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 : Génie logiciel avancé et applications (GLAA)

thème

Utilisation des régions pour la vérification formelle des systèmes temps-réel hétérogènes

Réalisé par :
— BOUCHEMAA Abir
— BOUTEBENTA Fatima Zahra

Encadré par :
LAYADI saïd

Session : Juin 2023

Résumé

Actuellement, les méthodes formelles sont de plus en plus fréquemment employées pour examiner le comportement des systèmes temps-réels. Ces systèmes font souvent partie d'applications critiques, où une erreur durant leur fonctionnement peut avoir des conséquences graves, notamment dans des domaines sensibles tels que l'avionique, le contrôle de processus industriels ou encore la gestion de centrales nucléaires. Par conséquent, il est impératif de vérifier ces systèmes avant leur mise en service.

Les méthodes de vérification formelle, ou simplement les méthodes formelles, sont de plus en plus utilisées pour répondre aux exigences auxquelles sont soumis 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 permettant la construction d'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 utilisé un paramètre appelé "slope", qui est le rapport entre les fréquences des horloges.

De plus, la relation d'équivalence entre les valuations d'horloges a été redéfinie afin de passer à des graphes de régions finis et pouvoir appliquer une approche formelle de vérification telle que le 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 tenons tout d'abord à exprimer notre gratitude envers Dieu, le tout-puissant, pour nous avoir accordé la force, la volonté et la santé nécessaires pour mener à bien ce mémoire.

Nous souhaitons sincèrement remercier Monsieur **LAYADI SAID** pour son encadrement exceptionnel, ses conseils précieux, sa disponibilité et sa rigueur tout au long de notre préparation. Sans son expertise et son soutien, ce travail n'aurait pas atteint sa qualité actuelle. Nous lui sommes profondément reconnaissants.

Nous exprimons également notre gratitude envers le jury qui a accepté de consacrer son temps et son expertise pour évaluer notre mémoire. Nous sommes honorés de votre présence et nous vous remercions de votre intérêt pour notre travail. Nous souhaitons également remercier

tous nos professeurs pour leur générosité, leur enseignement et leur patience malgré leurs nombreuses responsabilités académiques et professionnelles.

Enfin, nos remerciements s'adressent à toutes les personnes qui nous ont soutenues de près ou de loin tout au long de ce parcours. Votre soutien moral et votre encouragement ont été essentiels pour nous permettre d'accomplir ce travail. Nous sommes conscients que ce

mémoire n'aurait pas été possible sans l'appui de toutes ces personnes. Leur contribution a été inestimable et nous leur sommes profondément reconnaissants.

Merci infiniment à tous ceux qui ont contribué de quelque manière que ce soit à la réalisation de ce mémoire.

Dédicace

Ce mémoire est le fruit d'un travail acharné, mais il n'aurait jamais pu voir le jour sans le soutien et l'encouragement de nombreuses personnes exceptionnelles. Ainsi, nous tiens à dédier ce mémoire à tous ceux qui ont contribué de près ou de loin à sa réalisation. À

notre famille bien-aimée **BOUCHEMAA et LAIB , BOUTEBANTA et GHARBI** de nos **Pères, Mères, Sœurs, Frères** et même de nos maris **MOHAMMED ,RIDA** Nous souhaitons exprimer notre profonde gratitude et notre amour infini pour vous tous. Vous êtes le pilier de notre vie, notre source de force et notre refuge dans les moments de joie et de difficultés. Votre soutien inconditionnel et votre présence constante ont été essentiels pour notre parcours académique et pour la réalisation de ce mémoire.

Vous avez été là à chaque étape de notre vie, nous encourageant à poursuivre nos rêves et à atteindre nos objectifs. Votre soutien financier, émotionnel et moral nous a permis de nous concentrer sur nos études et de donner le meilleur de nous-mêmes dans chaque tâche que nous avons entrepris,nous sommes conscients des sacrifices que vous avez consentis pour nous permettre de poursuivre nos études.
Vous avez fait preuve de dévouement et de générosité en nous offrant un environnement

propice à l'apprentissage, en nous soutenant dans nos besoins matériels et en nous inspirant par votre exemple de persévérance.
À travers les hauts et les bas de notre parcours académique, votre amour et votre soutien ont été constants. Vous avez célébré nos réussites avec fierté et vous avez été là pour nous réconforter et nous encourager lorsque nous avons rencontré des obstacles. Votre confiance en nos capacités a été un moteur puissant pour nous pousser à aller de l'avant.
Nous tenons à vous remercier du fond du cœur pour tout ce que vous avez fait et continuez de faire pour nous. Votre amour inconditionnel est le moteur qui nous inspire et nous donne la force de poursuivre nos rêves. Nous sommes fiers d'appartenir à une famille aussi aimante et solidaire.
Que notre amour et notre gratitude vous accompagnent chaque jour. Nous l'espérons que notre réussite académique saidsera une source de fierté pour vous tous, car c'est grâce à votre soutien indéfectible que nous avons pu réaliser ce mémoire.

À nos professeur et à notre encadrant **LAYADI SAID** , pour expertise, sa patience et son dévouement à transmettre ses connaissances. Vos conseils avisés, votre mentorat précieux et votre disponibilité ont façonné notre formation et ont contribué à la réussite de ce mémoire.

Enfin, à nous-mêmes, nous dédions ce mémoire en signe de fierté et de reconnaissance envers notre propre persévérance et notre détermination. Nous sommes fiers d'avoir relevé les défis qui se sont présentés à nous et d'avoir accompli cette étape importante de notre parcours.
Avec tout notre amour et notre reconnaissance...

Table des matières

Table des figures

List of Algorithms

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

TCTL & **T**imed **C**omputation **T**ree **L**ogic

Chapitre 1

Introduction générale

Sommaire

1.1 Contexte de travail

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

La vérification des systèmes temps réel hétérogènes est un domaine d'étude très intéressant. Les systèmes temps réel sont de plus en plus présents dans notre quotidien et sont souvent distribués, ce qui signifie qu'ils sont composés de plusieurs éléments qui interagissent entre eux. Les réseaux ad-hoc mobiles (MANET) sont un exemple de tels systèmes distribués complexes, où des nœuds mobiles sans fil s'auto-organisent pour former des réseaux de communication.Chlamtac et al. (2003)

La vérification des systèmes temps réel hétérogènes est importante pour garantir leur bon fonctionnement et leur conformité aux spécifications. Les méthodes formelles, telles que le model-checking, sont souvent utilisées pour cette tâche. Le model-checking consiste à modéliser le comportement du système et à vérifier si certaines propriétés souhaitées sont satisfaites par ce modèle.Bérard et al. (2013)

Dans les systèmes temps réel hétérogènes, chaque composant peut avoir ses propres horloges physiques avec leur propre fréquence de progression. Il n'y a généralement pas d'horloge globale synchronisée pour l'ensemble du système. Cela signifie que les composants peuvent évoluer à des rythmes différents.Burns and Wellings (2009) Kopetz et al. (2011)

Les modèles de spécification des systèmes temps réel doivent prendre en compte les aspects temporels, et les contraintes temporelles sont souvent exprimées à l'aide d'horloges. Les horloges sont utilisées pour capturer les délais, les durées et d'autres contraintes temporelles du

système.Alur and Dill (1994b)

Les automates temporisés sont un modèle couramment utilisé pour décrire le comportement des systèmes temps réel avec des contraintes quantitatives. Les automates temporisés possèdent des horloges qui s'incrémentent automatiquement avec le temps et comptent les délais entre les différentes actions du système. Les techniques de vérification, telles que le model-checking, ont été étendues pour être compatibles avec ce modèle.Alur and Dill (1994b)

Il existe différentes approches de vérification pour les systèmes temps réel hétérogènes. La sémantique universelle capture les comportements qui sont maintenus indépendamment des fréquences des horloges individuelles. Cela est utile lorsqu'on veut vérifier si un système satisfait toujours une spécification positive. En revanche, la sémantique existentielle prend en compte l'ensemble des comportements possibles du système sous différents choix de fréquences d'horloges. Cela permet de vérifier des spécifications négatives.

Cependant, il convient de noter que la vérification des systèmes temps réel hétérogènes peut être un problème complexe et parfois indécidable. Certains problèmes de vérification peuvent ne pas avoir de solution algorithmique générale et nécessitent des approches spécifiques pour être résolus.

1.1.2 Motivations du travail

Dans ce travail, l'approche consiste à modéliser et vérifier les systèmes distribués avec des fréquences locales et relatives d'horloges. Chaque composant du système est décrit par un automate temporisé dans lequel toutes les horloges progressent selon la même fréquence. Cependant, les horloges appartenant à différents processus (composants) peuvent évoluer selon des fréquences différentes mais relatives. On suppose qu'une horloge peut être lue (utilisée) par tous les processus, mais sa remise à zéro ne peut être effectuée que par le processus auquel elle appartient.
Pour agréger les configurations du système et résoudre le problème de l'infini nombre de configurations, des relations d'équivalence sur les valuations d'horloges ont été proposées. Ces relations d'équivalence regroupent les configurations en classes d'équivalence appelées régions d'horloges.Alur and Dill (1994a)

Étant donné que le comportement futur d'un système modélisé est déterminé à tout instant par sa localité actuelle et les valeurs des horloges de tous les processus, de nombreux concepts tels que les régions d'horloges et l'automate de régions sont redéfinis en prenant en compte l'effet de la relativité entre les fréquences d'horloges.

Dans ce contexte, le problème de vérifier la satisfaction temporelle entre les composants coordonnés d'un système temps réel hétérogène peut être réduit à une recherche sur un graphe de régions.

Les automates temporisés ont été étendus par le temps relatif (r-TA) en caractérisant les vitesses relatives du temps associées aux composants pour faire face à l'hétérogénéité.

1.2 Contributions

Dans la littérature, les chercheurs se sont intéressés à la modélisation et à la vérification des systèmes temps réel hétérogènes, c'est-à-dire des systèmes où différents composants ont leurs propres horloges physiques avec des fréquences de progression différentes. L'objectif était de modéliser ces systèmes et de vérifier leurs propriétés comportementales et temporelles.

Pour cela, ils ont étendu le modèle des automates temporisés(TA) en introduisant la notion de temps relatif, noté r-TA. Dans un r-TA, les horloges des différents composants évoluent de manière synchrone, mais avec des rythmes relatifs qui leur sont assignés. Chaque rythme représente la vitesse du composant et dépend d'une référence temporelle globale.

Les chercheurs ont défini un paramètre appelé "slope", qui correspond au rapport entre les fréquences des horloges. Ce paramètre leur a permis d'évaluer et de prouver la décidabilité de l'accessibilité dans un r-TA.

Pour étudier la sémantique de ces systèmes, les chercheurs ont redéfini l'algorithme de construction d'un automate de régions. Cet automate de régions est basé sur deux nouvelles relations : la relation d'équivalence, qui concerne l'ensemble de toutes les valuations d'horloges, et la relation du successeur, qui s'applique à ces classes d'équivalence. Ces deux relations ont permis de proposer une méthode de construction d'un automate de régions fini, qui représente un ensemble fini de comportements spécifiés par le r-TA.

Les chercheurs ont démontré que la construction de l'automate de régions aboutit à un espace fini de régions. Cette construction offre la décidabilité pour le test du vide et le model-checking, c'est-à-dire la vérification de propriétés du système.

En résumé, l'objectif de ce mémoire est d'utiliser le modèle r-TA et de proposer des algorithmes de construction des abstractions finies pour vérifier les systèmes temps réel hétérogènes. Comme l'automate de régions construit à partir du r-TA représente une abstraction finie du système réel, la décidabilité de la propriété d'accessibilité a été prouvée. Cette approche permet de prendre en compte les contraintes temporelles spécifiques aux systèmes hétérogènes et offre des techniques de vérification efficaces.

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 Cette structure de chapitre permet de donner une vision globale des systèmes temps réels , des systèmes critiques et des systèmes hétérogènes , tout en offrant un exemples des systèmes de control au domaine de notre stage.

Le chapitre 03 du manuscrit se concentre sur les automates temporisés et la vérification

formelle. Il présente la syntaxe et la sémantique des automates temporisés, ainsi que leurs propriétés. Une construction basée sur les régions est également abordée pour assurer la décidabilité de la propriété d'accessibilité.
Ensuite, le chapitre explore la vérification automatique basée sur les modèles (model checking) en examinant différentes approches, logiques et techniques telles que la logique de temps, les automates de test et l'équivalence comportementale.
Enfin, le chapitre présente le model checking sur TCTL, une logique utilisée spécifiquement pour la vérification de systèmes temporisés. En résumé, ce chapitre offre une compréhension approfondie des automates temporisés et de leur utilisation dans la vérification formelle.

Le chapitre 04 du manuscrit aborde les automates temporisés avec temps relatif (r-TA) et les problèmes associés. Il présente différentes modélisations des systèmes temporisés, notamment les automates temporisés à temps absolu et les automates temporisés avec vitesses relatives du temps. La sémantique des r-TA est également expliquée. Ensuite, le chapitre se concentre sur les problèmes à résoudre, tels que les paramètres de pente, les classes d'équivalence sur les évaluations d'horloges, la représentation et les successeurs des régions d'horloges. L'automate de régions du r-TA et un algorithme de construction à partir d'un r-TA sont également présentés. En résumé, ce chapitre explore en détail les automates temporisés avec temps relatif et propose des solutions pour résoudre les problèmes liés à leur utilisation.

Le chapitre 05 du manuscrit se concentre sur l'implémentation de l'approche proposée. Il présente les diagrammes de classes pour l'automate temporisé avec temps relatif et l'automate de régions. Il explique également les environnements utilisés dans la mémoire. Ensuite, il décrit la structure du code source, y compris l'interface principale, et fournit des informations sur le code source spécifique pour schématiser la structure du R-TA. Ce chapitre offre un aperçu de l'implémentation pratique de l'approche proposée, en fournissant des diagrammes de classes et des explications sur la structure du code source utilisé.

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 et critiques

Sommaire

2.1 Introduction

Les systèmes temps-réel hétérogènes et critiques sont des systèmes informatiques complexes qui interagissent avec leur environnement en temps réel. Leur fonctionnement optimal est primordial pour garantir la sécurité et la fiabilité de leur utilisation. Ces systèmes sont généralement constitués de plusieurs éléments hétérogènes, tels que des processeurs, des capteurs, des actionneurs et des logiciels, qui doivent tous fonctionner de manière coordonnée pour assurer le bon fonctionnement du système. Les systèmes temps-réel hétérogènes et critiques sont largement utilisés dans de nombreux domaines, notamment l'aéronautique, l'automobile, les systèmes de contrôle industriel, les systèmes de surveillance et de sécurité, entre autres.

La conception et la mise en œuvre de ces systèmes sont des tâches complexes qui nécessitent une approche rigoureuse et une vérification formelle pour garantir leur sécurité et leur fiabilité. Les méthodes et les outils de vérification formelle permettent de prouver des propriétés importantes sur ces systèmes, telles que l'absence de comportements indésirables, la conformité aux spécifications et la sécurité. Ces méthodes et outils sont donc indispensables pour garantir la qualité et la sécurité des systèmes temps-réel hétérogènes et critiques.

2.2 Systèmes temps réel

En générale, les systèmes temps réel sont présents dans de nombreux domaines d'activités comme le transport, les télécommunications et l'industrie. Un système temps réel se présente comme un système réactif ayant une spécification temporelle. Il est vu comme un environnement à contrôler adjoint d'un système informatique de contrôle (figure 2.1). Plus précisément, un système temps réel est en interaction permanente avec son environment externe, représenté généralement par un procédé physique, afin de contrôler son comportement évoluant dans le temps. Le système doit être capable de détecter les changements d'états de l'environnement et de réagir sur celui-ci, en réalisant des traitements permettant de produire un changement d'état 'a la sortie, dans un intervalle de temps limité L'interaction du système avec l'environnement se fait par acquisition d'informations provenant des capteurs sous forme d'interruptions ou de mesures, et par réaction sur l'environnement en envoyant des affichages ou des commandes via les actionneurs.Mehiaoui (2014)

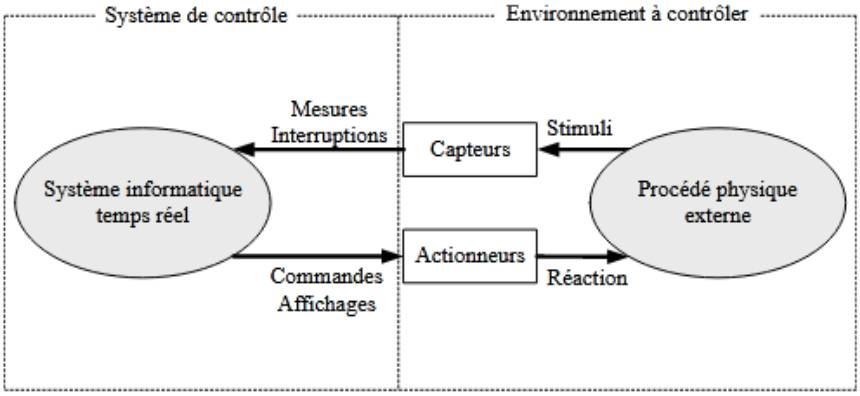


FIGURE 2.1 – Représentation schématique d'un système temps réel

Il existe dans la littérature plusieurs définitions de système temps réel, on prend par exemple les deux définitions suivantes :

Définition 1

" Un système fonctionne en temps réel s'il est capable d'absorber toutes les informations d'entrée qu'elles soient trop vieilles pour l'intérêt qu'elles présentent, et par ailleurs, de réagir à celles-ci suffisamment vite pour que cette réaction ait un sens."Berrou (2010)

Définition 2

"Les systèmes informatiques temps réel seraient donc ceux pour lesquels le comportement temporel, non seulement en terme de séquence d'opérations (propriété logique), mais également en terme de quantification de l'écoulement de ces opérations (propriété physique), a une importance."Berrou (2010)

2.2.1 Les catégories des systèmes temps réel

-Systèmes temps réel à contraintes strictes/dures (hard real time constraints)

Où le non-respect des contraintes de temps peut conduire à des défaillances avec des conséquences pouvant être graves. Si l'échéance est dépassée, il y a faute. Exemples : Contrôle aérien, contrôle d'une centrale nucléaire, etc.(figure 2.2) suivante illustre que une réponse dépassant son échéance est considéré comme réponse fautive.KHITER and HELLAL (2020)

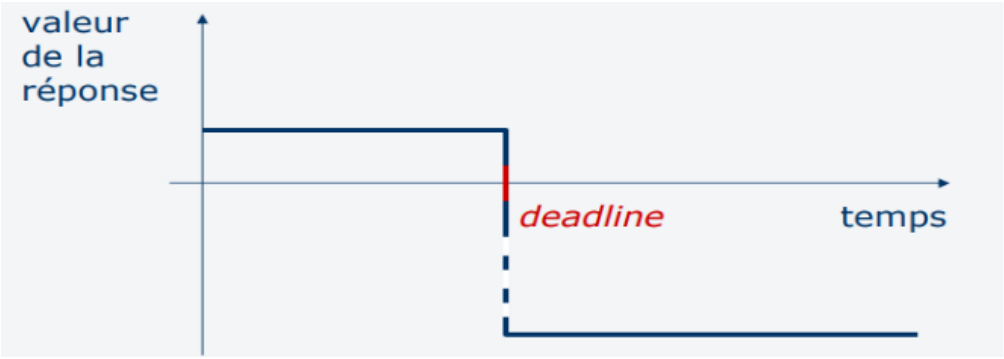


FIGURE 2.2 – système temps reél dur

-Systèmes temps réel à contraintes relatives/souples (soft real time constraints) :

Où le dépassement des échéances est considéré comme une faute bénigne. Lorsqu' une échéance est dépassée, il n'y a pas faute ; le résultat peut être exploitable même s'il est fourni après l'échéance. Exemples : jeux vidéo, logiciel embarqué de votre téléphone, iPod, etc.(figure 2.3) suivante illustre qu'une réponse dépassant son échéance peut être acceptée 'a condition que la faute temporelle n'est pas grande.

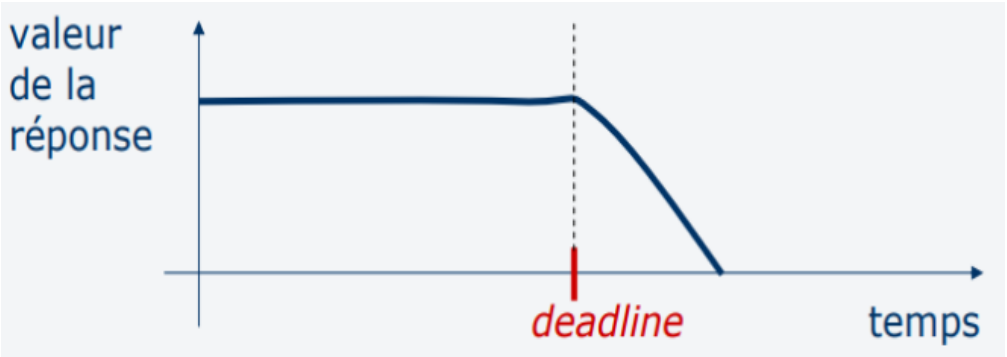


FIGURE 2.3 – système temps reél souple

-Systèmes temps réel à contraintes fermes (firm real time contraintes) :

Où le dépassement occasionnel des échéances est toléré. Il y a faute (bénigne) si l'échéance n'est pas respectée. Exemples : transactions en bourse : le temps réel ferme de l'un peut être le temps réel dur de l'autre.(figure 2.4) suivante montre que le dépassement de l'échéance n'arrête pas la tâche.

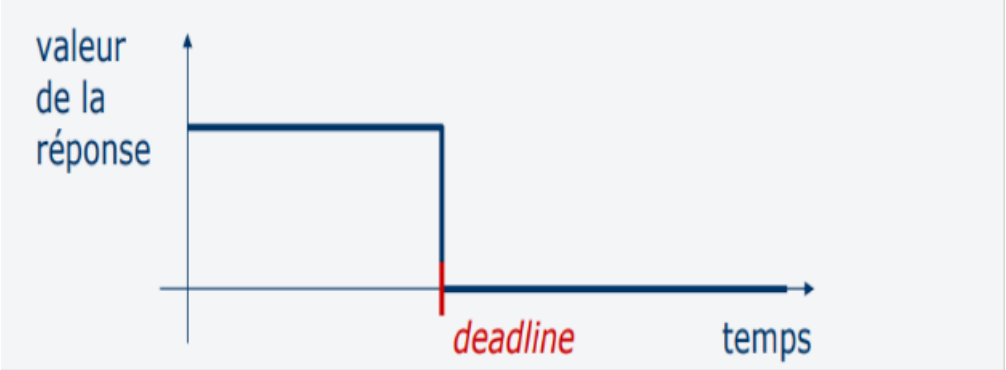


FIGURE 2.4 – système temps reél ferme

2.2.2 Modélisation des systèmes temps réel

Il existe plusieurs formalismes pour modéliser un système temps réel, telles que les méthodes formelles (logique temporelle, algèbre de processus) et les méthodes structurées (système réactif synchrone et asynchrone, réseau de Pétri), (figure 2.5)

La spécification de ces systèmes est réalisée en trois phases complémentaires et dépendantes.

- la spécification fonctionnelle.
- la spécification architecturale.
- la spécification des contraintes.

La spécification fonctionnelle consiste à définir l'algorithme avec ses exigences fonctionnelles.

La spécification architecturale consiste à définir l'architecture matérielle qui doit implanter l'algorithme.

La spécification des contraintes consiste à attribuer des propriétés temporelles et matérielles à l'exécution de l'algorithme sur l'architecture.Berrou (2010)

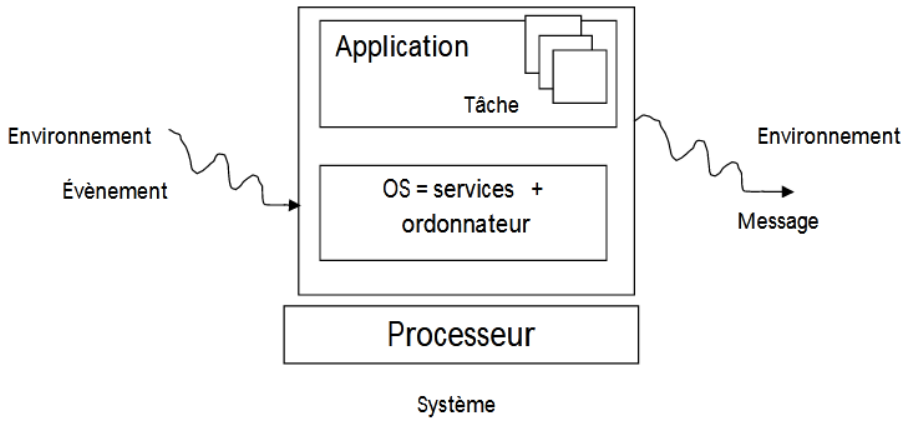


FIGURE 2.5 – Modélisation des systèmes temps reél

2.2.3 Caractéristiques d'un système temps réel

La **dynamicité**, l'**hétérogénéité** et la **concurrence** sont des caractéristiques importantes des systèmes temps-réel.

- **Dynamicité :** un système est vu comme un espace ouvert.
  - **Ex des MANET :** la dynamicité est exprimée en termes de mobilité où les composants mobiles peuvent dynamiquement s'auto-organiser en topologies de réseaux arbitraires.
- **Hétérogénéité :** vue comme une différence entre les vitesses de fonctionnement des différents composants.Layadi (2017)

2.3 les systèmes critiques

2.3.1 Définition

Les systèmes critiques pour la sécurité sont des systèmes dont la défaillance pourrait entraîner la perte de vies humaines, des dommages matériels importants ou des dommages à l'environnement. Il existe de nombreux exemples bien connus dans des domaines d'application tels que les dispositifs médicaux, les commandes de vol des avions, les armes et les systèmes nucléaires. De nombreux systèmes d'information modernes deviennent critiques pour la sécurité au sens général du terme, car leur défaillance peut entraîner des pertes financières, voire des pertes de vie. Les futurs systèmes critiques pour la sécurité seront plus courants et plus puissants. Du point de vue des logiciels, le développement de systèmes critiques pour la sécurité en nombre suffisant et avec une fiabilité adéquate va nécessiter des avancées significatives dans des domaines tels que la spécification, l'architecture, la vérification et le processus. Les problèmes très visibles qui sont apparus dans le domaine de la sécurité des systèmes d'information suggèrent que la sécurité constitue également un défi majeur.Knight (2002) Il existe de plus en plus de logiciels dans les systèmes critiques et leur processus de conception et de développement doivent être rigoureux et conduire à des modèles fiables. Notre intérêt ne se limite pas à la conception de systèmes critiques et s'étend à une définition plus large de systèmes qualifiés de réactifs Luong (2010)

2.3.2 Types de systèmes critiques

Il existe trois types de systèmes critiques :

Système critique pour la sécurité

Toute défaillance de ces systèmes entraîne des blessures, la mort ou des dommages à l'environnement. Par exemple, le système d'une usine chimique.

Systèmes essentiels à la mission

Toute défaillance de ces systèmes entraîne l'échec de certains objectifs prévus. Par exemple, le système de navigation d'un engin spatial.

Système critique pour l'entreprise

Toute défaillance de ces systèmes entraîne des pertes financières importantes. Par exemple, le système de comptabilité de la banque.Shafei et al. (2013)

2.3.3 Systèmes logiciel critiques de sécurité

Les SCSS peuvent être définis comme des logiciels qui surveillent, exerce une commande et un contrôle directs sur la condition ou l'état des composants matériels. Et s'il n'est pas exécuté, s'il est exécuté hors séquence, ou mal exécutées, elles peuvent entraîner des fonctions de contrôle, ce qui pourrait causer un danger ou permettre l'existence d'une condition dangereuse. Les SCSS exigent un travail rigoureux en matière d'analyse de la sécurité, test et vérification de la sécurité pour assurer la sécurité du système global. Les systèmes logiciels sont considérés comme critiques pour la sécurité s'ils remplissent l'une des fonctions suivantes :

- Implémenter un processus critique de prise de décision.
- Contrôler ou surveiller les fonctions critiques de sécurité du logiciel ou du matériel.
- Causer ou contribuer à des dangers.
- Intervenir lorsqu'une condition non sécuritaire est présente ou imminente.
- Exécuter sur le même système cible que les logiciels de sécurité.
- Atténuer les dommages en cas de risque.

Shafei et al. (2013)

2.3.4 Méthodologie des exigences pour la planification des systèmes logiciels critiques

La méthodologie proposée décrit le développement de SCSS certifiés sur la base de normes et de directives. Cette méthodologie se compose de trois phases(phase de planification de la sécurité et des exigences, phase d'analyse et phase de conception, de mise en œuvre et-d'exploitation) comme le montre la figure **??**. La phase de planification de la sécurité et des exigences se compose de quatre processus, comme le montre la figure **??**. Elle commence par la description du système critique et de ses composants, de la relation entre les composants,

de l'identification des besoins en matière de sécurité et de l'évaluation des risques. Entre les composants, l'identification des fonctions du système critique, la spécification des SCSS plan de sécurité, et enfin l'identification des fonctions des fonctions des SCSS.Shafei et al. (2013)

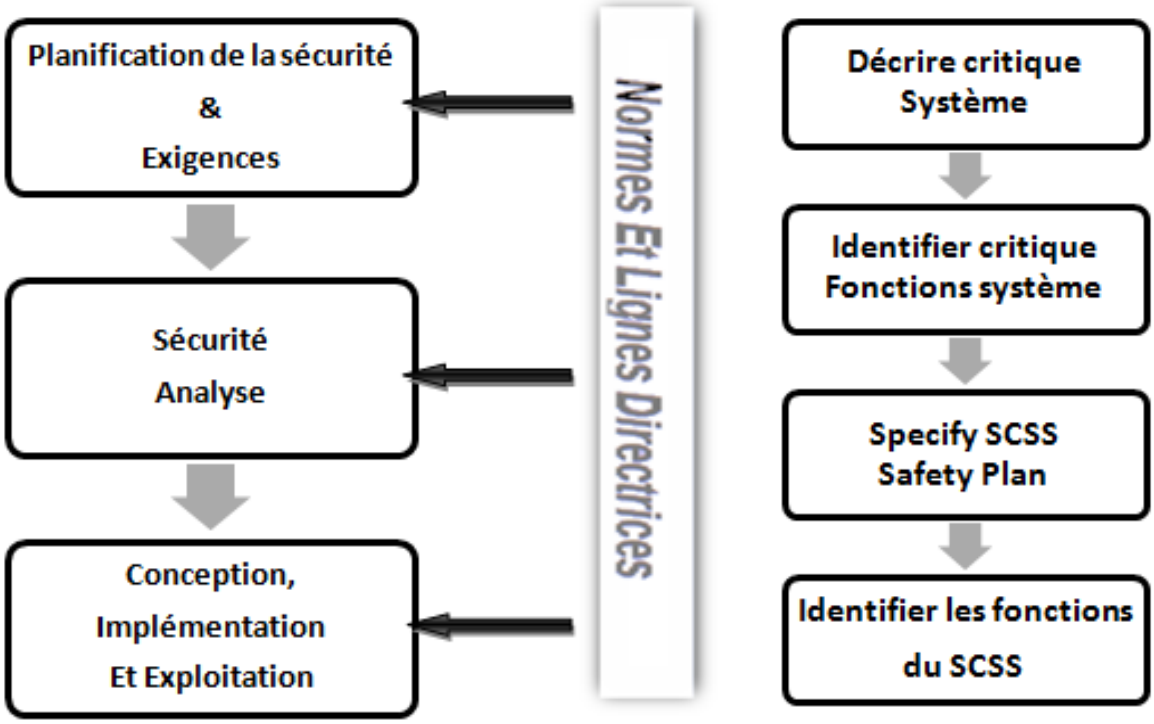


FIGURE 2.6 – Développement des SCSS Méthodologie & Processus de planification de la phase des exigences

2.4 Les systèmes hétérogènes

2.4.1 problèmatique

Les systèmes hétérogènes nécessitent donc des outils spécialisés qui sont suffisamment génériques pour pouvoir s'adapter à différentes . Des outils existent aujourd'hui qui permettent l'analyse de processeurs spécialisés. uniquement compatibles avec un modèle de programmation .
De plus, il est souvent impossible d'adapter ces outils pour les rendre plus génériques ou les utiliser pour corréler leurs informations avec d'autres outils, car ils sont à source fermée. une approche adaptable et qui permet de répondre à ce problème. L'analyse d'un système hétérogène est très complexe, car de nombreux systèmes communiquent entre eux et chaque système a sa propre architecture. Il devient ainsi difficile d'avoir une vue d'ensemble du fonctionnement d'un programme. Il est donc nécessaire d'avoir une trousse d'outils qui permet d'obtenir toutes les visualisations nécessaires pour comprendre et détecter les problèmes de performance. Ces difficultés sont causées par le manque d'un modèle générique de fonctionnement d'un système hétérogène et le manque d'interfaces communes de communication avec

ces coprocesseurs.
De plus, les outils existants ne permettent pas d'avoir assez d'information pour générer une vue d'ensemble du système. Il est aussi nécessaire d'étudier les avancées actuelles dans le domaine des coprocesseurs pour pouvoir mieux les analyser et instrumenter les librairies nécessaires.Fiorini (2020)

Définition

Dans un environnement les systèmes hétérogènes informatique, une suite de machines différentes (Ces systèmes sont composés de composants variés qui peuvent différer dans leur architecture, leur système d'exploitation, leur langage de programmation, leurs protocoles de communication, leurs capacités de calcul, leurs périphériques, etc.) est interconnectée pour fournir une variété de capacités de calcul afin de répondre aux demandes de calcul de groupes de tâches vastes et diversifiées. Wang and Li (2011) Les systèmes hétérogènes sont souvent utilisés pour tirer parti des forces spécifiques de chaque composant et pour résoudre des problèmes complexes qui nécessitent des ressources spécialisées. Par exemple, dans le domaine du calcul distribué, les systèmes hétérogènes peuvent combiner des ordinateurs de différentes architectures pour effectuer des calculs intensifs plus rapidement. Un autre exemple courant de système hétérogène est l'environnement de cloud computing, où différentes machines virtuelles ou conteneurs peuvent fonctionner sur des serveurs physiques différents avec des configurations matérielles et logicielles différentes. La gestion des systèmes hétérogènes peut être complexe, car elle implique de gérer les différences entre les composants, de s'assurer de leur compatibilité, de coordonner leurs interactions et de faciliter la communication entre eux.

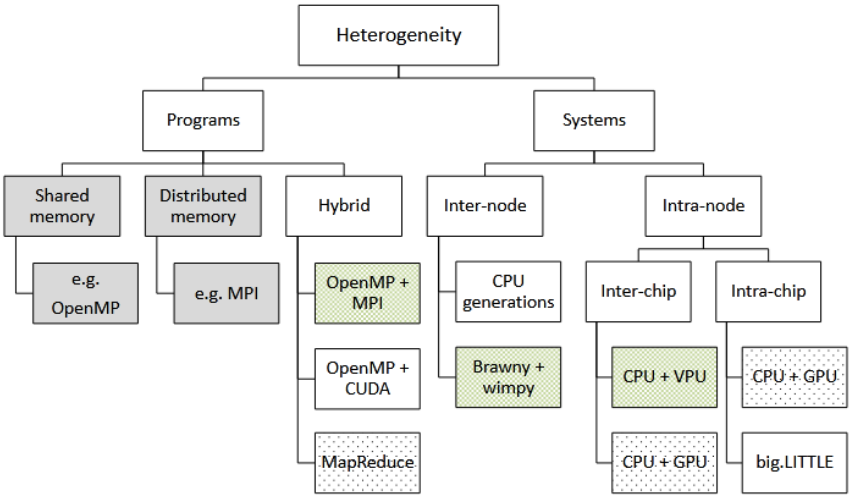


FIGURE 2.7 – Classification d'un système hétérogène

2.4.2 Calcul sur systèmes hétérogènes

le calcul sur un système hétérogène fait référence à l'utilisation de ressources informatiques variées et de performances différentes pour effectuer des calculs. Voici exemple de calculs sur des systèmes hétérogènes :

— Calcul sur des clusters informatiques : Un cluster est un ensemble de plusieurs ordinateurs connectés en réseau, travaillant ensemble pour effectuer des calculs distribués. Chaque nœud du cluster peut avoir des caractéristiques matérielles différentes, et les tâches peuvent être réparties entre les nœuds en fonction de leurs capacités et de la charge de travail.Coulouris et al. (2005)

L'objectif du calcul sur des systèmes hétérogènes est d'optimiser les performances en exploitant les ressources adaptées à chaque tâche spécifique. Cela nécessite souvent une programmation parallèle et l'utilisation de bibliothèques, de langages de programmation ou de modèles spécifiques pour exploiter les caractéristiques des différentes ressources disponibles.

2.5 Exemples des systèmes critiques a travers notre stage

2.5.1 Système *DCS*

Rappel sur l'ancien système :

A l'origine, le système de commande et contrôle des différentes pompes du terminal (pompe de chargement, Booster, drainage) était assuré par la logique câblée (relais, thyristors, transistor carte de commande de puissance, ...etc.) sujet à de nombreux inconvénients, ce qui rendait la tâche de conduite et de maintenance difficile. Nous pouvons citer en particulier les inconvénients suivants :

— difficultés lors de la mise en service.
— difficulté dans la maintenance,
— encombrement.
— durée de vie courte.
— Défauts fugitifs (sensibilité aux vibrations).
— Documentation complexe.
— pièce de rechange coûteuse.

L'ancien système de commande utilisait la régulation classique basée sur des transmetteurs à quatre fils (2 fils pour l'alimentation et 2 fils pour la mesure) pour la transmission du signal de mesure vers une carte dédiée pour la régulation (PID : Proportionnelle/ Intégrale/ Dérivée, sélecteurs,...) cette carte fournira un signal approprie pour commander la vanne réglante. Cette chaîne de régulation présente les inconvénients suivants :

— difficulté du paramétrage du régulateur : temps de réponse, P, I, D....etc.
— précision insuffisante des transmetteurs (pression et débit)
— difficulté de l'étalonnage

Devant l'évidence de ces constats, les services de la direction régionale de Skikda avaient réalisé une rénovation de l'ancien système en en le remplaçant par un système *DCS* de type numérique à commande distribuée, utilisant comme logicielle de programmation le *CS*3000 de *YOOGAWA* que nous allons présenter ci-après .

Définition du *DCS* :

Un *DCS* (Distributed Control Système) ou un *SNCC* (système numérique de Contrôle et de Commande) est un système qui permet à la fois la supervision et le contrôle en temps réel des procédés industriels. La supervision consiste à traduire l'état de chaque instrument existant

sur le site sous des formes différentes et les associer à des vues ou à des fenêtres descriptives afin d'avoir sur (HIS) un site animé en temps réel et cela pour but de pouvoir donner à l'opérateur la possibilité de surveiller le site et de faire les interventions nécessaires au moment demande (changer la consigne d'un PID,le mode Auto, l'acquittement des alarmes...etc.). Le contrôle en temps réel consiste à utiliser la régulation ou le calcul séquentiel afin d'avoir un système répondant aux exigences du cahier des charges.

Il en existe plusieurs types de système :

— $CS1000$: PFCS
— $CS3000$: SFCS, LFCS, KFCS et FFCS

Présentation du $CS3000$:

Le $CS3000$ est un système numérique de contrôle et de commande ($SNCC$) qui comprend principalement 4 éléments :

— Les stations de contrôle qui assurent la gestion des E/S issues du procédé
— le réseau redondant de communication V-NET.
— les poste de conduites opérateur (HIS) et /ou ingénieur (ENG) – (conduite et/ou dévlopment).
— le centum $CS3000$ intègre également la fourniture de solution de type :

• instrumentation (analyse, débit, pression, température).

• ENREGISTRURE, régulateurs.

• système de sécurités et datation d'événements.

• supervision.

• automate programmable.

• contrôle avancé.

• chromatographie.

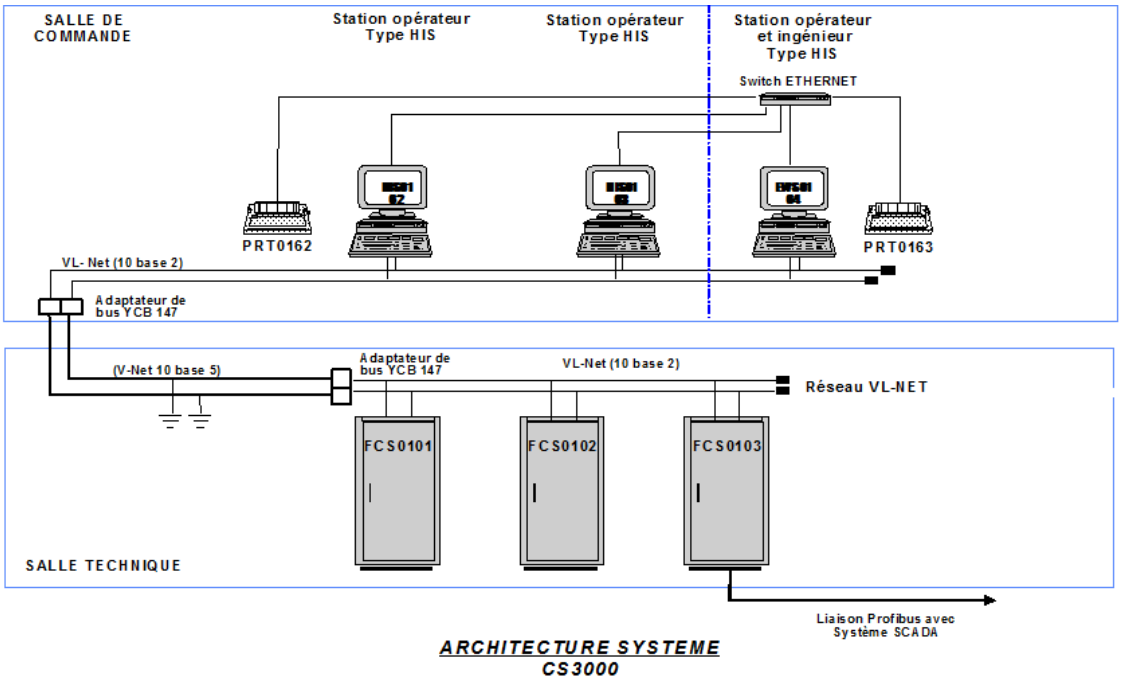


FIGURE 2.8 – Schéma d'architecture du Système

2.5.2 Système *SCADA*

Vue d'ensemble

Application architecture

Le système comprend les principaux éléments suivants :

- Réseau local TCP/IP Ethernet redondant et système *SCADA* dans *SCC* .
- *MTU* installée dans *SCC* .
- Système *SCADA* dans PC05.
- Réseau local TCP/IP Ethernet redondant et système *SCADA* dans SCS2 .
- MTU installée dans SCS2 .
- 31 RTU installées sur les Postes de Sectionnement PS25 à PS55, basés sur l'automate redondant ControlLogix .
- 3 RTU installées sur les Postes de Coupure PC05, PC06 et PC07, basés sur l'automate redondant ControlLogix.
- 2 RTU installées à Terminale Arrivée Skikda et Terminale Arrivée EL Kala, basé sur l'automate redondant ControlLogix .

Le paragraphe suivant contient une description de certaines parties.

SCC(Skikda) – Réseau local et système $SCADA$

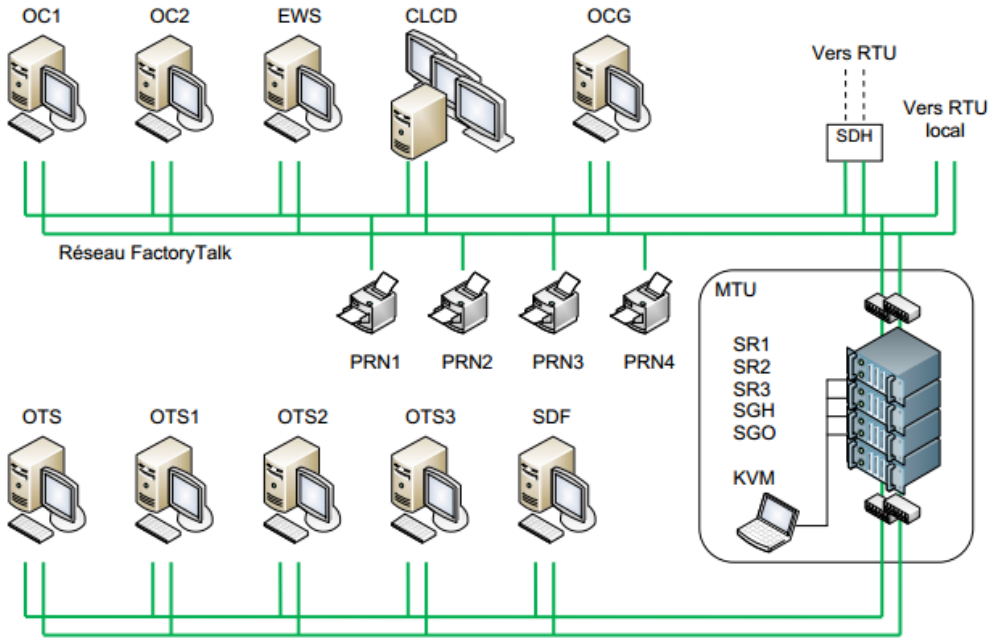


FIGURE 2.9 – Réseau OTS

Le réseau Ethernet local, de type redondant, est organisé en deux réseaux distincts non connectés. Le premier est le réseau appelé FactoryTalk. C'est le réseau du système de contrôle de supervision et d'acquisition de données basé sur la suite de logiciels Rockwell Automation FactoryTalk Services Platform. Le deuxième est le réseau appelé OTS. C'est le réseau du système de simulation d'opérations.

Le réseau FactoryTalk interconnecte les ordinateurs/serveurs/imprimantes suivants :

$OC1$: Poste de travail d'opérateur 1
$OC2$: Poste de travail d'opérateur 2
EWS : Poste de travail d'ingénierie
$CLCD$: Console d'exploitation avec 3 écrans LCD de 65"
OCG : Poste de travail GMAO (système de gestion d'état des machines)
SDF : Poste de travail système détection de fuite
$SR1$: Serveur de réseau 1 (serveur de gestion des données)
$SR2$: Serveur de réseau 2 (serveur de gestion des données)
$SR3$: Serveur de réseau 3 (serveur de gestion des données)
SGH : Serveur de gestion historique
SGO : Serveur de gestion OTS (système de simulation d'opérations)
OTS : Poste de travail enseignant OTS (système de simulation d'opérations)
$OTS1$: Poste de travail OTS 1 (système de simulation d'opérations)
$OTS2$: Poste de travail OTS 2 (système de simulation d'opérations)
$OTS3$: Poste de travail OTS 3 (système de simulation d'opérations)

$PRN1$: Imprimante pour copie d'écran
$PRN2$: Imprimante pour alarmes et événements
$PRN3$: Imprimante pour rapports
$PRN4$: Imprimante pour le poste de travail d'ingénierie

Les noms des ordinateurs/serveurs/imprimantes sont structurés comme suit :

[signes d'identification de la salle de contrôle]-[signes d'identification de l'ordinateur]

Exemple : TAS-OC1, TAS-SR1, TAS-PRN1, etc.

SCC (Skikda) – *MTU* Unité de télémétrie centrale

L'unité de télémétrie centrale consiste en une armoire placée dans la salle de contrôle/salle serveurs avec un ensemble de serveurs connectés au réseau local Ethernet, les commutateurs redondants, les systèmes de sauvegarde de données, le commutateur écran-clavier-souris pour les serveurs et l'unité de synchronisation horaire GPS.

Le MTU comprend les équipements suivants :

$SR1$: Serveur de réseau 1 (serveur de gestion des données)
$SR2$: Serveur de réseau 2 (serveur de gestion des données)
$SR3$: Serveur de réseau 3 (serveur de gestion des données)
SGH : Serveur de gestion historique
SGO : Serveur de gestion OTS (système de simulation d'opérations)
$SW1A$: Commutateur « principal » de réseau FactoryTalk
$SW1B$: Commutateur « réserve » de réseau FactoryTalk
$SW2A$: Commutateur « principal » de réseau OTS
$SW2B$: Commutateur « réserve » de réseau OTS
GPS : Unité de synchronisation horaire GPS
NBK : Unité de sauvegarde de données sur disque dur

2.6 Conclusion

Ce chapitre est structuré en trois parties : les systèmes temps réel, les systèmes critiques et les systèmes hétérogènes.Dans la première partie, nous avons présenté les systèmes temps réel en décrivant leur vocabulaire associé, tel que les catégories, la modélisation et les caractéristiques de systèmes temps réel . Dans la deuxième partie, nous avons présenté les types de systèmes critiques, y compris les systèmes logiciels et les méthodes de planification. Enfin, la dernière partie traite des problèmes et des calculs liés aux systèmes hétérogènes, avec un exemple concret de notre propre système.

Chapitre 3

Automates temporisés : Un modèle pour une vérification formelle et automatique

Sommaire

3.1 Introduction

Les automates temporisés sont des automates qui permettent de modéliser des systèmes où le temps est une variable importante. Ils ont été introduits dans les années 90 par R. Alur et D. Dill pour répondre aux besoins de vérification de systèmes réactifs, critiques ou embarqués. Contrairement aux automates classiques,les automates temporisés sont équipés d'horloges qui évoluent de manière continue avec le temps. Pour cela, les approches basées sur la vérification automatique basée sur les modèles, appelée model-checking, ont connu un développement important ces derniers temps Bérard et al. (2013); Clarke et al. (1999). Cette

méthode consiste à modéliser le comportement du système à vérifier et à formuler la propriété de correction attendue sous forme d'une propriété d'accessibilité, d'un automate ou encore d'une formule écrite en logique temporelle. Ensuite, un model-checker peut être utilisé pour vérifier si la propriété attendue est satisfaite par le modèle.
Dans le domaine des systèmes temps-réel, les contraintes liées à l'aspect temporel sont d'une grande importance. Ces contraintes sont de nature quantitative et interprètent les délais ou les durées, tels que les temps de réponse ou les timeouts. Pour modéliser le comportement des systèmes temps-réel avec des contraintes quantitatives, Alur et Dill ont proposé en 1990 le modèle des automates temporisés Alur et al. (1993); Alur and Dill (1994a). Ces horloges permettent de modéliser des contraintes temporelles sur les transitions de l'automate. Les automates temporisés sont largement utilisés en vérification formelle pour modéliser des systèmes en temps réel et pour prouver des propriétés sur ces systèmes. Alur et al. (1993); Henzinger et al. (1994); Larsen et al. (1995). Les formalismes de spécification, tels que les logiques temporelles, ont également été étendus pour pouvoir interpréter les propriétés temps-réel qui caractérisent les systèmes Alur et al. (1993); Alur and Henzinger (1994); Aceto and Laroussinie (2002). Enfin, des travaux ont été menés pour développer des outils de model-checking

3.2 Syntaxe et sémantique des automates temporisés

Dans les années 1990, les automates temporisés ont été introduits par R. Alur et D. Dill Alur and Dill (1994a). 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 sémentique

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 $s \overset{G,a,R}{\rightarrow} s'$) 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,
- $I : S \rightarrow C_{\lessdot}(H)$ associe un invariant à chaque localité,
- Act est un alphabet d'actions.

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

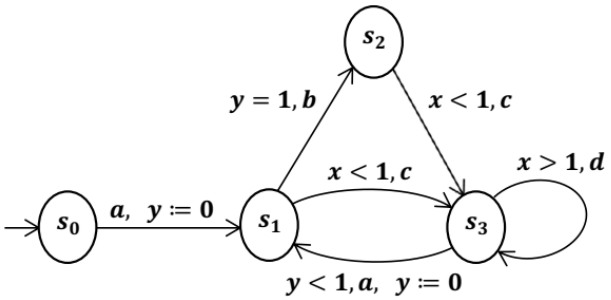


FIGURE 3.1 – Automate temporisé.

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 $q \xrightarrow{0} q'$, alors $q = q'$,

— si $q \xrightarrow{d} q'$ et $q' \xrightarrow{d'} q''$ avec $d, d' \in \mathbb{R}_{\geq 0}$, alors $q \xrightarrow{d''} q''$,

— si $q \xrightarrow{d} q'$ avec $d \in \mathbb{R}_{\geq 0}$, alors pour tout $0 \leq d' \leq d$, il existe $q'' \in Q$ tel que $q \xrightarrow{d'} q''$.

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 :

— $Q = S \times \mathbb{R}_{\geq 0}^{H}$,

— $q_0 = (s_0, v_0)$ 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 : $(s, v) \xrightarrow{a} (s', v')$ si et seulement s'il existe $s \xrightarrow{G,a,R} s' \in T$ tel que $v \models G, v' = v[R]$ et $v' \models I(s')$.

  — les transitions de temps : pour $d \in \mathbb{R}_{\geq 0}$, $(s, v) \xrightarrow{d} (s, v + d)$ 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 :
$(s_0, (0, 0)) \xrightarrow{7.35} (s_0, (7.35, 7.35)) \xrightarrow{a} (s_1, (7.35, 0)) \xrightarrow{1} (s_1, (8.35, 1)) \xrightarrow{b} (s_2, (8.35, 1)) \cdots$
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 : $(s_0, v_0, t_0) \xrightarrow{a_1} (s_1, v_1, t_1) \xrightarrow{a_2} \cdots \xrightarrow{a_n} (s_n, v_n, t_n)$ 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 $(s_i, v_i, t_i) \xrightarrow{a_{i+1}} (s_{i+1}, v_{i+1}, t_{i+1})$ 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 $(a_1, t_1)(a_2, t_2) \cdots$ Par exemple, le mot temporisé de l'exécution présentée ci-dessus est $(a, 7.35)(b, 8.35) \cdots$

Comme il existe un nombre infini d'éxecutions temporisés pour un automates temporisés donné, il nous faut une autre construction fini sur laquelle on applique la vérification des différentes propriétés.

3.3 Autre construction pour la décidabilité de la propriété d'accessibilité

Dans cette partie, nous voulons décrire une construction proposée dans Alur and Dill (1994a) 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 $v \sim v'$.

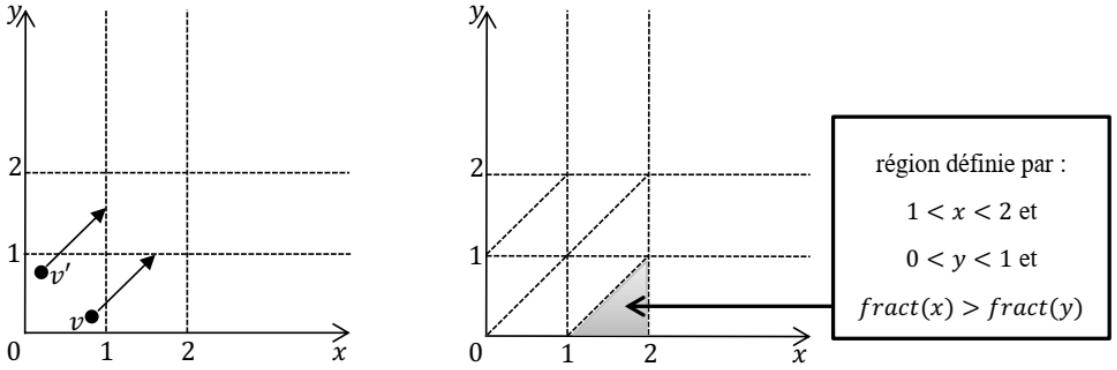
Pour toute horloge x, soit C_x la plus grande valeur avec laquelle l'horloge x a été comparée dans l'automate (supposons que cette valeur existe). On dit que deux valuations d'horloges v et v' sont équivalentes, $v \sim v'$, si et seulement si :

- 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 3.2 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 3.2(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 3.2(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 $\sim$

(b) répartition respectant les trois conditions de $\sim$

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 $(s, \alpha) \xrightarrow{a} (s', \alpha')$ s'il existe :

- une transition $s \overset{G,a,R}{\rightarrow} s'$ dans l'automate temporisé A,
- une valuation d'horloges $v \in \alpha$ et $t \geq 0$ tels que :
  - $v + t \models I(s)$,
  - $v + t \models G$;
  - $(v + t)[R] \models I(s')$ et
  - $(v + t)[R] \in \alpha'$.

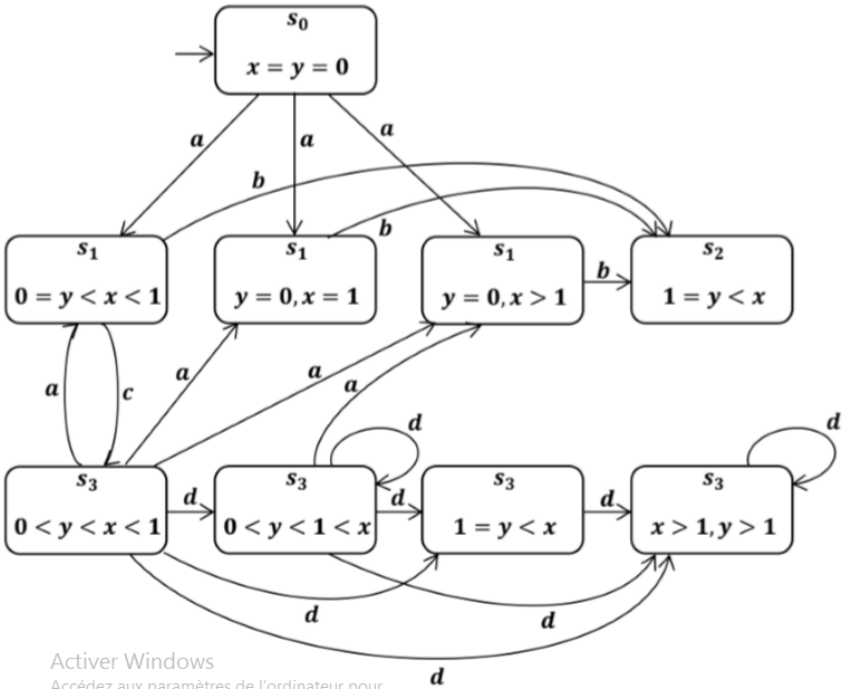


FIGURE 3.3 – Automate de régions associé à l'automate A.

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, 1994a)

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 $(s_0, x = y = 0) \xrightarrow{a} (s_1, 0 = y < x < 1) \xrightarrow{c} (s_3, 0 < y < x < 1)$. Ce chemin nous apprend qu'il existe une exécution dans l'automate temporisé A de la forme $(s_0, v_0) \xrightarrow{t_1} (s_0, v_0 + t_1) \xrightarrow{a} (s_1, v_1) \xrightarrow{t_2} (s_1, v_1 + t_2) \xrightarrow{c} (s_3, v_2)$ 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 (1994a).

— 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.

3.3.1 Definition des Régions

Nous considérons pour chaque couple d'horloges de $X, (x, y)$, une constante entière $max(y, z)$ et nous définissons l'ensemble

$$\begin{aligned} J_{y,z} = \quad & \{]-\infty; -max_{x,y}[\} \\ & \cup\{[d] | -max_{z,y} \leq d \leq max_{y,z}\} \\ & \cup\{]d; d+1[| -max_{z,y} \leq d < max_{y,z}\} \\ & \cup\{]max_{y,z}; +\infty[\} \end{aligned}$$

La région définie par l'uplet $R = ((I_x)_x \in X, (J_{x,y})_{x,y} \in X, \prec)$ où

— $\forall x \in X, I_x \in I_x$,

— $\forall (y, z) \in X_\infty, J_{y,z} \in J_{y,z}$ où $X_\infty = \{(y, z) \in X^2 \mid I_y$ ou I_z est non borné $\}$,

— $\prec$ est un préordre total sur $X_0 = \{x \in X \mid I_x$ est un intervalle de la forme $]c, c+1[\}$

est le sous-ensemble de $(\mathbb{T})^X$ tel que :

$$\left\{ \begin{array}{c} v \in \mathbb{T}^X \end{array} \quad \begin{array}{c} |\forall x \in X, v(x) \in I_x, \\ |\forall x, y \in X_0, x \prec y \leftrightarrow frac(v(x)) \leq frac(v(y)), \\ |\forall (y, z) \in X_\infty, v(y) - v(z) \in J_{y,z}\} \end{array} \right\}$$

L'ensemble fini$R_{(max_x)_x \in} X, (max_{y,z})_{y,z} \in X$ de tous ces sous-ensembles de$(\mathbb{T})^X$forme une partition de$(\mathbb{T})^X$. Par une preuve similaire a celle du lemme , il est facile de vérifier que cet ensemble vérifie aussi la condition ,*i.e.* que le lemme suivant est vérifié :

Lemme

L'ensemble $R_{(max_x)_{x \in X}, (max_{y,z})_{y,z \in X}}$ est un ensemble de régions.

Exemple

Supposons que nous ayons uniquement deux horloges, x et y, et que les constantes maximales soient $max_x = 3, max_y = 2, max_{x,y} = 1$ et $max_{y,x} = 0$. Alors l'ensemble des régions associées à ces constantes est décrit sur le (figure3.4) à droite. La région grise est définie par $I_x =]3; +\infty[, I_y =]2; +\infty[$ et $-1 < y - x < 0 (i \cdot e \cdot J_{y,x}$ est $]-1; 0[)$.

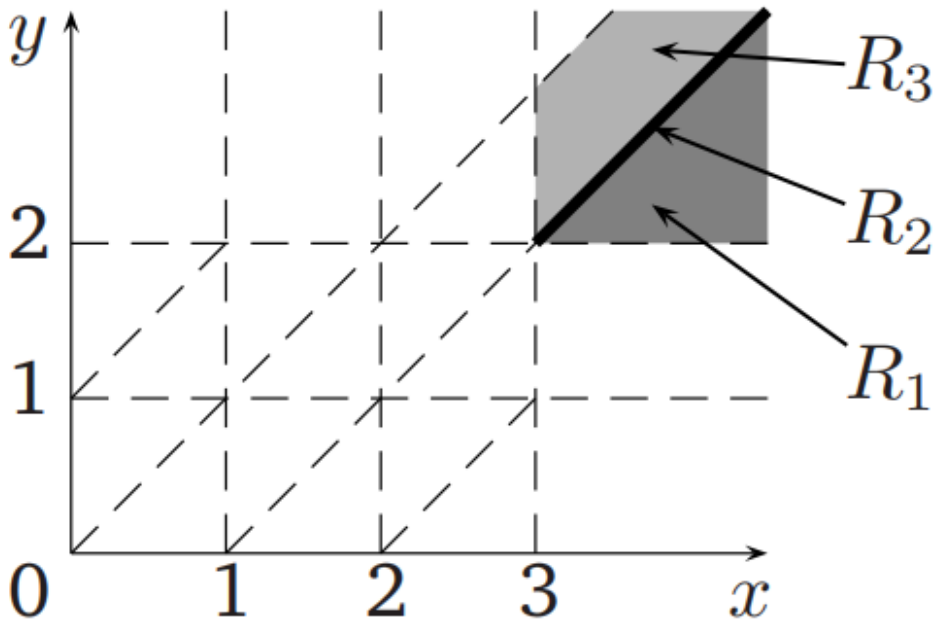


FIGURE 3.4 – l'ensemble des régions constantes

Notons que les ensembles de régions dans le cas des contraintes non diagonales peuvent apparaître comme des cas particuliers des ensembles de régions que nous venons de décrire en étendant la définition précédente à des constantes $max_{x,y}$ qui valent $-\infty$. Nous appellerons ces ensembles des ensembles de régions non diagonaux.Bouyer (2002)

3.3.2 Le graphe de régions

Soit $A = (Q, X, \Sigma, Q_0, F, R, T)$ un automate temporisé classique avec des constantes entières et des contraintes non diagonales. Pour toute horloge $x \in X$, nous notons max_x la plus grande constante c telle qu'une contrainte d'horloges $x \sim c$ apparaisse dans A L'équivalence des régions est définie sur $\mathbb{T}^X$ par :

$$v \equiv v' \Leftrightarrow \left\{ \begin{array}{c} Ent(v(x)) = Ent(v'(x)) ou (v(x) > max_x et v'(x) > max_x) \\ si (v(x) \leq max_x et v(y) \leq max_y) \\ alors [(frac(v(x)) < frac(v(y)) \Leftrightarrow frac(v'(x)) < frac(v'(y)))] \end{array} \right.$$

où pour tout réel $\alpha, Ent(\alpha)$ représente la partie entière de α alors que $frac(\alpha)$ représente la partie fractionnaire de α.
La relation $\equiv$ est une relation d'équivalence. Elle vérifie la propriété suivante :

$$v \equiv v' \Leftrightarrow \left\{ \begin{array}{c} (i) pour toute contrainte g de A, v \models g \Leftrightarrow v' \models g \\ (ii) \forall t \in \mathbb{T}, \exists t' \in \mathbb{T} tel que v + t \equiv v' + t' \end{array} \right\}$$

La première propriété ci-dessus indique que l'équivalence$\equiv$ est compatible avec les contraintes de l'automate$A$ alors que la deuxième propriété indique que l'équivalence$\equiv$ est compatible

avec l'écoulement du temps. Il est facile de voir que l'équivalence des régions est d'index fini. Une classe de $(\mathbb{T}^X)/\equiv$ est appelée une région

Exemple

Considérons un automate temporisé pour lequel $max_x = 3$ et $max_y = 2$. L'ensemble des régions associé à cet automate peut être décrit par le schéma de la (figure3.5)de droite. La région grisée est définie par la contrainte
$1 < x < 2\Lambda 1 < y < 2\Lambda x > y$
Bouyer (2002)

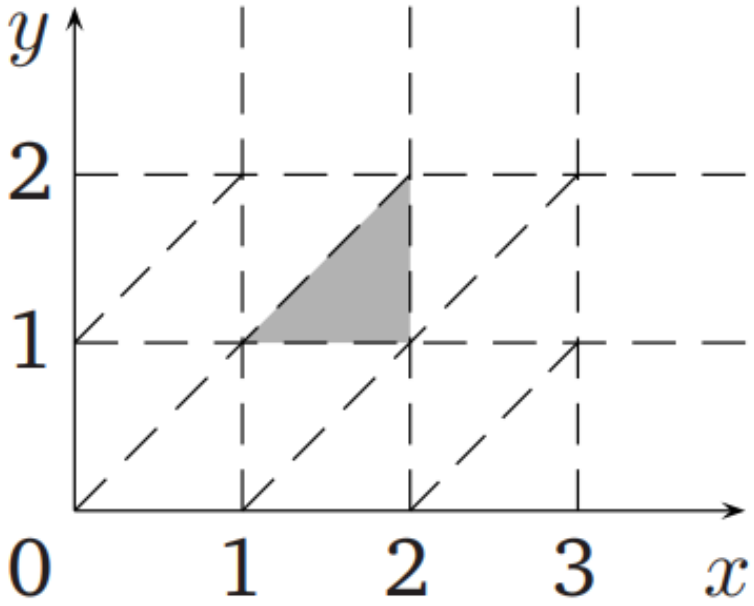


FIGURE 3.5 – l'ensemble des associé régions

3.3.3 L'automate de régions

Nous avons vu que l'équivalence des régions est compatible avec l'écoulement du temps. Il est donc possible de définir une fonction successeur sur l'ensemble des régions. Si R est une région, nous notons $Succ(R)$ l'ensemble des successeurs de R pour l'écoulement du temps, c'est-à-dire l'ensemble de régions $Succ(R)$ vérifiant la propriété suivante :
$R' \in Succ(R) \Leftrightarrow \exists v \in R. \exists t \in T$ tels que $v + t \in R'$

Exemple

Le premier successeur de la région en gris foncé est la région dessinée par une ligne épaisse noire. L'ensemble des autres successeurs de cette même région est dessiné en gris clair.(figure3.6)

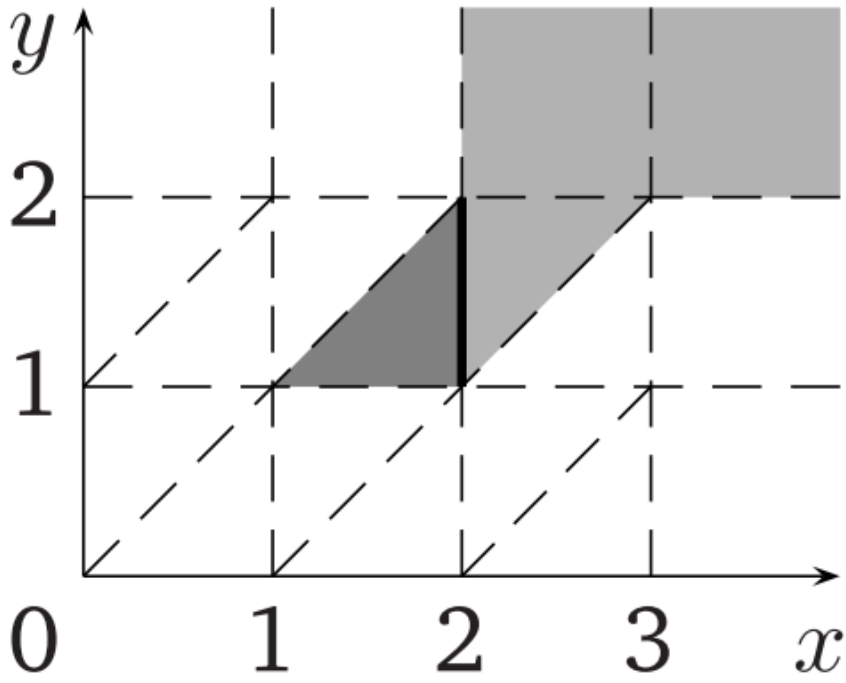


FIGURE 3.6 – l'ensemble de région succ

Nous sommes maintenant en mesure de définir un automate fini $B = (Q', \Sigma, q'_0, F', R', T')$ de la manière suivante :

— $Q' = Q \times \mathbb{T}^X \equiv, q'_0 = (q_0, 0), F' = F \times \mathbb{T}^X / \equiv, R' = R \times \mathbb{T}^x / \equiv$,

— $T' = \left\{ \begin{array}{c} (q, R) \xrightarrow{a} (q', R') \mid \\ (\exists q \overset{g,a,C}{\rightarrow} q' \in Tet \exists R'' \in Succ(R) \\ telque R \subseteq getR' = R''[C \leftarrow 0] \end{array} \right\}$

Cet automate est appelé l'automate des régions associé à A .

Exemple

Nous traitons un exemple de construction de l'automate des régions. Considérons donc l'automate A dessiné sur la(figure3.7) .

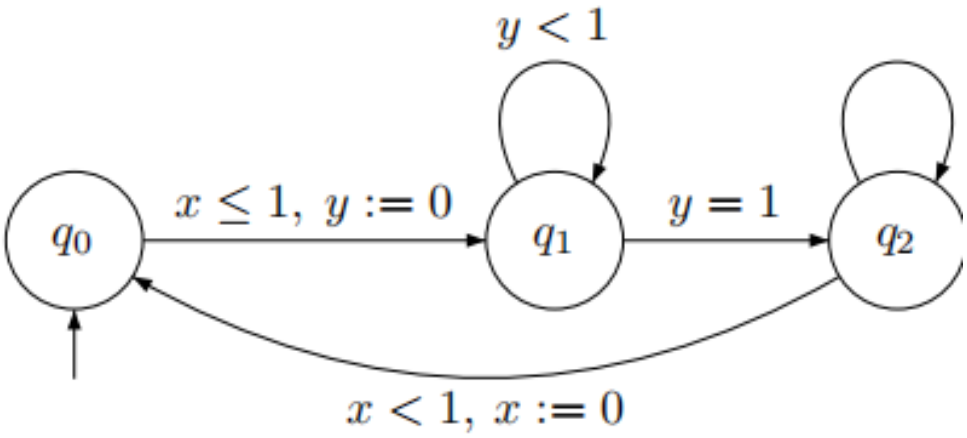


FIGURE 3.7 – Automate temporisé A

L'automate des régions que nous obtenons à partir de A en prenant comme constantes $maximales1$ pour les deux horloges x et y est dessiné sur la (figure 3.3).
L'automate des régions vérifie la propriété suivante :

Propriété

Le langage accepté par$B$ correspond au Untime du langage accepté par A . Nous avons alors le théorème plus précis suivant :

Théorème

Tester le vide d'un langage accepté par un automate temporisé est un problème PSPACE-complet.Bouyer (2002)

3.4 Vérification automatique basée modèles (model cheking)

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 U. Alors on peut dire qu'une formule $\varphi \cup_{\sim c} \psi$ 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
underlinein 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(problème $\Rightarrow$ $F_{\leq 3}$ 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
  $(s_1, v_1) \xrightarrow{a} (s'_1, v'_1)$ avec $a \in Act$,il existe une transition $(s'_2, v_2) \xrightarrow{a} (s'_2, v_2)$ telle que$(s'_1, s'_1) \approx (s'_2, s'_2)$,et inversemant
- pour toute transition
  $(s_1, v_1) \xrightarrow{t} (s'_1, v'_1)$ avec $t \in R_{\geq 0}$, il existe une transition $(s_2, v_2) \xrightarrow{t} (s'_2, v'_2)$ 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).

3.4.6 Model checking

Le Model-checking est une technique de vérification automatique des systèmes dynamiques. Il s'agit de construire un modèle du système à vérifier avec un formalisme donnée,généralement représenté par un automate à états fini. Ce modèle est vérifié s'il satisfait un ensemble de propriétés (une spécification) souvent formulées en logique temporelle . En autres termes et de point de vue logique, le système est décrit par un modèle sémantique(structure de Kripke), et les propriétés sont décrites par des formules logiques .Tibermacine (2009) La difficulte dans l'analyse des systèmes temporisè provient du fait que les systemes de transitions associès aux modeles considèrés ont un nombre infini d' états (non dénombrable lorsque le domaine de temps est $(R \geq 0)$. Il n'est donc pas possible d'appliquer directement les techniques standard du model checking. En fait, le premier resultat de dcidabilitée obtenu pour les automates temporisés est dua Alur et Dill . Il porte sur les langages ‘ (de mots temporises) acceptés par des automates temporisés (de Buchi) et ¨ enonce que le vide est décidable dans cette classe de langages. Une grande partie des résultats positifs, incluant la decidabilité du modél checking de TCTL utilise la technique introduite dans cet mémoire, qui consiste en la construction d'un automate fini bisimilaire (avec abstraction du temps) a l'automate temporisé original. Nous rappelons d'abord rapidement les algorithmes utilisés dans le cadre non temporisée, puis nous éxpliquons la construction d'un automate fini associéa un automate temporise et enfin, nous montrons comment ces techniques ont et éadaptées aux modéles et aux logiques temporisées.Midonnet (2007) Les modèles de représentation des systèmes peuvent être des automates, et leurs extensions, mais aussi l'algèbre de processus, les réseaux de Petri, etc., et les propriétés peuvent être modélisées soit dans les mêmes formalismes, soit sous forme

de formules dans certains (par exemple, temporel) logique. Par exemple, on peut écrire des formules commeBouyer (2009)

3.4.7 Le travail du model-checking

Le travail se situe dans le model-checking. Nous allons en décrire le principe de manière plus précise. Supposons qu'il nous soit donné d'une part un système (sous la forme d'un programme, ou d'un cahier des charges par exemple) et d'autre part une propriété de ce système. L'étape préliminaire du model-checking consiste d'une part à modéliser le système, c'est-à-dire construire un objet dans le cadre formel bien défini qui « reproduit » aussi fidèlement que possible le comportement du système réel, et d'autre part à modéliser la propriété à vérifier dans un langage de spécification adapté. Le model-checking consiste alors à vérifier que le modèle construit pour le système vérifie la propriété exprimée dans le langage précité (d'où le terme « model-checking », ce sont les modèles que l'on vérifie). Pour pouvoir réaliser cela, il est bien entendu nécessaire de disposer d'algorithmes de model-checking. La (figure3.8)schématise le principe général du model-checking.
De ce principe général se dégagent assez clairement trois grands axes de recherche complémentaires permettant d'améliorer les techniques du model-checking : le développement de modèles pour représenter les systèmes que l'on étudie ainsi que les propriétés que l'on cherche à vérifier Bouyer (2002)

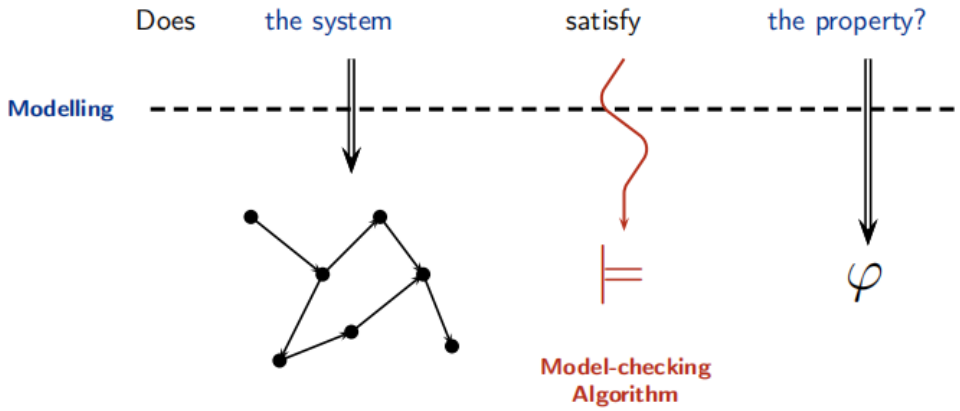


FIGURE 3.8 – schéma général du model ckecking

3.4.8 Vérification de modèles (Model-checking)

La vérification de modèles correspond l'ensemble des techniques qui permettent de d´eduire algorithmiquement des propriétés sur le comportement d'un systéme (logiciel, algorithme ou protocole)à partir de l'exploration d'un modèle le représentant (automate fini, automate temporisé, réseau de Petri, algèbre de processus,$\cdots$). La première étape consiste à établir un modèle du logiciel, algorithme ou protocole que l'on veut vérifier. Le formalisme de ce modèle

est un système de transition étiquetées. Il convient ensuite de traduire les propriétés que l'on veut vérifier en formules logiques ($LTL, CTL, CTL\divideontimes, TCTL, FOL, \cdots$). La dernière étape, c'est à dire la vérification proprement dite,est totalement prise en charge par le logiciel de vérification. Celui-ci va combiner le modèleet la formule logique et calculer l'ensemble des états accessibles en avant (post$\divideontimes$ ou 'forwardanalysis') ou en arrière (pre $\divideontimes$ ou 'backward analysis'). S'il existe un chemin entre l'état initial et l'ensemble des états qui vérifient la formule, alors la propriété est vérifiée. Il existe un certain nombre de formalismes permettant de modèliser et de vérifier des propriétés par ce biais :

- Les réseaux de Petri dont l'accessibilitéest décidable et dont le model-checking des formules LTL est décidable . Un certain nombre d'outils utilisent cette théorie comme base, on peut citer : DESIGN-CPN , PAPETRI et PEP.
- Les automates 'a pile dont l'ensemble des états accessibles est reconnaissable et calculable et dont le model-checking des formules CTL*est décidable.
- Les automates temporisés dont le vide est décidable ainsi que le model-checking de TCTL et d'un certain nombre d'autres logiques temporelles temporisées. Plusieurs outils utilisent cette théorie Kronos , Uppaal et CMC .
- Les automates hybrides sont un cas 'a part. L'accessibilité est indécidable mais certains semi-algorithmes efficaces dans une grande partie des cas réels existe

L'outil le plus connu dans ce domaine étant HyTech .
La vérification de modèles pose deux problèmes cruciaux.Le premier apparaˆıt lors de la conception du modèle. En effet, il est crucial de conserver une equivalence entre le modèle que l'on va utiliser et la réalité (au moins en ce qui concerne les propriétés que l'on vérifie).De cette équivalence dépend la validité de la preuve que l'on apporte. L'autre problèmevient de l'explosion du nombre d'´etats du modèle que l'on vérifie. Ce problème se rencontre quasiment systématiquement sur les algorithmes de calcul de pre$\divideontimes$ ou de post$\divideontimes$. C'est bien souvent ce problème qui empˆeche la vérification de systémes trop complexes. Il est cependant possible de réduire le nombre des états par le choix d'abstractions qui préservent les propriét´esvoulues, tout en réduisant le nombre d'états 'a parcourir. C'est d'ailleurs l'étude d'une de ces abstractions qui motive cette thèse.Fleury (2002) Il est aussi possible de vérifier des propriétés dites "temps réel", incluant des indications temporelles. Ces propriétés peuvent être vérifiées en utilisant la technique des observateurs : les propriétés temporisées sont traduites en propriétés LT L et la vérification est conduite sur le produit du modèle initial et de l'observateur associé à la propriété.Bourdil (2015)

3.4.9 Model-checking sur TCTL

La logique la plus répandue pour la vérification d'automates temporisés est la logique TCTL (Timed Computation Tree Logic) qui est une extension de la logique CTL autorisant l'expression de contraintes temporelles dans les propriétés. Les formules TCTL sont définies de manière inductive par la grammaire suivante définie dans :
$f ::= p \mid x \in I \mid \neg f \mid f1 \vee f2 \mid \exists\Diamond_I f \mid \forall\Diamond_I f \mid \exists\Box_I f \mid \forall\Box_I f$
avec $\rho \in P$ un prédicat de base, $\chi \in X$ une horloge et I un intervalle de temps. Intuitivement l'opérateur $\exists\Diamond_I f$ signifie qu'il existe une exécution menant à une configuration satisfaisant la propriété f au temps $t \in I$. $\forall\Diamond_I f$ signifie que chaque exécution possède une configuration où la propriété f est valide au temps $t \in I$.$\forall\Box_I f$ signifie que toutes les configurations sur

toutes les exécutions satisfont la propriété f.$\exists\Box_I f$ signifie qu'il existe une exécution telle que tous les configurations de l'exécution satisfont la propriété f .
Un automate temporisé ayant la particularité de posséder un nombre infini de comportements liés à la valuation de ses horloges, on se trouve confronté au problème de l'explosion combinatoire. Afin de résoudre ce problème, le model-checking symbolique représente des ensembles de configuration à l'aide de structures de données efficaces comme les diagrammes de décision binaire (encore appelés BDD pour Binary Decision Diagram), ou les matrices de bornes (DBM pour Difference Bound Matric)Zhao (2014)

3.5 Conclustion

Nous introduisons dans ce chapitre les automates temporisés.au modèle original d'Alur et Dill . Nous commencerons par définir quelques notions de bases relatives aux automates temporisés (syntexe ,Préliminaires, sémentique des automates temporisés).Nous allons mettre l'accent sur l'utilisation de ce modèle dans la vérification formelle qui peut être appliquée à différents types de vérifications telles que l'accessibilité, Vérification automatique basée modèles (model cheking).

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 (2017)), 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 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.
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. (1995)Bouyer et al. (2005) Rodriguez-Navas and Proenza (2012)). 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 locales 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 and Cheng (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 et al. (2014)), 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 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 $\neg$ 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 $\upsilon : Z \to \mathbb{R}_{\geq 0}$.
Une valuation υ satisfait $\varphi \in \mathcal{C}_Z$,noté $\upsilon \models \varphi$, si φ est évaluée à true on respectant les valeurs données par υ. Pour $R \subseteq Z, \upsilon[R]$ dénote la valuation d'horloges définie par $\upsilon[R](x) = 0$ si $x \in R$ et $\upsilon[R](x) = \upsilon(x)$ sinon.υ_0 dénote la valuation d'horloges initiale qui mappe toute horloge de Z à 0.
Pour un tuple non-vide $t = (\mathrm{t}_1,\mathrm{t}_2,...\mathrm{t}_n), (\mathrm{t}, e)$ 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 $s(s_1, s_2, \cdots, s_n)$et $t = (t_1, t_2, \cdots, t_n)$ comme suit :
concat$(s, t) = (\cdots((s, t_1)t_2)\cdots, t_n(s_1, \cdots, s_n, t_1 \cdots, t_n)$
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 and Dill (1994a)) 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)est l'ensemble fini des transitions, $I : S \to \mathcal{C}_Z$ 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 $\tau : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ avec $\tau(0) = 0$ et $\tau(t)$ 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 and Dill (1994a)**?**), 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).

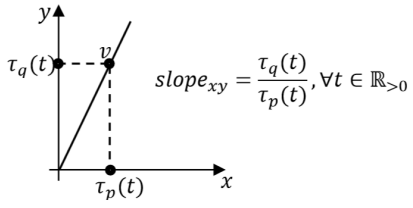


FIGURE 4.1 – Evolution de deux horloges avec différentes fréquences

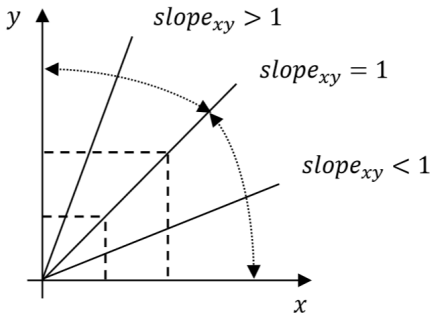


FIGURE 4.2 – Différentes possibilités de l'évolution de deux horloges

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 (Verissimo (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.

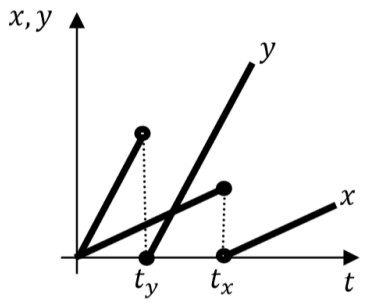


FIGURE 4.3 – Réinitialisation de deux horloges avec différentes fonctions linéaires de vitesses

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 and Dill (1994a)). 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 r-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 4.1 (une première répartition)

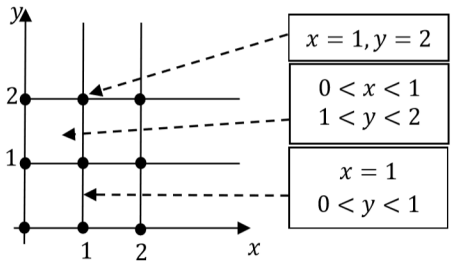


FIGURE 4.4 – Une première répartition

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.

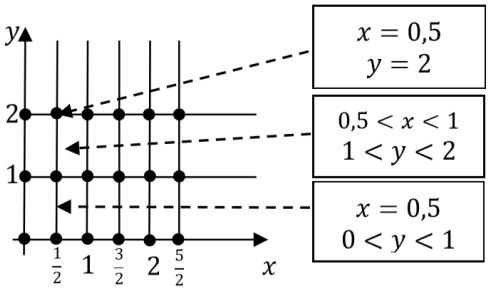


FIGURE 4.5 – Une deuxième répartition

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 $\varphi_3 = y \geq 1$.

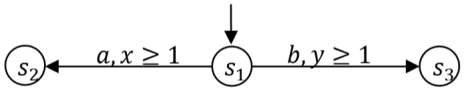


FIGURE 4.6 – r-TA B

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.

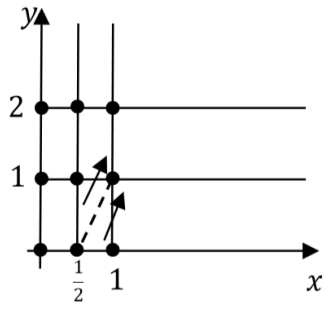


FIGURE 4.7 – passage du temps pour deux valuations d'horloges

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)

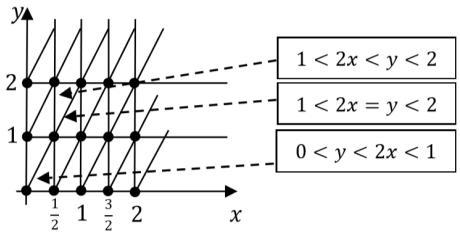


FIGURE 4.8 – Une troisième répartition

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 $[0 < 2x < y < 1]$).

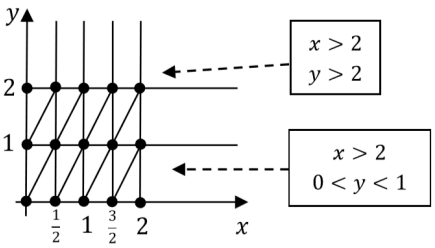


FIGURE 4.9 – Répartition finale

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 4.9 , 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(figure4.4) 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(figure 4.4) , 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 changement 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 :
$\left\{x = c \mid c = 0, \frac{1}{slope_{xy}}, 2\frac{1}{slope_{xy}}, ..., 1, 1 + \frac{1}{slope_{xy}}, 1 + 2\frac{1}{slope_{xy}}, ..., C_x forally \in H\right\}$
$cup \left\{\bigwedge_{y \in H} (c - \frac{1}{slope_{xy}} < x < c) \mid c =\right.$
slope$_{xy}$, $2\frac{1}{slope_{xy}}, \cdots, 1, 1 + \frac{1}{slope_{xy}}$,
$1 + 2\frac{1}{slope_{xy}}, \cdots, C_x$ for each $ycup\{x > C_x\}$. (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 $(x > c_x)$
— **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 4.9

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 α sera 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. Si α 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 4.9, 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 d'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 Figure4.9, 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 du r-TA

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 αsi et seulement s'il existe une transition étiquetée par α partir d'une localité s avec une valuation d'horloges $v \in \alpha$ vers une localité s' avec une valuation d'horloges $v' \in \alpha'$ 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 v de α Formellement $\forall v \in \alpha, \forall x \in Z$:

— si $x \in R$ alors $v(s) = 0$ dans $\alpha[R]$.
— soit $v(x)$ dans $\alpha[R]$ est le $m\hat{e}$me 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 $< s_o|[v_0] > o\grave{u} v_0(x) = 0$ pour tout $x \in Z$.
— Une transition de $R(B)$, de la configuration $< s|\alpha >$ vers $< s'|\alpha' >$, est étiquetée 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 υ valuation d'horloges sur Z On représente chaque région d'horloges α un tuple $< interv, permut, pred >$ *où* interv 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 $\upsilon(x) \in interv_x$ 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 :

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

$permul = \{x_{i1}, \cdots, x_{ik}\}$ est une permutation de $Z_{open} = \{x_1, \cdots, x_k\}$ tel que pour toute valuation d'horloges υ la région α les horloges de Z_open sont arrangées selon : $i_h < i_j$ implique $slop_{x_{ih}x_{ij}(x-x_h-c)} \leq (x_{ij} - d)$ *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 υ de la région α : $slop_{x_{ih}x_{ij-1}}(x_ij - c) = x(i_{j-1} - d$

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 $\upsilon(x)$ 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 (termainaison)

Pour un 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 4.1 .

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 regions du r-TA (B) est possible, le test du vide et model checking sont prouvés décidable.

4.5 Utilisation de structures de données efficaces pour la recherche et la gestion des successeurs

— La conception de **la pile de régions**, également appelée pile-ordre pile-contrainte temporelle atomique, repose sur la notion de cas. Chaque cas représente un ensemble de données et d'opérations associées qui sont regroupées ensemble.

— La pile de régions est composée de plusieurs cas, et chaque cas contient différentes régions. La région "***alpha*** et ***beta***" est l'une des régions présentes dans chaque cas, mais il peut y avoir d'autres régions spécifiques selon les besoins du système.

— La pile est divisée en deux parties distinctes : ***la pile-ordre*** et ***la pile-contrainte temporelle atomique***. La pile-ordre gère les cas selon leur ordre d'arrivée. Lorsqu'un nouveau cas est ajouté à la pile, il est placé en haut de la pile et devient le cas actif. Les opérations effectuées sur la pile-ordre suivent l'ordre séquentiel des cas.

— D'autre part, la pile-contrainte temporelle atomique gère les cas en fonction de leurs contraintes temporelles. Les cas ayant des contraintes temporelles similaires sont regroupés ensemble. Cela permet d'optimiser les opérations sur ces cas en termes d'allocation et de libération de ressources.

— En utilisant la conception de la pile de régions avec la pile-ordre et la pile-contrainte temporelle atomique, il devient possible de gérer efficacement un ensemble de cas et de leurs régions associées. Cette approche facilite la gestion de la mémoire, l'allocation des ressources et l'exécution des opérations de manière cohérente et optimisée.

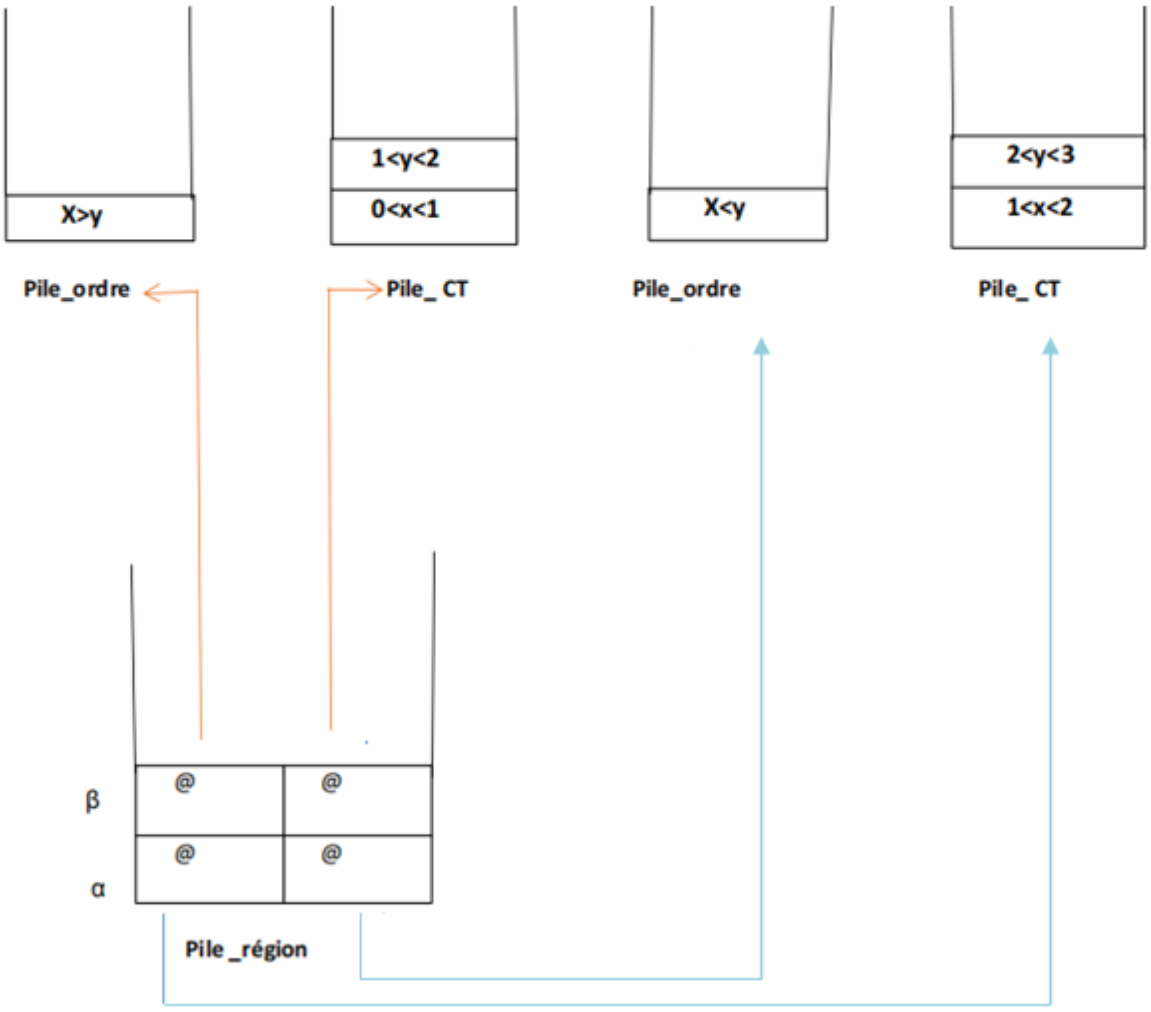


FIGURE 4.10 – Structure illustrative pour la Pile

4.6 Algorithme de construction d'un automate de régions à partir d'un r-TA

Algorithm 1 Algorithme des régions successeurs

Variables :

α, β : enregistrement de deux champs.

pile-région :pile d'éléments de type enregistrement. \\ *chaque élément possède deux champs pour pointer les deux piles pile-ordre et pile-CT.*

pile-ordre , pile-CT : pile de contraintes temporelles.

CTA : contrainte temporelle atomique.

x :horloge

Début du programme principal

Tantque (pile-vide(pile-région)=faux)
$\alpha \leftarrow$ dépiler (pile-région)
Si '=' $\in$ operators ($\alpha\cdot$ pile-CT)
α appartient au 2éme cas}
Sinon
Si operators ($\alpha\cdot$ pile-CT)={ >}
α appartient au 1er cas
Sinon
α appartient au 3éme cas
Finsi
Finsi

Si $\alpha \in$ {régions du 1er cas}alors
$\beta = \alpha$
Finsi

Si $\alpha \in$ { région du 2éme cas }
$\beta\cdot$ pile-ordre = $\alpha\cdot$ pile-ordre
Tantque (pile-vide)($\alpha \cdot$ pile-CT)= false)faire
CTA $\leftarrow$ depiler($\alpha\cdot$ pile-CT)
Si operators (CTA)$\neq$ {=} alors
Empiler (CTA , $\beta \cdot$ pile-CT)
Sinon
Si val (CTA)= val_{max} (clock Id(CTA)
CTA $\leftarrow$ clock Id(CTA) >val_{max} (clock Id(CTA)
Empiler (CTA , $\beta\cdot$ pile -CT)
Sinon
CTA $\leftarrow$ val(CTA) < (clock Id(CTA)< $Val(CTA) + \frac{1}{slope_{max(clockId)}}$
Empiler (CTA , $\beta \cdot$ pile-CT)
Finsi
Finsi
FinTantque
Finsi

Si $\alpha \in$ { région du 3éme cas }
$\beta \cdot$ pile-ordre $= \alpha \cdot$ pile-ordre
$x \leftarrow$ clockIdMax(depiler($\alpha \cdot$ pile-ordre))
Tantque (pile - vide ($\alpha \cdot$ pile-CT)= false)faire
CTA $\leftarrow$ depiler($\alpha \cdot$ pile-CT)
Si clock Id(CTA) $\neq x$ alors
Empiler (CTA , $\beta \cdot$ pile-CT)
Sinon
CTA $\leftarrow clockId(CTA) = val_{sup}(CTA)$
Empiler (CTA , $\beta \cdot$ pile-CT)
Finsi
FinTantque
Finsi
FinTantque
Fin du programme principal

4.6.1 Les fonctions utilisées dans l'algorithme :

Operators :renvoie l'ensemble des opérateurs présents dans alpha et pileCT.

ClockId :renvoie l'identifiant d'horloge de CTA.

ClockIdMax : renvoie l'identifiant d'horloge maximum de CTA.

pile-vide :Fonction pour vérifier si une pile est vide ou non.

val : renvoie la valeur associée au CTA.

valmax :returns the maximum value associated with the clock identifier.

valsup :returns the successor value of the clock identifier.

4.7 Conclusion

Dans ce chapitre, une approche pour l'analyse des systèmes temporisés avec des horloges évoluant à des fréquences relatives a été présentée. La décidabilité a été prouvée en utilisant une abstraction appelée région, qui redéfinit les concepts nécessaires.
Pour poursuivre cette contribution, il reste à étudier la puissance d'expressivité des automates temporisés avec des vitesses relatives du temps et à examiner l'impact des autres relations entre les fréquences des horloges. Cela pourrait être réalisé en explorant différentes formes et valeurs du paramètre "slope". De plus, il serait intéressant d'explorer les capacités de ce modèle pour décrire la sémantique des spécifications des systèmes ambiants.
Enfin, il est possible qu'une extension supplémentaire des résultats soit nécessaire pour appliquer cette approche au model-checking. Ces études permettront de mieux comprendre les possibilités et les limites de l'analyse des systèmes temporisés avec des horloges à fréquences relatives.

Chapitre 5

Conception et implémentation

Sommaire

5.1 Introduction

Dans ce chapitre, nous présentons les résultats de notre travail après avoir effectué une étude approfondie. Nous commençons par présenter les diagrammes de classes utilisés pour la modélisation du r-TA et de l'automate de régions.
Dans la continuité, nous abordons maintenant la phase d'implémentation où nous présentons l'environnement matériel ainsi que les outils de développement utilisés
Pour conclure cette partie, nous vous présenterons quelques captures d'écran qui démontreront les fonctionnalités de notre application.

5.2 Diagrammes de classes

Le diagramme de classes est un type de diagramme UML (Unified Modeling Language) qui permet de représenter les classes, les interfaces, les associations et les relations entre ces éléments. Avec notre logiciel de création de diagrammes≪**UML**≫, il est facile de créer des diagrammes de classes. Ce processus n'est pas aussi difficile qu'on pourrait le penser.

Le terme≪ **interactions** ≫ fait référence aux diverses relations et connexions qui peuvent exister entre les éléments dans les diagrammes de classes et d'objets.Voici quelques-unes des interactions les plus communes :

— *Héritage* : également connu sous le nom de généralisation, il s'agit du processus par lequel un enfant ou une sous-classe adopte la fonctionnalité d'un parent ou d'une super-classe. On le symbolise par une ligne de connexion droite avec une pointe de flèche fermée orientée vers la super-classe.

Ce guide vous montrera comment comprendre, planifier et créer diagrammes de classes à travers deux classes principales.

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

Le diagramme de classes représente une modélisation d'un système de transitions d'automates temporisés (r-TA). Il est composé de plusieurs classes :

r-TA

une classe qui représente l'automate temporel, qui contient les transitions entre les états.

Transition

une classe qui représente une transition entre deux états, qui peut être déclenchée par une garde et qui peut entraîner une action.

State

une classe qui représente un état de l'automate temporel.

Clock

une classe qui représente une horloge, qui est utilisée pour mesurer le temps dans le système.

CTAR

une classe qui représente une contrainte temporelle atomique restrainte, qui est une condition de temps sur un événement.

CTAG

une classe qui représente une contrainte temporelle atomique globale, qui est une condition de temps sur l'ensemble des événements du système.

Garde

une classe qui représente une condition qui doit être remplie pour qu'une transition puisse être déclenchée.

Inv

une classe qui représente une invariance, qui est une condition qui doit être respectée à tout moment dans le système.

Initial

une classe qui représente l'état initial de l'automate temporel.

Qq

une classe qui représente un ensemble de contraintes temporelles atomiques globales.

Action

une classe qui représente une action qui doit être exécutée lorsqu'une transition est déclenchée.

Reà0

une classe qui représente une initialisation des horloges.

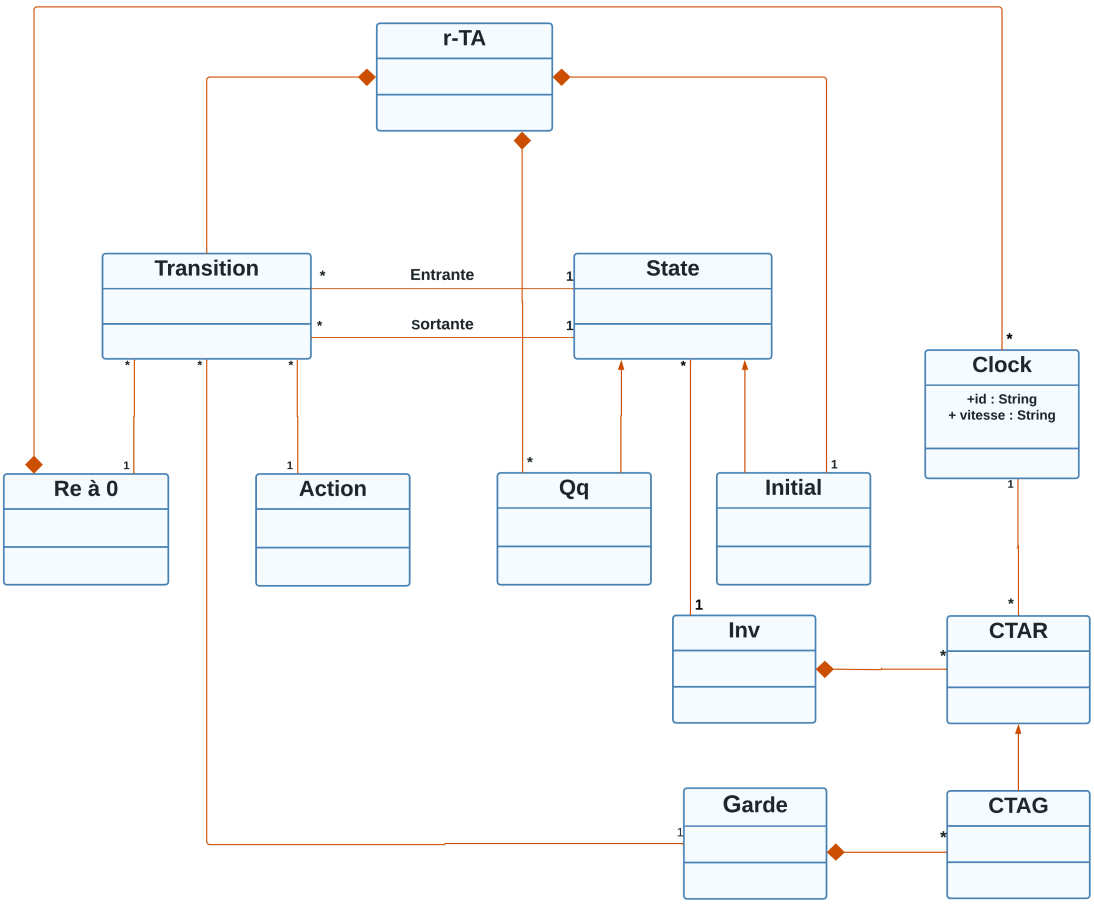


FIGURE 5.1 – Diagramme de classes du r-TA.

5.2.2 Diagramme de classes pour l'automate de régions

Ce diagramme de classes représente les différentes classes utilisées pour modéliser un système à événements discrets et à temps continu :

La classe RG

représente un graphe de régions, qui est une structure de données utilisée pour décrire les différents modes de fonctionnement d'un système.

La classe Configuration

représente une configuration du système, qui est un ensemble d'états et de régions actives à un moment donné.

La classe Transition

représente une transition entre deux configurations, qui peut être déclenchée par une action.

La classe Action

représente une action qui peut déclencher une transition.

La classe State

représente un état du système.

La classe Région

représente une région du graphe de régions, qui peut contenir plusieurs états.

La classe CTAG

est utilisé pour modéliser les transitions entre les régions.

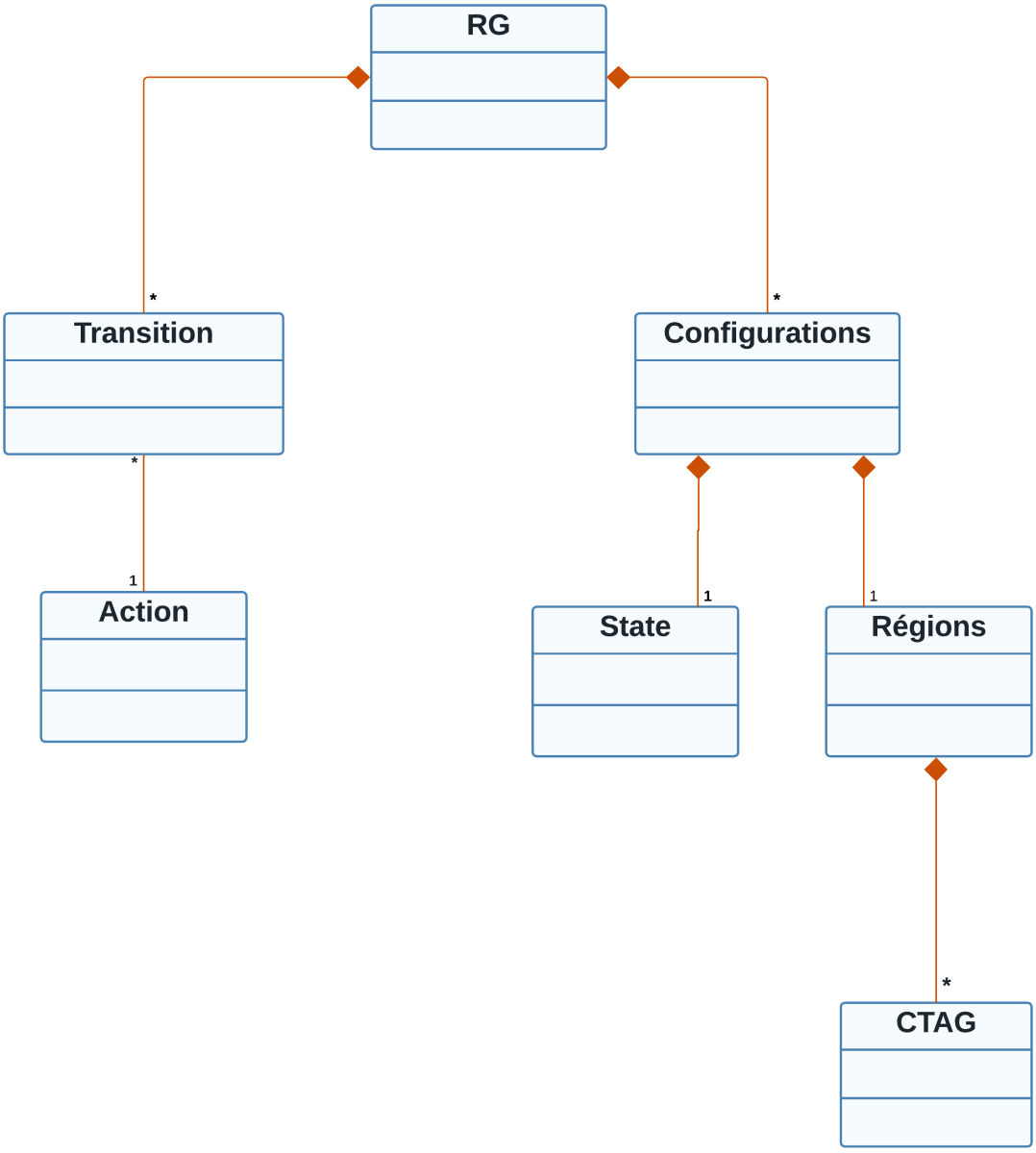


FIGURE 5.2 – Diagramme de classes de l'automate de régions.

5.3 Les environnement utilisés dans notre mémoire

Apres avoir présenté les moyens matériels mise à notre disposition dans le cadre de réalisation de cette application, nous abordons dans cette partie les moyens logiciels utilisés. Ces logiciels sont utilisés pour la réalisation de ce projet ainsi que pour la rédaction du rapport sont :

Environnement logiciel :

— **Overleaf**
Dans notre mémoire, nous avons utilisé l'outil Overleaf pour la rédaction du manuscrit au format LaTeX. Overleaf est une plateforme en ligne qui permet de créer, partager et collaborer sur des documents LaTeX

FIGURE 5.3 – Logo Overleaf

— **Intellij IDEA**
IntelliJ IDEA est un IDE pour les développeurs Java, créé par JetBrains et qui offre une multitude de fonctionnalités pour faciliter le développement, la compilation, le débogage et la gestion de projets Java.

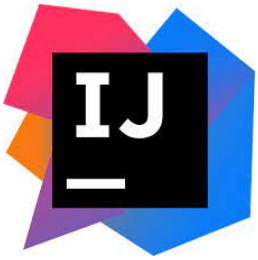


FIGURE 5.4 – Logo IntelliJ IDEA

— **Lucidchart**
Est une plateforme de collaboration en ligne, basée sur le cloud, permettant la création de diagrammes et la visualisation de données, et autres schémas conceptuels

FIGURE 5.5 – Logo Lucidchart

Langages utilisé :

— **Langage JAVA**
Java est un langage de programmation orienté objet qui est utilisé pour développer une variété d'applications. Ce qui le rend unique est sa portabilité car il utilise une machine virtuelle appelée JVM (Java Virtual Machine) qui agit comme interface entre le programme et le système d'exploitation. Cette caractéristique permet aux programmes écrits en Java d'être exécutés sur différentes plateformes sans nécessiter de modifications de code.

FIGURE 5.6 – Logo JAVA

GraphStream

GraphStream est une bibliothèque Java open-source qui permet de créer, manipuler et visualiser des graphes dynamiques. Les graphes dynamiques sont des graphes qui évoluent dans le temps, et GraphStream offre des fonctionnalités pour créer des nœuds et des arêtes dynamiques, ainsi que pour les visualiser avec différents algorithmes de disposition et de rendu. Cette bibliothèque est largement utilisée dans différents domaines, tels que la recherche en informatique, la visualisation de données, la simulation et l'analyse de réseaux.

FIGURE 5.7 – Logo GraphStream

5.4 Code source

5.4.1 Structure du code source

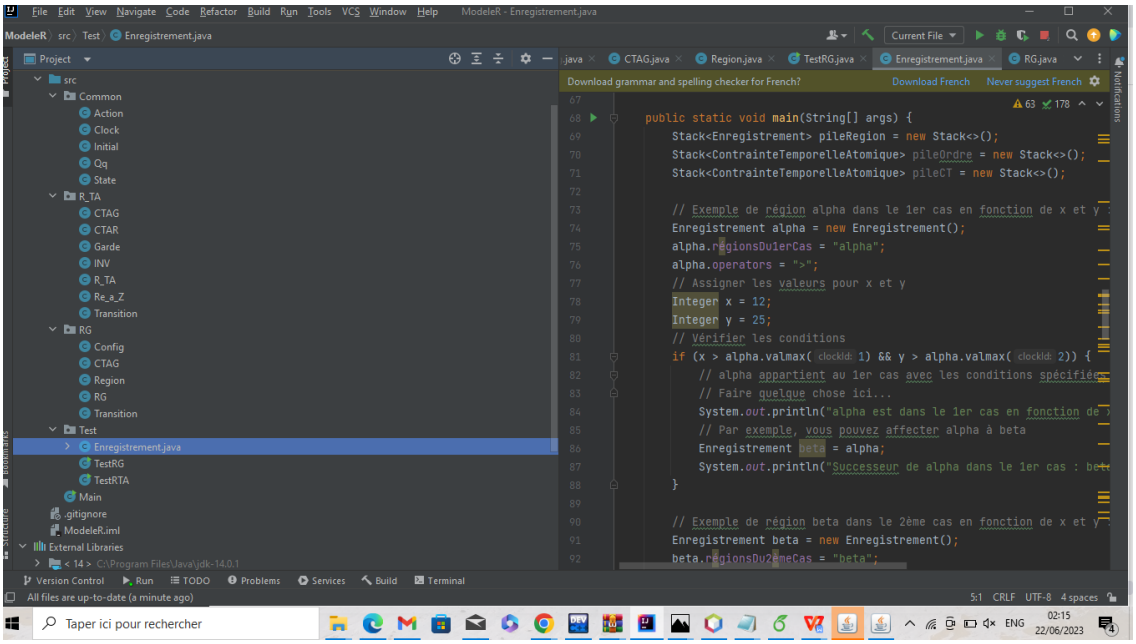


FIGURE 5.8 – Structure du Code Source

5.4.2 Schématisation la structure de r-TA

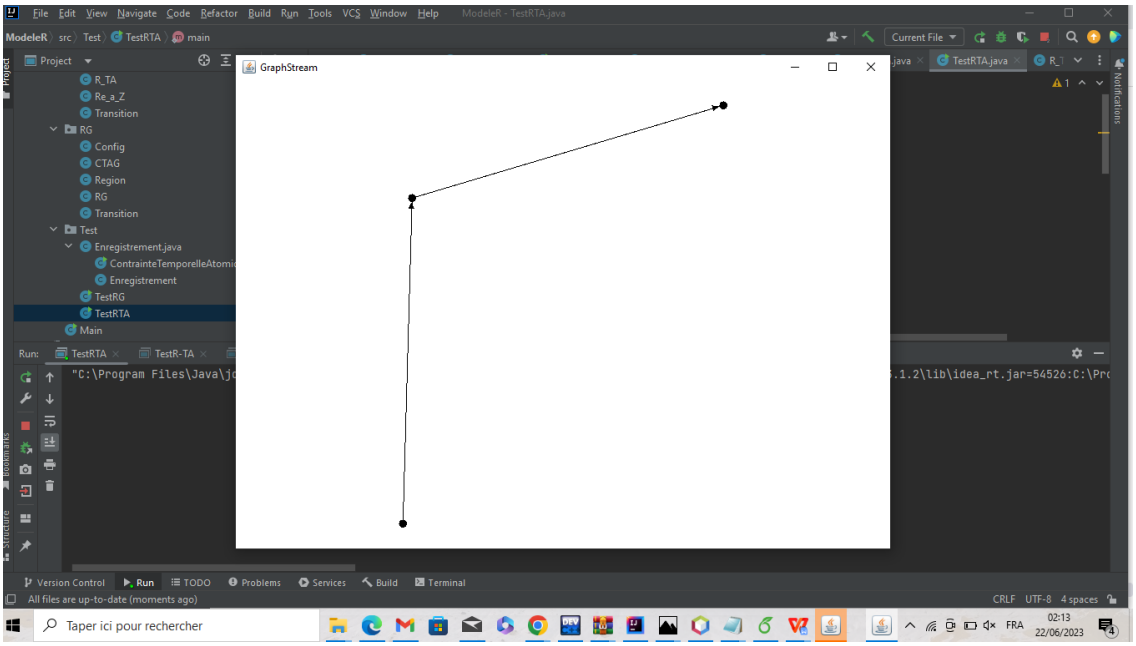


FIGURE 5.9 – Graphe de r-TA

5.4.3 Code source pour Successeur

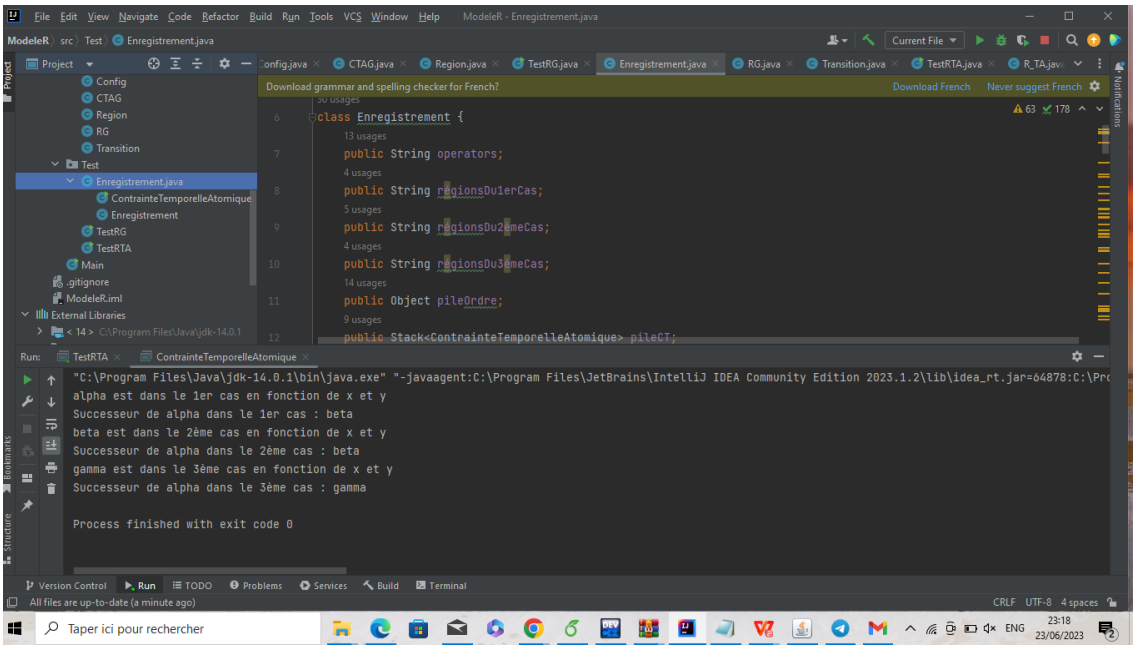


FIGURE 5.10 – Code Source du Successeur

5.4.4 Schématisation la structure de RG

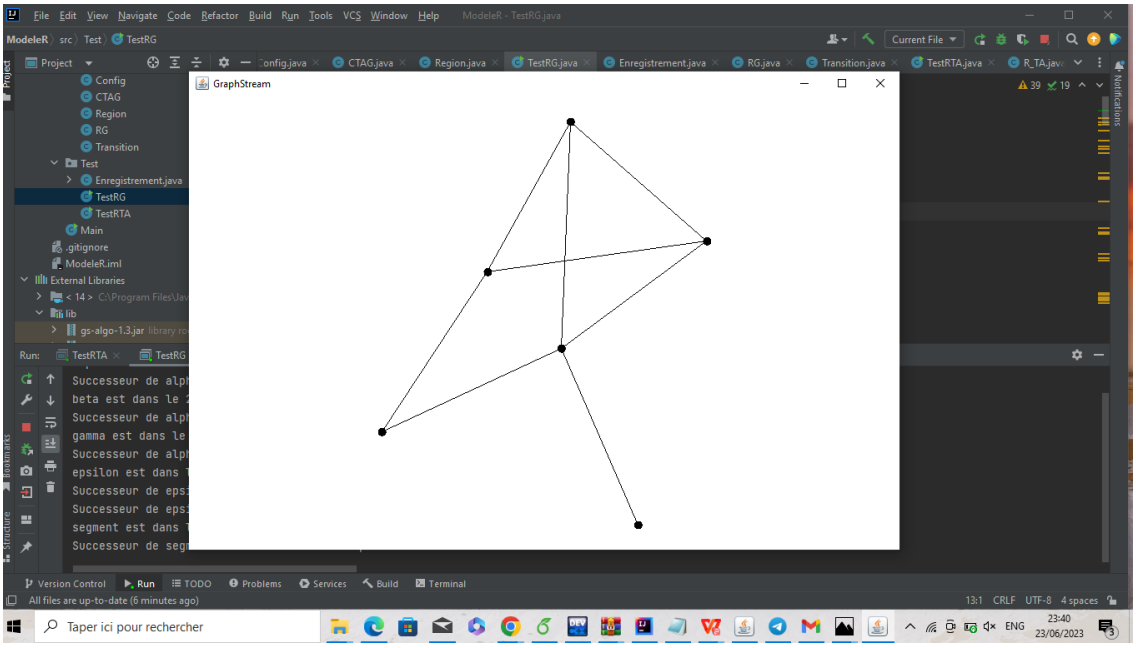


FIGURE 5.11 – Graphe de RG

5.4.5 Conclusion

Dans ce chapitre, nous avons introduit le langage de programmation Java et présenté l'outil de développement Intellij IDEA. Nous avons également montré les deux diagrammes de classes qui ont été modélisés à l'aide d'un logiciel de dessin graphique appelé Lucidchart.

Chapitre 6

Conclusion générale et perspectives

Sommaire

6.1 Bilan

En conclusion, ce travail représente une avancée majeure dans le domaine de la modélisation et de la vérification des systèmes temps-réel, en se concentrant sur les aspects de réactivité, d'hétérogénéité et de distribution. Les approches basées sur le model-checking ont connu une expansion considérable ces derniers temps, avec une attention particulière portée aux propriétés temporelles.
Pour tenir compte de la nature distribuée des systèmes, il a été postulé que les différents composants évoluent à des rythmes distincts. L'approche développée dans ce travail a été spécifiquement appliquée au domaine des systèmes temps réel hétérogènes, qui reste d'une grande importance dans le domaine des applications informatiques.
Les contributions majeures de cette étude peuvent être résumées comme suit :
- Proposition d'une approche novatrice pour analyser les systèmes temps-réel hétérogènes, prenant en compte l'évolution des horloges associées à différents processus à des fréquences variables mais relatives. - Démonstration de la décidabilité de l'accessibilité grâce à une nouvelle abstraction des comportements des systèmes, nécessitant une redéfinition de plusieurs concepts essentiels.
- Introduction du modèle r-TA pour interpréter les comportements temporels dans les systèmes temps réel hétérogènes et coordonnés.
- Utilisation du paramètre "slope" afin d'étudier la sémantique de ces systèmes et de redéfinir la relation d'équivalence sur les valuations des horloges d'un r-TA. - Mise en évidence de la finitude du nombre de régions d'horloges associées à un r-TA, permettant ainsi de redéfinir la relation de successeur et de construire un automate de régions. Cette construction offre la possibilité de tester le vide et de réaliser le model-checking de manière décidable.
- Pour garantir la terminaison de la construction d'un automate de régions, il est nécessaire d'avoir un index fini de régions et une relation de successeur sur cet index. Cet automate de régions spécifie les mêmes comportements que ceux définis par la

spécification r-TA.
En résumé, ce travail représente une contribution significative à l'analyse et à la vérification des systèmes temps-réel hétérogènes, en proposant des modèles et des algorithmes adaptés. Les résultats obtenus ont un impact important sur la compréhension et l'étude des systèmes temps-réel, et ouvrent de nouvelles perspectives pour des recherches futures dans ce domaine sur cet index, afin de garantir la terminaison du processus de construction. En effet, l'existence d'un index fini de régions et d'une relation de successeur sur cet index assure que la construction de l'automate de régions aboutira à un résultat final. Cette condition est essentielle pour obtenir un automate de régions qui spécifie les mêmes comportements que ceux définis par la spécification r-TA. Ainsi, la définition et la mise en place appropriée de cet index et de la relation de successeur sont des éléments cruciaux dans la démarche de construction de l'automate de régions.

6.2 Perspectives

En ce qui concerne les perspectives de l'automate de régions, plusieurs directions de recherche peuvent être envisagées :
Étude de la sémantique des spécifications des systèmes ambiants : Une perspective intéressante consiste à évaluer les capacités du modèle de l'automate de régions pour décrire la sémantique des spécifications des systèmes ambiants. Les systèmes ambiants sont des environnements informatiques où des dispositifs interagissent de manière autonome pour fournir des services aux utilisateurs. Il serait intéressant d'explorer comment le modèle de l'automate de régions peut être adapté pour prendre en compte les spécificités et les contraintes des systèmes ambiants.
Construction de l'automate de zones : L'application de l'approche du model-checking nécessite une extension des résultats existants pour la construction de l'automate de zones. L'automate de zones est une structure mathématique utilisée pour représenter les comportements temporels des systèmes temps-réel. Son utilisation permettrait de développer des outils de vérification plus efficaces et implémentables basés sur l'approche de l'automate de régions.
Intégration de concepts issus d'autres modèles : Des modèles tels que les automates hybrides et les automates avec prix ont déjà été développés et se sont révélés utiles dans certains domaines. Une perspective intéressante serait d'étudier la possibilité d'intégrer des concepts similaires issus de ces modèles dans l'approche de l'automate de régions. Cela pourrait enrichir les capacités du modèle et permettre de traiter des problèmes spécifiques de manière plus efficace.
Comparaison formelle avec d'autres modèles de spécification : Il serait également intéressant d'explorer des exemples concrets dans des applications réelles et de les comparer formellement à d'autres modèles de spécification célèbres tels que les réseaux de Petri et leurs diverses extensions. Cette comparaison permettrait de mieux comprendre les avantages et les limitations de l'approche de l'automate de régions par rapport à d'autres approches existantes et de déterminer les domaines d'application les plus appropriés pour chaque modèle.
En explorant ces perspectives, il serait possible d'étendre et d'améliorer les résultats

existants, tout en ouvrant de nouvelles possibilités de recherche et d'application dans le domaine de la modélisation et de la vérification des systèmes temps-réel.

Bibliographie

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.

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

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. (1994a). A theory of timed automata. *Theoretical Computer Science*, 126(2) :183–235.

Alur, R. and Dill, D. L. (1994b). 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.

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., 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.

Berrou, Y. (2010). *Evaluation de la fiabilité des systèmes temps réel distribués embarqués*. PhD thesis, Université de Batna 2.

Bourdil, P.-A. (2015). *Contribution à la modélisation et la vérification formelle par model checking-Symétries pour les Réseaux de Petri temporels*. PhD thesis, Toulouse, INSA.

Bouyer, P. (2002). Modèles et algorithmes pour la vérification des systèmes temporisés. *These de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France*.

Bouyer, P. (2009). From qualitative to quantitative analysis of timed systems. *Mémoire d'habilitation, Université Paris*, 7 :135–175.

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.

Burns, A. and Wellings, A. (2009). Real-time systems and programming languages : Ada, real-time java and c/real-time posix.

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

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

Coulouris, G. F., Dollimore, J., and Kindberg, T. (2005). *Distributed systems : concepts and design*. pearson education.

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

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

Fiorini, A. (2020). *Traçage Et Profilage De Systèmes Hétérogènes*. PhD thesis, Ecole Polytechnique, Montreal (Canada).

Fleury, E. (2002). *Les automates temporisés avec mises à jour*. PhD thesis, École normale supérieure de Cachan-ENS Cachan.

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

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.

KHITER, A. and HELLAL, C. (2020). *Conception d'approche de r´ egulation pour l'ordonnancement temps r´ eel des tˆaches*. PhD thesis, Université Akli Mohand Oulhadje-Bouira.

Knight, J. C. (2002). Safety critical systems : challenges and directions. In *Proceedings of the 24th international conference on software engineering*, pages 547–550.

Kopetz, H. et al. (2011). Real-time systems : design principles for distributed embedded applications.

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

Kumar, S. S. and Cheng, J. C. (2015). A bim-based automated site layout planning framework for congested construction sites. *Automation in Construction*, 59 :24–37.

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.

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

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.

Layadi, S. (2017). *rd-TA et daTA-R : modèles de temps relatif pour les systèmes temps-réel hétérogènes*. PhD thesis.

Lenzen, C., Locher, T., Sommer, P., and Wattenhofer, R. (2010). Clock synchronization : Open problems in theory and practice. In *SOFSEM 2010 : Theory and Practice of Computer Science : 36th Conference on Current Trends in Theory and Practice of Computer Science, Špindlerv Mlỳn, Czech Republic, January 23-29, 2010. Proceedings 36*, pages 61–70. Springer.

Luong, H.-V. (2010). *Construction incrémentale de spécifications de systèmes critiques intégrant des procédures de vérification*. PhD thesis, Université Paul Sabatier-Toulouse III.

Mehiaoui, A. (2014). *Techniques d'analyse et d'optimisation pour la synthèse architecturale de systèmes temps réel embarqués distribués : problèmes de placement, de partitionnement et d'ordonnancement*. PhD thesis, Brest.

Midonnet, S. (2007). La spécification java pour le temps réel. In *5ème Ecole d'été Temps Réel (ETR'07)*, pages 201–209. Citeseer.

Misra, S., Krishna, P. V., Saritha, V., Agarwal, H., and Ahuja, A. (2014). Learning automata-based multi-constrained fault-tolerance approach for effective energy management in smart grid communication network. *Journal of Network and Computer Applications*, 44 :212–219.

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

Rodriguez-Navas, G. and Proenza, J. (2012). Using timed automata for modeling distributed systems with clocks : Challenges and solutions. *IEEE Transactions on Software Engineering*, 39(6) :857–868.

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

Shafei, E., Moawad, I., Sallam, H., and Mostafa, A. (2013). A methodology for safety critical software systems planning. In *Proceedings 7th WSEAS European Computing Conference (ECC'13), Dubrovnik, Croatia*.

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

Tibermacine, O. (2009). *UML et model checking*. PhD thesis, Université de Batna 2.

Verissimo, P. (1994). Ordering and timeliness requirements of dependable real-time programs. *Real-time systems*, 7(2) :105–128.

Wang, B. X. N. and Li, C. (2011). A cloud computing infrastructure on heterogeneous computing resources. *Journal of computers*, 6(8).

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.

Zhao, Y. (2014). *Modélisation qualitative des agro-écosystèmes et aide à leur gestion par utilisation d'outils de model-checking*. PhD thesis, Université Rennes 1.