramme de classes de l'automate temporisé avec temps relatif

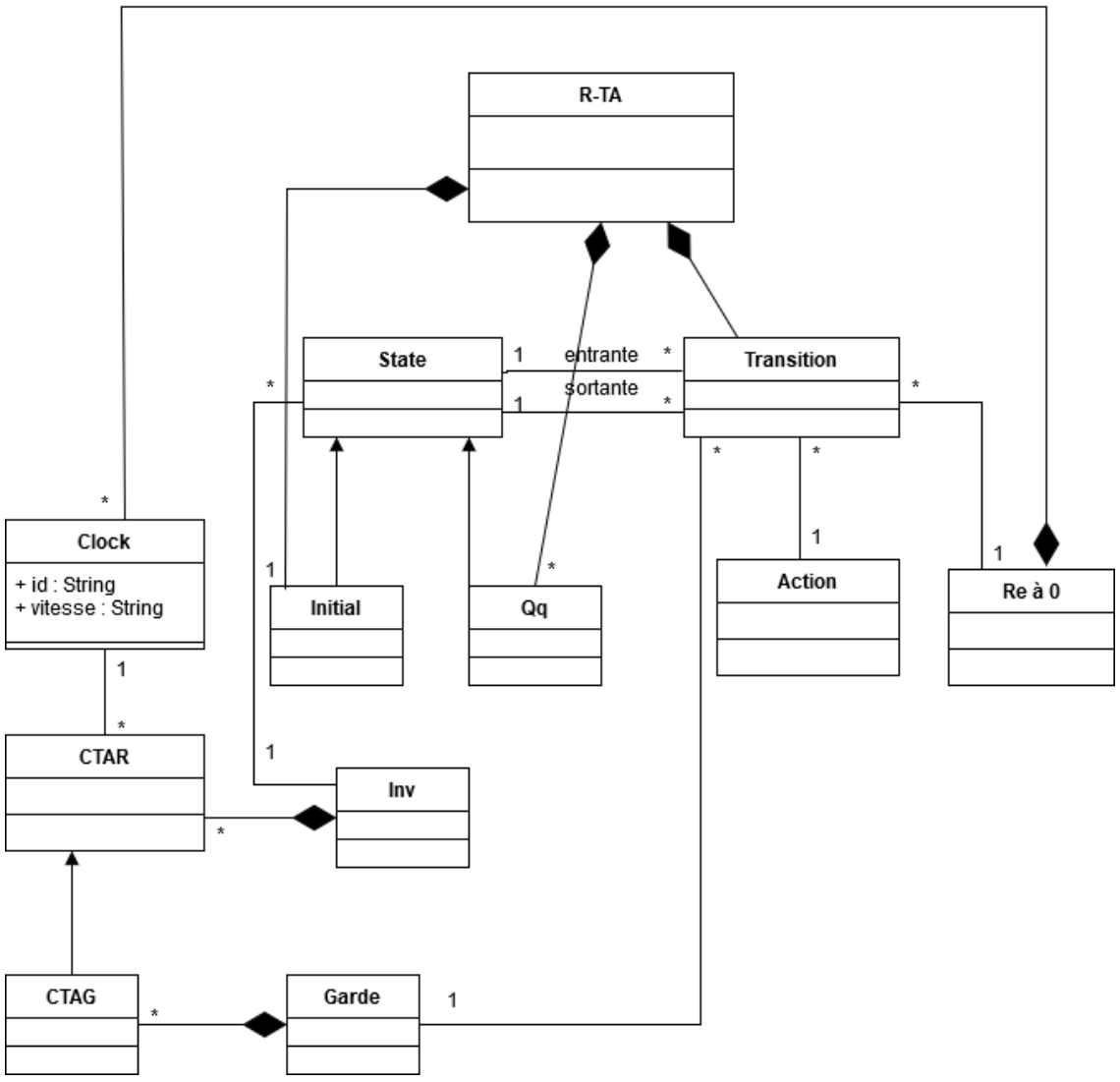


FIGURE 5.1 – Diagramme de classes du R-TA.

5.2.2 Diagramme de classes de l'automate de régions

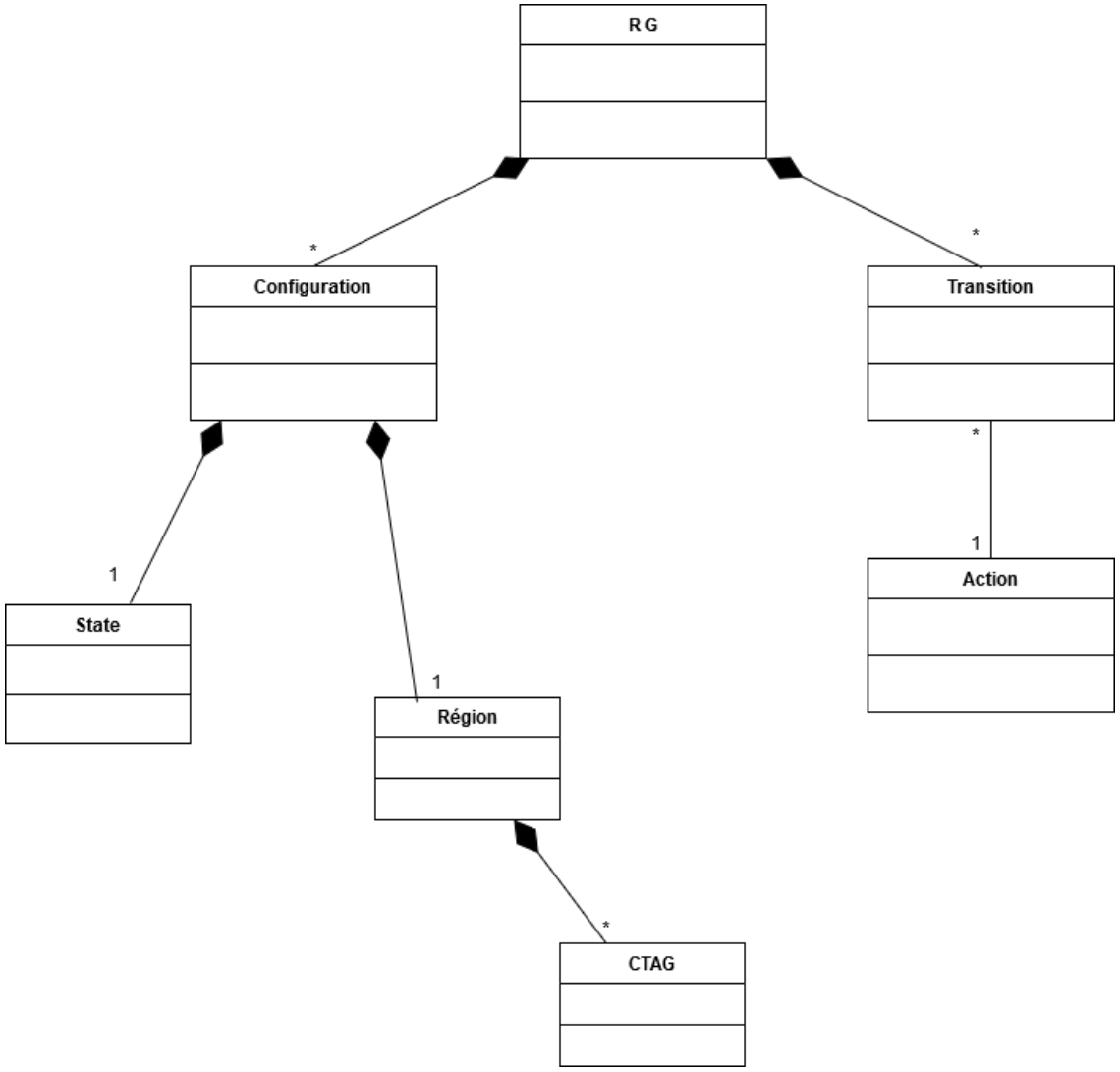


FIGURE 5.2 – Diagramme de classes de l'automate de régions.

5.3 Les outils Hardware et Software utilisés dans notre mémoire

5.3.1 Les outils Software

Langage JAVA

Java est un langage orienté objet, utilisé pour le développement de divers types d'application. Il est caractérisé par sa probabilité due à l'utilisation d'une machine virtuelle la JVM(Java Virtual Machine) qui fait interface entre le programme et le système d'exploitation.

FIGURE 5.3 – Logo JAVA

IntelliJ IDEA

IntelliJ IDEA est un Environnement de développement intégré conçu spécialement pour le langage Java par l'entreprise Jetbrains.

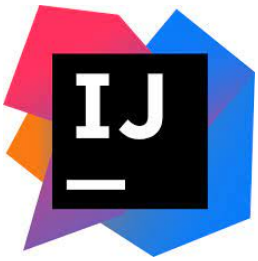


FIGURE 5.4 – Logo IntelliJ IDEA

Overleaf

Dans notre mémoire, on a utilisé l'outil Overleaf pour la rédaction du manuscrit en format **.tex**. Overleaf est un éditeur LaTeX en ligne, collaboratif en temps réel.

FIGURE 5.5 – Logo Overleaf

Diagrams.net

Diagrams.net (anciennement draw.io) est un logiciel de dessin graphique multiplate-forme gratuit et open source développé en HTML5 et JavaScript. Son interface peut être utilisée pour créer des diagrammes tels que des organigrammes, des structures filaires, des diagrammes UML, des organigrammes et des diagrammes de réseau.
Diagrams.net est disponible en ligne en tant qu'application Web multi-navigateur et en tant qu'application de bureau hors ligne pour Linux, macOS et Windows

FIGURE 5.6 – Logo Diagrams.net

GraphStream

GraphStream est une bibliothèque Java de gestion de graphes. Son objectif principal est la modélisation de réseaux d'interaction dynamiques de différentes tailles. Le but de la Bibliothèque est de fournir un moyen de représenter des graphiques et de travailler dessus.

FIGURE 5.7 – Logo GraphStream

5.3.2 Les outils Hardware

Nous avons uilisé dans notre mémoire deux micro-portables HP ELITEBOOK et DELL :

HP ELITEBOOK

— Processeur Intel(R) Core(TM) i7-8565U CPU @ 1. 80GHz 1. 99 GHz
— Mémoire installées(RAM) : 8 Go
— Système d'exploitation 64 bits

DELL

— Processeur : Intel(R) Core(TM) i3-7020U CPU @ 2. 30GHz 2. 30GHz
— Mémoire installées(RAM) : 4 Go
— Système d'exploitation 64 bits

5.4 Code source

5.4.1 Structure du code source

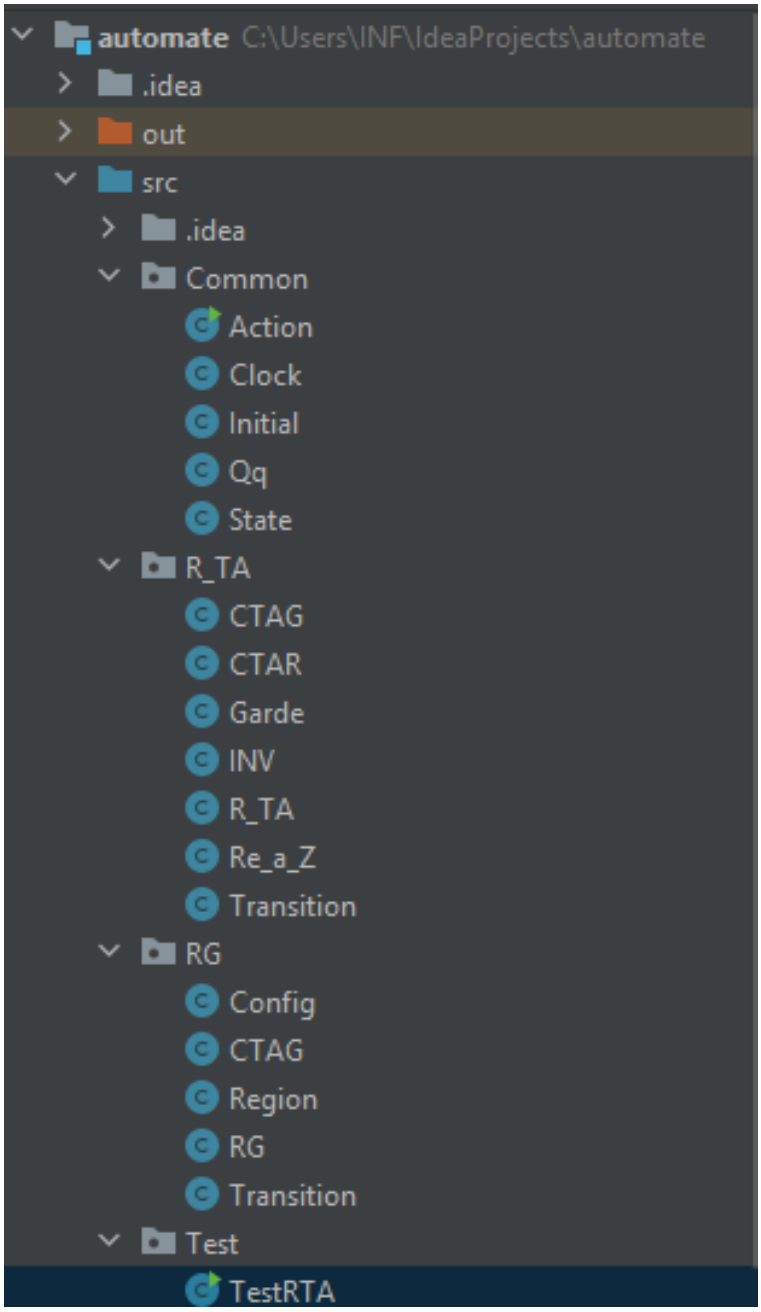


FIGURE 5.8 – Structure du code source

5.4.2 Interface principale

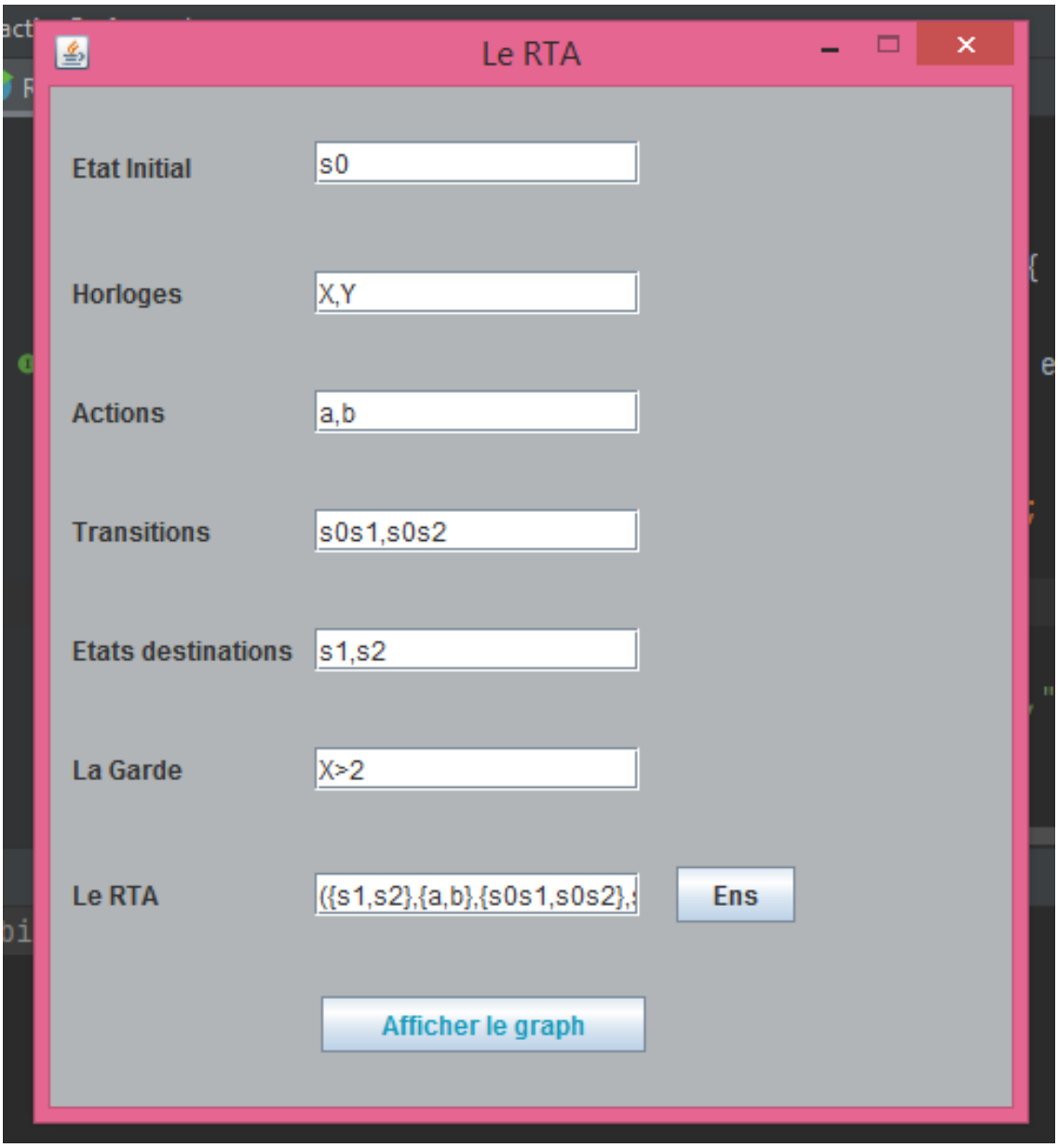


FIGURE 5.9 – Interface principale

5.4.3 Code source pour schématiser la structure R-TA

```
public void drawRTA() {
    int id = 0;
    for (Qq e : qq) {

        rta.getNode(e.getIdqq());

    }
    for (Transition t : transitions) {

        rta.addEdge(Integer.toString(id), t.getInitial().getIdInitial(),t.getqq().getIdqq(), directed: true);

        id++;
    }
    rta.display();
}
```

FIGURE 5.10 – Méthode pour schématiser la structure R-TA

5.4.4 Conclusion

Dans ce chapitre nous avons présenté le langage de programmation java avec l'outil de développement Intellij IDEA. Nous avons aussi présenté les deux diagrammes de classes modélisés par un logiciel de dessin graphique qui est diagram.net

Chapitre 6

Conclusion générale et perspectives

Sommaire

6.1 Bilan

L'objectif principal de ce travail était de participer au développement de modèles et d'algorithmes pour la modélisation et la vérification des systèmes temps-réel où les notions de réactivité, hétérogénéité, distribution doivent être considérées.
Les approches qui utilisent model-checking se sont largement développées ces derniers temps. Ces approches doivent prendre en considération les propriétés temporelles. Comme les modèles utilisant un temps global ne reflètent pas réellement les aspects des systèmes distribués, nous avons supposé dans ce travail que les différents composants d'un système distribué évoluent selon des rythmes différents.
Nous avons appliqué notre approche dans le domaine des systèmes temps réel hétérogènes . Ce domaine est toujours présent dans les applications informatiques. Les contributions de ce travail peuvent se synthétiser comme suit :
Nous avon proposé une approche pour l'analyse des systèmes temps-réel hétérogènes. Les horloge associées aux différents processus évoluent selon des fréqunece différentes mais relatives . Nous avons montré la décidabilité de l'accessibilité en utilisant une nouvelle abstraction de comportements des systèmes. Cette abstraction a nécessité la redéfinition de plusieurs concepts nécessaires.
Dans les systèmes temps réel hétérogènes et coordonnés, bien que les composants puissent être hétérogènes, nous avons proposé le modèle R-TA pour interpréter les comportements temporels.
Pour étudier la sémantique de tels systèmes, nous avons défini le paramètre "slope". Ce paramètre nous a permis de redéfinir la relation d'équivalence sur les valuations d'horloges d'un R-TA.
Nous avons présenté que le nombre de régions d'horloges associées à un R-TA est fini. Par conséquent, nous avons redéfini la relation de successeur sur ces régions d'horloges pour pouvoir construire l'automate de régions. Cette possibilité de construction nous offre la décidabilité pour le test du vide et model-checking.

Pour avoir une construction d'un automate de régions qui se termine, il faut avoir un index fini de régions et une relation de successeur sur cet index. En effet, cet automate de régions spécifie les mêmes comportements que ceux impliqués par la spécification R-TA.

6.2 Perspectives

Comme perspectives, nous voulons étudier les capacités de notre modèle pour décrire la sémantique des spécifications des systèmes ambiants.
L'application de l'approche model-checking demande une extension de ces résultats comme la construction de l'automate de zones qui est utile pour l'implémentabilité des outils de vérification.
Des concepts intéressants, qui peuvent être bénéfiques, ont déjà été développés par d'autres modèles, par exemple les automates hybrides Henzinger (2000) et les automates avec prix Larsen (2009). Cela nous motive à étudier la possibilité d'intégrer des concepts similaires au sein du notre approche.
Aussi, nous voulons également explorer des exemples dans des applications réelles et établir une comparaison formelle avec d'autres modèles de spécification célèbres tels que les réseaux de Petri et ses diverses extensions.

Bibliographie

Abdulla, P. A., Deneux, J., Ouaknine, J., and Worrell, J. (2005). Decidability and complexity results for timed automata via channel machines. In *International Colloquium on Automata, Languages, and Programming*, pages 1089–1101. Springer.

Aceto, L., Bouyer, P., Burgueno, A., and Larsen, K. G. (2003). The power of reachability testing for timed automata. *Theoretical Computer Science*, 300(1) :411–475.

Aceto, L., Burgueno, A., and Larsen, K. G. (1998). Model checking via reachability testing for timed automata. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 263–280. Springer.

Aceto, L. and Laroussinie, F. (2002). Is your model checker on time ? On the complexity of model checking for timed modal logics. *The Journal of Logic and Algebraic Programming*, 52 :7–51.

Ahola, J. (2002). Ambient Intelligence : Plenty of Challenges by 2010. In *Advances in Database Technology — EDBT 2002*, pages 14–14. Springer Berlin Heidelberg.

Akshay, S., Bollig, B., Gastin, P., Mukund, M., and Kumar, K. N. (2014). Distributed Timed Automata with Independently Evolving Clocks.

Akshay, S., Bollig, B., Gastin, P., Mukund, M., and Narayan Kumar, K. (2008). Distributed Timed Automata with Independently Evolving Clocks. In *CONCUR 2008-Concurrency Theory*, chapter Distribute, pages 82–97. Springer Berlin Heidelberg.

Alur, R., Courcoubetis, C., and Dill, D. (1993). Model-checking in dense real-time. *Information and computation*, 104(1) :2–34.

Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T., Ho, P.-H., Nicollin, X., Olivero, a., Sifakis, J., and Yovine, S. (1995). The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1) :3–34.

Alur, R. and Dill, D. (1990). Automata for modeling real-time systems. In *International Colloquium on Automata, Languages, and Programming*, pages 322–335. Springer.

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

Alur, R., Feder, T., and Henzinger, T. A. (1996). The benefits of relaxing punctuality. *Journal of the ACM (JACM)*, 43(1) :116–146.

Alur, R. and Henzinger, T. (1993). Real-Time Logics : Complexity and Expressiveness. *Information and Computation*, 104(1) :35–77.

Alur, R. and Henzinger, T. A. (1991). Logics and models of real time : A survey. In *Workshop/School/Symposium of the REX Project (Research and Education in Concurrent Systems)*, pages 74–106. Springer.

Alur, R. and Henzinger, T. A. (1994). A really temporal logic. *Journal of the ACM (JACM)*, 41(1) :181–203.

Baier, C. and Katoen, J.-P. (2008). *Principles of model checking*. MIT press.

Basagni, S., Chlamtac, I., and Syrotiuk, V. (2000). Location Aware One-to-Many Communication in Mobile Multi-Hop Wireless Networks. In *Proceedings of the 51st IEEE Semiannual Vehicular Technology Conference, VTC 2000 Spring*, volume 1, pages 288–292.

Behrmann, G., David, A., and Larsen, K. G. (2004). A Tutorial on Uppaal. In *Formal Methods for the Design of Real-Time Systems*, volume 3185, pages 200–236.

Bengtsson, J., Griffioen, W. O. D., Kristoffersen, K. J., Larsen, K. G., Larsson, F., Pettersson, P., and Yi, W. (2002). Automated verification of an audio-control protocol using UPPAAL. *The Journal of Logic and Algebraic Programming*, 52 :163–181.

Bérard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., and Schnoebelen, P. (2013). *Systems and software verification : model-checking techniques and tools*. Springer Science & Business Media.

Bérard, B. and Dufourd, C. (2000). Timed automata and additive clock constraints. *Information Processing Letters*, 75(1) :1–7.

Bérard, B. and Fribourg, L. (1999). Automated verification of a parametric real-time program : the ABR conformance protocol. In *International Conference on Computer Aided Verification*, pages 96–107. Springer.

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

Berthomieu, B. and Menasche, M. (1983). An enumerative approach for analyzing time Petri nets. In *Proceedings IFIP 9th World Computer Congress*. Citeseer.

Bouyer, P. (2004). Forward analysis of updatable timed automata. *Formal Methods in System Design*, 24(3) :281–320.

Bouyer, P. and Chevalier, F. (2005). On conciseness of extensions of timed automata. *Journal of Automata, Languages and Combinatorics*, 10(4) :393–405.

Bouyer, P., Chevalier, F., and Markey, N. (2005). On the expressiveness of TPTL and MTL. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 432–443. Springer.

Bouyer, P., Dufourd, C., Fleury, E., and Petit, A. (2004). Updatable timed automata. *Theoretical Computer Science*, 321(2-3) :291–345.

Chlamtac, I., Conti, M., and Liu, J. J.-N. (2003). Mobile ad hoc networking : imperatives and challenges. *Ad hoc networks*, 1(1) :13–64.

Cimatti, A., Clarke, E., Giunchiglia, F., and Roveri, M. (2000). NuSMV : a new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2(4) :410–425.

Clarke, E. M., Grumberg, O., and Peled, D. (1999). *Model checking*. MIT press.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to algorithms*, volume 6. MIT press Cambridge.

Courcoubetis, C. and Yannakakis, M. (1992). Minimum and maximum delay problems in real-time systems. *Formal Methods in System Design*, 1(4) :385–415.

Dill, D. L. (1989). Timing assumptions and verification of finite-state concurrent systems. In *International Conference on Computer Aided Verification*, pages 197–212. Springer.

Emerson, E. A., Mok, A. K., Sistla, A. P., and Srinivasan, J. (1992). Quantitative temporal reasoning. *Real-Time Systems*, 4(4) :331–352.

Fersman, E., Pettersson, P., and Yi, W. (2002). Timed Automata with Asynchronous Processes : Schedulability and Decidability. In *Lecture Notes in Computer Science*, pages 67–82.

Hennessy, M. and Milner, R. (1985). Algebraic laws for nondeterminism and concurrency. *Journal of the ACM (JACM)*, 32(1) :137–161.

Henzinger, T. A. (2000). The theory of hybrid automata. In *Verification of Digital and Hybrid Systems*, pages 265–292. Springer.

Henzinger, T. A., Ho, P.-H., and Wong-Toi, H. (1997). HYTECH : a model checker for hybrid systems. *International Journal on Software Tools for Technology Transfer*, 1(1-2) :110–122.

Henzinger, T. A., Kopke, P. W., Puri, A., and Varaiya, P. (1998). What's Decidable about Hybrid Automata ? *Journal of Computer and System Sciences*, 57(1) :94–124.

Henzinger, T. A., Nicollin, X., Sifakis, J., and Yovine, S. (1994). Symbolic model checking for real-time systems. *Information and computation*, 111(2) :193–244.

Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2001). Introduction to automata theory, languages, and computation, 2nd edition.

Koymans, R. (1990). Specifying real-time properties with metric temporal logic. *Real-time systems*, 2(4) :255–299.

Laroussinie, F., Markey, N., and Schnoebelen, P. (2004). Model checking timed automata with one or two clocks. In *International Conference on Concurrency Theory*, pages 387–401. Springer.

Laroussinie, F., Markey, N., and Schnoebelen, P. (2006). Efficient timed model checking for discrete-time systems. *Theoretical Computer Science*, 353(1) :249–271.

Laroussinie, F. and Schnoebelen, P. (2000). The state explosion problem from trace to bisimulation equivalence. In *International Conference on Foundations of Software Science and Computation Structures*, pages 192–207. Springer.

Laroussinie, F., Schnoebelen, P., and Turuani, M. (2003). On the expressivity and complexity of quantitative branching-time temporal logics. *Theoretical Computer Science*, 297(1) :297–315.

Larsen, K. G. (1990). Proof systems for satisfiability in Hennessy-Milner logic with recursion. *Theoretical Computer Science*, 72(2) :265–288.

Larsen, K. G. (2009). Priced timed automata : Theory and tools. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 4. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

Larsen, K. G., Larsson, F., Pettersson, P., and Yi, W. (1997a). Efficient verification of real-time systems : compact data structure and state-space reduction. In *Real-Time Systems Symposium, 1997. Proceedings., The 18th IEEE*, pages 14–24. IEEE.

Larsen, K. G., Pettersson, P., and Yi, W. (1995). Model-checking for real-time systems. In *International Symposium on Fundamentals of Computation Theory*, pages 62–88. Springer.

Larsen, K. G., Pettersson, P., and Yi, W. (1997b). Uppaal in a nutshell.

Larsen, K. G., Yi, W., and Pearson, J. (1999). Clock Difference Diagrams. *Nordic Journal of Computing-NJC*, 6(December) :271—-298.

Markey, N. and Schnoebelen, P. (2004). Symbolic model checking for simply-timed systems. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 102–117. Springer.

Ouaknine, J. and Worrell, J. (2004). On the language inclusion problem for timed automata : Closing a decidability gap. In *Logic in Computer Science, 2004. Proceedings of the 19th Annual IEEE Symposium on*, pages 54–63. IEEE.

Raskin, J.-F. (1999). *Logics, automata and classical theories for deciding real time*. PhD thesis.

Raskin, J.-F. (2005). An introduction to hybrid automata. In *Handbook of networked and embedded control systems*, pages 491–517. Springer.

Sarkar, S. K., Basavaraju, T., and Puttamadappa, C. (2007). *Ad Hoc Mobile Wireless Networks : Principles, Protocols, and Applications*. CRC Press.

Stirling, C. (2001). *Modal and Temporal Properties of Processes*. Texts in Computer Science. Springer New York, New York, NY.

Tripakis, S. and Yovine, S. (1998). Verification of the fast reservation protocol with delayed transmission using the tool kronos. In *Real-Time Technology and Applications Symposium, 1998. Proceedings. Fourth IEEE*, pages 165–170. IEEE.

Wang, Y. (1990). Real-time behaviour of asynchronous agents. In *International Conference on Concurrency Theory*, pages 502–520. Springer.

Yovine, S. (1997). Kronos : A verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1) :123–133.

Zimmerman, T. G. (1996). Personal Area Networks : Near-field intrabody communication. *IBM Systems Journal*, 35(3.4) :609–617.