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

Spécialité :Systèmes Informatique (SI)

T H E M E

Proposition et implémentation d'un algorithme basé zones pour la vérification formelle des systèmes temps réel hétérogènes

Réalisé par : MEKHNACHE Aya
BOURBIA Ouedjdane

Encadré par : Dr. LAYADI Saïd

Session : Juin 2023

Résumé

Les méthodes formelles sont de plus en plus couramment employées pour analyser le comportement des systèmes temps-réel. Ces méthodes utilisent des modèles formels de spécification qui ont des sémantiques clairement définies et des techniques de vérification formelle.

Les automates temporisés sont un modèle largement étudié. Sa décidabilité a été prouvée à l'aide de la construction des automates de régions. Cette construction fournit une abstraction correcte des comportements des automates temporisés, mais elle ne permet pas une implémentation naturelle et, en pratique, les algorithmes basés sur la notion de zones sont implémentés en utilisant des structures de données adaptées comme les DBM.

Notre étude porte sur les systèmes temps réel critiques et hétérogènes, ce qui implique la prise en compte de l'impact de la différence entre la fréquence de traitement des différents composants du système informatique. Cette hétérogénéité est modélisée par la notion "temps relatif" dans le modèle des automates temporisés.

Dans ce mémoire, nous étudions en détails l'algorithme de construction de l'automate de zones. Cet algorithme nous permet de définir une nouvelle construction finie qui répond aux besoins du model-checking de la catégorie des systèmes sus-cités.

Mots clés : systèmes temps-réel, systèmes critiques, systèmes hétérogènes, vérification formelle, automates temporisés, zones, DBM, r-TA, model-checking.

Remerciements

Nous tenons à remercier en premier lieu Dieu le tout-puissant de nous avoir accordé la santé, la force et la volonté nécessaires pour réaliser ce travail.

Nous souhaitons également exprimer notre gratitude envers notre encadreur, Dr LAYADI Saïd, pour nous avoir encadrés, orientés, aidés et conseillés tout au long de ce projet. .

Nos remerciements vont également aux membres du jury qui ont accepté la lourde tâche d'évaluer et d'enrichir notre travail grâce à leurs remarques constructives.

Enfin, nous tenons à remercier tous nos enseignants et toutes les personnes qui nous ont aidé et soutenue de prés ou de loin.

Dédicace

Nous dédions ce travail à chacune des membres de nos familles respectives. Les familles "BOURBIA" , "MEKHNACHE" ,"TREA" et "LAROUM".

Table des matières

1 Introduction générale **1**
- 1.1 Motivations et contributions du travail . . . 2
- 1.2 Plan du mémoire . . . 3

2 Systèmes temps réel, critiques et hétérogènes **5**
- 2.1 Introduction : . . . 5
- 2.2 Systèmes temps réel (real-time systems) . . . 6
  - 2.2.1 Déférentes définitions : . . . 6
  - 2.2.2 Les structures des systèmes temps réel . . . 10
  - 2.2.3 Domaines d'application . . . 11
- 2.3 Systèmes critiques . . . 11
  - 2.3.1 Définition . . . 11
  - 2.3.2 Systèmes critiques pour la sécurité . . . 11
  - 2.3.3 Utilisation du système . . . 12
  - 2.3.4 Domaines d'application . . . 13
- 2.4 Systèmes hétérogènes . . . 13
  - 2.4.1 Déférentes définitions : . . . 13
  - 2.4.2 Domaines d'application . . . 13
  - 2.4.3 Exemples pour les systèmes hétérogènes . . . 14
- 2.5 Les diverses méthodes de vérification du système . . . 17
  - 2.5.1 Analyse statique(static analysis) . . . 17
  - 2.5.2 Preuve assistée(theorem proving) . . . 18
  - 2.5.3 Vérification du modèle (model-cheking) . . . 18
- 2.6 Les avantages de la vérification formelle . . . 19
- 2.7 Rapport de stage . . . 20
  - 2.7.1 La société Sonatrach . . . 20
  - 2.7.2 Description générale sur l'usine . . . 21
  - 2.7.3 Le système SCADA . . . 21
- 2.8 Conclusion . . . 24

3 Les automates temporisés **25**
- 3.1 Introduction : . . . 25
- 3.2 Notions préliminaires : . . . 26
- 3.3 Langages temporisés . . . 27
  - 3.3.1 Notions : . . . 27
- 3.4 Automates temporisés . . . 27
  - 3.4.1 Définition 01 : . . . 27
  - 3.4.2 Définition 02 : . . . 28
- 3.5 Sémantique d'un automate temporisé . . . 28

3.6 Propriétés des automates temporisés . . . 29
3.7 Problème du vide . . . 30
3.7.1 Description du problème . . . 30
3.7.2 Décidabilité des automates temporisés . . . 31
3.7.3 Les régions . . . 32
3.8 Les zones . . . 35
3.8.1 Les DBMs . . . 36
3.8.2 Graphe des zones . . . 38
3.8.3 Approximations et améliorations . . . 39
3.9 L'approche d'analyse en arrière . . . 39
3.10 L'approche d'analyse en avant . . . 40
3.11 Outils de vérification : . . . 41
3.12 Conclusion . . . 43

4 Automate temporisé avec temp relatif r-TA 45
4.1 Introduction . . . 45
4.2 Automates temporisés avec temp relatif . . . 46
4.2.1 Définition . . . 46
4.2.2 Sémantique . . . 47
4.3 Véréfication automatique . . . 47
4.3.1 Paramètre *slope* et les régions d'horloges . . . 48
4.3.2 Classes d'équivalence sur les valuations d'horloges . . . 49
4.3.3 Successeurs des régions d'horloges . . . 50
4.4 Automate de régions . . . 52
4.4.1 Définition . . . 52
4.5 Automate de zones avec temps relatif . . . 53
4.5.1 Définition . . . 53
4.5.2 Algorithme de zones classique . . . 54
4.5.3 Algorithme de zones avec temps relatif . . . 55
4.5.4 Les r-DBMs . . . 55
4.5.5 Exemple : . . . 56
4.6 Conclusion . . . 62

5 Conception et implémentation 63
5.1 Introduction . . . 63
5.2 Langage de modélisation UML . . . 64
5.2.1 A quoi sert UML ? . . . 64
5.2.2 Diagrammes de classes . . . 64
5.3 Langage de développement java . . . 67
5.3.1 Présentation du java . . . 67
5.3.2 Caractéristiques de base du langage Java . . . 67
5.4 Outils de développement . . . 67
5.4.1 L'environnement Netbeans . . . 67
5.4.2 L'environnement Overleaf . . . 68
5.4.3 Pacestar UML Diagrammer . . . 69
5.5 Présentation du l'application . . . 69

5.5.1 Interface principale 69
5.5.2 Interface du résultat 70
5.6 Conclusion 71

6 Conclusion générale **73**

Bibliographie **73**

Table des figures

2.1 La valeur d'un calcul en fonction de son instant d'arrivé (temps strict) (58) . 7
2.2 La valeur d'un calcul en fonction de son instant d'arrivée (temps souple) (58). 8
2.3 La valeur d'un calcul en fonction de son instant d'arrive (Temps ferme) (58). 9
2.4 Structure générale d'un système temps réel 10
2.5 Présentation du model-cheking 19
2.6 Système SCADA 22

3.1 Automate temporisé simple 28
3.2 Représentation un région 33
3.3 Représentation des autres régions 33
3.4 Automate temporisé A 34
3.5 Automate des régions associé à A 34
3.6 Représentation graphique de zone 35
3.7 Automate de zone 38
3.8 Le calcul, pas à pas, des prédécesseurs de l'état final dans l'analyse en arrière. 39
3.9 Le calcul, pas à pas, des successeurs de l'état initial dans l'analyse en avant. 40
3.10 Un automate temporisé qui illustre la non terminaison de l'analyse en avant. 41
3.11 Le calcul itératif en avant associé. 41

4.1 Evolution de deux horloges avec différentes fréquences 48
4.2 Différentes possibilités de l'évolution de deux horloges 48
4.3 Réinitialisation de deux horloges avec différentes fonctions linéaires de vitesses 49
4.4 Exemple d'un r-TA 57
4.5 Future Z_0 58
4.6 Future Z_1 59
4.7 Future Z_2 61
4.8 Automate de zone de r-TA 61

5.1 Diagramme du r-TA 65
5.2 Diagramme de classes de l'automate de zones 66
5.3 Logo JAVA 67
5.4 Logo Overleaf 68
5.5 Logo Pacestar UML Diagrammer 69
5.6 Interface principale avec un exemple 70
5.7 Interface de résultat 71

Liste des tableaux

2.1 Echantillon de systèmes durs, souples, et fermes (63) 9

3.1 DBM de la contrainte $1 \leq x_1 \leq 4 \wedge 1 \leq x_2 \leq 3 \wedge x_1 - x_2 \leq 1$ 36
3.2 DBM de la garde g 37
3.3 DBM de $z \cap g$ 37
3.4 DBM de $z \cap g[\{x_2\} := 0]$ 37

Liste d'abréviations

AT **A**utomate **T**emporisé

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

AR **A**utomate de **R**égions

AZ **A**utomate de **Z**ones

STT **S**ystème de **T**ransitions **T**emporisé

LTS **S**ystème de **T**ransitions **L**abeled (étiqueté)

DBM **M**atrices à **D**ifférences **B**ornées

r-DMB **M**atrices à **D**ifférences **B**ornées avec temps **r**elatif

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

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

GR **G**raphe des **R**égions

GZ **G**raphe des **Z**ones

Chapitre 1 Introduction générale

Sommaire

1.1 Motivations et contributions du travail 2
1.2 Plan du mémoire 3

De nos jours, l'informatique est omniprésente dans de nombreuses applications essentielles. Ces applications sont liées à des enjeux importants tels que la sécurité des personnes, la protection des données sensibles et la gestion de ressources financières importantes. les systèmes de temps réel critiques et hétérogènes font partie de ces applications et doivent donc être vérifiés.

Un système de temps réel est un système qui doit réagir à des événements dans un délai précis et dont la défaillance peut entraîner des pertes humaines, matérielles ou d'autres conséquences graves, alors la c'est un système critique. Les systèmes de temps réel ont des contraintes temporelles telles que le temps de réponse, l'échéance, la date d'exécution la plus proche, la cadence, etc. Ils doivent produire des résultats corrects d'un point de vue logique et les exécuter dans le temps imparti. La vérification des systèmes de temps réel et critique peut être effectuée à l'aide de méthodes formelles telles que checking. Les systèmes de temps réel critiques peuvent être hétérogènes et multiprocesseurs, ce qui peut apporter des gains de performance mais rend également leur analyse plus complexe. L'isolation des systèmes de temps réel est nécessaire pour garantir la fiabilité et la sécurité en limitant ou en empêchant certaines interactions entre les différents composants d'un système. Les systèmes à criticité mixte sont un type de système de temps réel qui doit fournir certaines garanties quant à sa capacité à effectuer des tâches critiques.

La vérification formelle des systèmes de temps réel critiques et hétérogènes est une discipline cruciale dans le domaine de l'informatique et de l'ingénierie des systèmes. Les systèmes de temps réel critiques sont ceux dont le bon fonctionnement dépend du respect de contraintes temporelles strictes. Ils sont souvent utilisés dans des domaines sensibles tels que l'aérospatiale, l'automobile, les systèmes médicaux et l'industrie nucléaire, où une défaillance peut avoir des conséquences graves, voire catastrophiques. L'hétérogénéité des systèmes de temps réel critiques pose un défi supplémentaire en raison de la diversité des composants matériels, logiciels et de communication impliqués.

La vérification formelle vise à garantir que ces systèmes fonctionnent correctement et en toute sécurité en utilisant des techniques rigoureuses basées sur des fondements mathé-

matiques solides telles que le model checking, la méthode proving et l'analyse statique. Le model checking est une méthode formelle qui vérifie si un modèle répond à certaines propriétés spécifiées. La méthode proving utilise des preuves mathématiques pour prouver la validité d'un système. L'analyse statique examine le code source pour détecter les erreurs potentielles.

Parmi les différentes méthodes de vérification formelle, le model checking occupe une place importante. Le model checking est une technique de vérification automatique qui consiste à explorer de manière exhaustive tous les états possibles d'un modèle afin de vérifier si certaines propriétés spécifiées sont satisfaites. Cette approche présente l'avantage d'une couverture complète des états du système, permettant ainsi de détecter des erreurs potentielles.

Dans le cadre de ce mémoire, nous proposons d'utiliser la technique du model checking comme méthode de vérification pour les systèmes de temps réel critiques et hétérogènes. Cependant, étant donné la complexité et la taille souvent considérable de ces systèmes, l'exploration exhaustive de tous les états devient rapidement impraticable. C'est pourquoi nous adoptons une approche basée sur la notion de zones, en nous concentrant sur des régions spécifiques de l'espace de conception.

1.1 Motivations et contributions du travail

La vérification formelle joue un rôle crucial dans l'assurance de la validité et de la conformité des systèmes de temps réel critiques par rapport à leurs spécifications. Parmi les différentes méthodes de vérification disponibles, le model checking est largement reconnu comme l'une des approches les plus performantes et rigoureuses. Dans notre mémoire, nous nous concentrons sur l'application du model checking aux systèmes de temps réel critiques et hétérogènes.

Notre objectif est de vérifier des systèmes de temps réel critiques et hétérogènes . Pour ce faire, nous avons choisi l'automate avec temps relatif comme théorie de modèle de vérification. Nous allons donc transformer un automate temporisé avec temps relatif en un automate de zones. Cette transformation est nécessaire car le model checking requiert une structure finie en entrée, ainsi que des propriétés formulées formellement pour effectuer la vérification. Pour cela, nous proposons un algorithme de transformation spécifique permettant de construire l'automate de zones correspondant à partir de l'automate temporisé avec temps relatif , initialement infini.

Il est important de noter que, bien que d'autres approches, telles que les automates de régions, puissent également être utilisées pour traiter les systèmes comportant des aspects temporels, nous avons choisi de ne pas les utiliser car elles sont complexes et longues à mettre en œuvre, ce qui peut limiter leur applicabilité dans le contexte des systèmes de temps réel critiques.

En outre, nous prenons en compte l'impact de la différence entre la fréquence de traitement des différents composants du système informatique. Cette hétérogénéité est modélisée

par la notion "temps relatif" dans le modèle des automates temporisés. Les systèmes critiques peuvent présenter des caractéristiques temporelles complexes, telles que des horloges fonctionnant à différentes périodes ou des contraintes de délai strictes. Nous prenons en considération ces aspects lors de la transformation de l'automate temporisé en automate de zones, afin de garantir une vérification précise et pertinente pour les systèmes critiques concernés.

En résumé, notre mémoire se concentre sur l'utilisation du model checking pour la vérification formelle des systèmes de temps réel critiques et hétérogènes. Nous proposons une transformation d'automate temporisé avec temps relatif vers un automate de zones, grâce à un algorithme spécifique que nous développons. Nous choisissons cette approche en raison de sa performance et de sa rigueur. Nous évitons l'utilisation des automates de régions en raison de leur complexité et du temps qu'ils nécessitent. De plus, nous prenons en compte les caractéristiques temporelles spécifiques de cette catégorie de systèmes étudiée lors de notre démarche de transformation. Notre travail contribue ainsi à renforcer la validité et la fiabilité des systèmes de temps réel critiques et hétérogènes grâce à une vérification formelle avancée.

1.2 Plan du mémoire

Le reste de ce mémoire est organisé de la manière suivante :

Le Chapitre 2 : Dans ce chapitre nous avons présenté les concepts fondamentaux des systèmes temps réel, critiques et hétérogènes. Nous avons également examiné les différents domaines d'application et méthodes de vérification de ces systèmes.Et à la fin , nous avons brièvement décrit l'entreprise Sonatrach et son usine, ainsi que le système SCADA utilisé pour la supervision et le contrôle de l'usine.

Le Chapitre 3 : Le chapitre est consacré à la modèle célèbre des automates temporisés. Nous en présentons les notions de base de ce modèle, à savoir la syntaxe, la sémantique, la composition de plusieurs automates temporisés. . .; et nous insisterons sur l'utilisation de ce modèle dans la vérification formelle , ensuit nous allons sur le problème de décidabilité et proposition du leur solution par donner quelques concepts sur la pratique de la vérification, notamment les deux solution régions et zone (approches d'analyse des systèmes en avant et en arrière, les structures DBM, les graphes de zones) et une présentation succincte sur les outils de vérification comme Uppaal .

Le Chapitre 4 : Dans ce chapitre on commence par définir l'automate temporisé avec temps relatif, en mettant en évidence ses caractéristiques principales et leur sémantique, et nous avons parler sur la solution de régions qui déjà proposé, et la nouvelle approche basée zone qui nous avons proposé .

Le Chapitre 5 : nous avons présenté la partie de la réalisation de notre application et le langage de programmation et les techniques et les outils utilisés.

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

Chapitre 2

Systèmes temps réel, critiques et hétérogènes

Sommaire

2.1 Introduction : 5
2.2 Systèmes temps réel (real-time systems) 6
2.2.1 Déférentes définitions : 6
2.2.2 Les structures des systèmes temps réel 10
2.2.3 Domaines d'application 11
2.3 Systèmes critiques 11
2.3.1 Définition 11
2.3.2 Systèmes critiques pour la sécurité 11
2.3.3 Utilisation du système 12
2.3.4 Domaines d'application 13
2.4 Systèmes hétérogènes 13
2.4.1 Déférentes définitions : 13
2.4.2 Domaines d'application 13
2.4.3 Exemples pour les systèmes hétérogènes 14
2.5 Les diverses méthodes de vérification du système 17
2.5.1 Analyse statique(static analysis) 17
2.5.2 Preuve assistée(theorem proving) 18
2.5.3 Vérification du modèle (model-cheking) 18
2.6 Les avantages de la vérification formelle 19
2.7 Rapport de stage 20
2.7.1 La société Sonatrach 20
2.7.2 Description générale sur l'usine 21
2.7.3 Le système SCADA 21
2.8 Conclusion 24

2.1 Introduction :

L'introduction de la contrainte du temps inhérent à l'application, c'est à dire aux processus à contrôler en spécifiant des délais avec les quels seront produites les réponses aux stimuli permet d'introduire la notion de système temps réel. Ces systèmes réalisent des tâches très complexes et souvent critiques, donc ce système doit respecter toutes les contraintes temporelles spécifiées avant toute exécution réelle du système d'une part, d'autre part un point commun entre ces différents systèmes, c'est qu'ils possèdent tous, de nos jours, un ou plusieurs processeur(s) ou microcontrôleur(s) pour les gérer. On parle alors des systèmes hétérogènes. Ce chapitre porte principalement sur les systèmes en temps réel, les systèmes critiques et les systèmes hétérogènes, ainsi que leur classification et leur domaine d'application.

2.2 Systèmes temps réel (real-time systems)

2.2.1 Déférentes définitions :

Définition 01 :

Les systèmes temps réel sont des systèmes numériques, réactifs, parce qu'ils réagissent continuellement aux stimuli venant de leur environnement considéré comme externe au système. Ces systèmes permettent l'implantation d'applications temps réel où le respect des contraintes temporelles et la principale contrainte à satisfaire. Un tel système pour qu'il fonctionne correctement doit obligatoirement réagir à chacun des stimuli qu'il reçoit. La réponse aux stimuli d'entrée ne dépend pas seulement des stimuli, mais aussi de l'état du système quand les stimuli arrivent. L'interface entre un système temps réel et son environnement est constituée par un ensemble de périphériques d'entrée appelés capteurs servant à la collecte des signaux émis par l'environnement et de sortie appelés actionneurs fournissant à l'environnement les commandes du système de contrôle. La validité des systèmes temps réel ne dépend pas seulement de l'exactitude logique des calculs (exactitude des traitements) mais aussi de la date à laquelle le résultat est produit (Exactitude temporelle) (5)
Ainsi, un système temps réel doit satisfaire les deux contraintes suivantes : Exactitude des traitements : calculer les bonnes sorties du système temps réel (contrôleur) en fonction de ses entrées et de l'état du système physique (procédé). Exactitude temporelle : les résultats de calcul sont présentés au bon moment (un calcul juste mais hors délai est un calcul faux).

Définition 02 :

Est un système qui permet de réaliser certaines tâches ou fonctions en réagissant avec son environnement qui lui-même évolue avec le temps, il exploite des ressources limitées. La correction d'un système temps réel ne dépend pas seulement des valeurs des résultats produits, mais également des délais dans lesquels ces résultats sont réalisés. En d'autres termes, un système est temps réel s'il est capable de respecter des échéances temporelles (20) . Un système temps réel est un système dans lequel l'exactitude des applications ne dépend pas seulement de l'exactitude du résultat mais aussi du temps auquel ce résultat est produit (57). Si les contraintes temporelles de l'application ne sont pas respectées, on parle de défaillance du système. Il est donc essentiel de pouvoir garantir le respect des contraintes temporelles du système. Ceci nécessite que le système permette un taux d'utilisation élevé, tout en respectant les contraintes temporelles identifiées (20).

Définition 03 :

Systèmes qui prennent en compte à la fois les résultats produits et les délais dans lesquels ils sont obtenus. Il précise également que les systèmes temps réel ne sont pas simplement des systèmes rapides, mais plutôt des systèmes qui doivent respecter des contraintes temporelles spécifiques. Ces contraintes varient selon le domaine d'application, par exemple la milliseconde pour les systèmes radar, la seconde pour les systèmes de visualisation humaine, quelques heures pour le contrôle de production impliquant des réactions chimiques, 24 heures

pour les prévisions météorologiques et plusieurs mois ou années pour les systèmes de navigation de sonde spatiale.

Classification des systèmes temps réel

Il s'agit d'un système où l'ensemble des contraintes temporelles doit absolument être respecté. Pour ce faire il faut pouvoir définir les conditions de fonctionnement du système, c'est-à-dire connaître parfaitement l'environnement du système. Il faut également être capable de garantir la fiabilité du système avant son exécution le classement des systèmes temps réel selon le respect des contraintes temporelles :

1. **Systèmes temps réel à contraintes strictes/dures (hard real time constraints) :** Il s'agit d'un système où l'ensemble des contraintes temporelles doit absolument être respecté, pour ce faire il faut pouvoir définir les conditions de fonctionnement du système, c'est-à-dire connaître parfaitement l'environnement du système (58) . Il faut également être capable de garantir la fiabilité du système avant son exécution. Tous les scénarios possibles d'exécution doivent donc être étudiés et le bon fonctionnement du système doit être, aussi Les systèmes à contraintes temporelles dures ne tolèrent qu'une gestion stricte du Temps afin de conserver l'intégrité du service rendu.
   L'échéance stricte (hard) : le dépassement de l'échéance provoque une exception dans le système temps réel (qui peut engendrer des dommages).

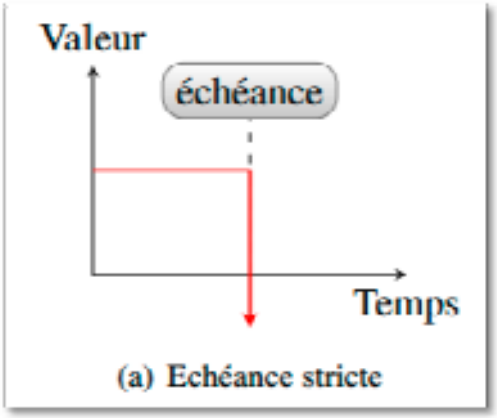

FIGURE 2.1 – La valeur d'un calcul en fonction de son instant d'arrivé (temps strict) (58)

Exemple :

- Les applications caractéristiques du temps réel dur se trouvent en aéronautique, dans le domaine automobile, dans la robotique, la commande de processus industriels.
- Contrôle de trafic aérien, système de conduite de missile

2. **Systèmes temps réel à contraintes relatives/souples (soft real time contraints) :** Les systèmes temps réel souple ont des contraintes de temps moins exigeantes que les précédents, une faute temporelle n'y est pas catastrophique pour le fonctionnement.

Un tel système pourra donc accepter un certain nombre de fautes temporelles, tout en pouvant continuer son exécution. A titre d'exemple, les applications de type multimédia telles que la téléphonie ou la vidéo sont des applications temps réel souple, car la perte de certaines informations, liée à un débit trop faible, n'est pas dangereux ou catastrophique. Les systèmes de ce type peuvent ensuite être comparés en fonction de la qualité de service qu'ils offrent, en termes de probabilité d'erreur (58). Le respect des échéances est important mais le non-respect des échéances ne peut occasionner de graves conséquences aussi (33). La réponse du système après les délais réduit progressivement est intérêt, Les pénalités ne sont pas catastrophiques (59).
Échéance molle, lâche (soft) : un dépassement d'échéance ne provoque pas d'exception dans le système temps (Bruno. Sadeg 2016).

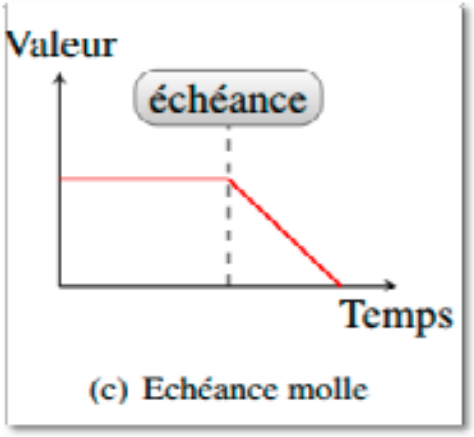

FIGURE 2.2 – La valeur d'un calcul en fonction de son instant d'arrivée (temps souple) (58).

Exemple :

- Projection vidéo (décalage entre le son et l'image) ;
- Un robot qui capte des infos sur des objets défilant sur un convoyeur.

3. **Contrainte système temps réel ferme ou non critique (firm real time) :** Temps réel souple mais où il n'y a aucun intérêt à avoir du retard ou temps réel dur pour lequel quelques échéances peuvent être occasionnellement manquées Yann Thoma ,2010). La réponse du système dans les délais est essentielle et le résultat ne sert plus à rien une fois le deadline passé.

Définition alternative La pénalité de non-réponse est dans le même ordre de magnitude que la valeur de la réponse).

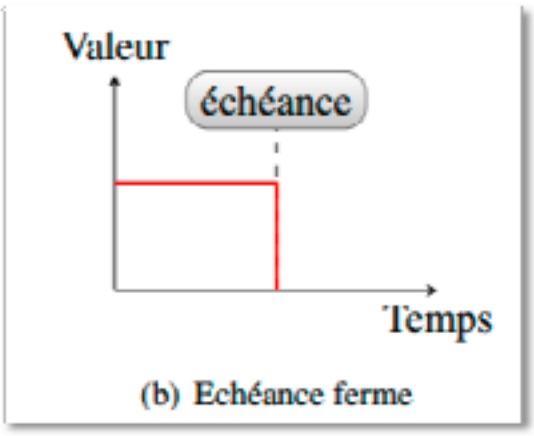

FIGURE 2.3 – La valeur d'un calcul en fonction de son instant d'arrive (Temps ferme) (58).

Exemple :
Transactions en bourse (c'est subjectif : le temps réel ferme l'un peut être le temps(réel dur de l'autre)(59)

Et voilà on donne un tableau qui représente des échantillons de systèmes (durs, souples, et fermes)(63) :

Exemple	Classification	Explication
Guichet automatique	Soft	Le non respect de nombreuses contraintes temporelles ne menr pas a une defaillance catastrophique,seules les performances sont degredees.
Controleur embarque de navigation pour un robot autonome d'elimination de mauvaises herbes	Ferme	Le non respect des contraintes critiques de navigation rend le robot definitivement hors de controle et endommage les cultures.
Systeme de mise a feu d'armes en aeronautique dans lequel l'appui un bouton met a feu un missile air-air	Dur	Le non respect d'une seule contrainte temporelle(lancement d'un missile en un temps contraint apres avoir appuye sur le bouton) peut conduire a manquer la cible, ce qui peut provoquer une catastrophe

TABLEAU 2.1 – Echantillon de systèmes durs, souples, et fermes (63)

2.2.2 Les structures des systèmes temps réel

Un système temps réel est une association logicielle matérielle où le logiciel permet, entre autre, une gestion adéquate des ressources matérielles en vue de remplir certaines tâches ou fonctions dans des limites temporelles bien précises. La partie du logiciel qui réalise cette gestion est le système d'exploitation ou noyau temps réel.(PHILIPP MABILLEAU, 2001) Ce noyau temps réel va offrir des services au(x) logiciel(s) d'application ; ces services seront basés sur les ressources disponibles au niveau du matériel.

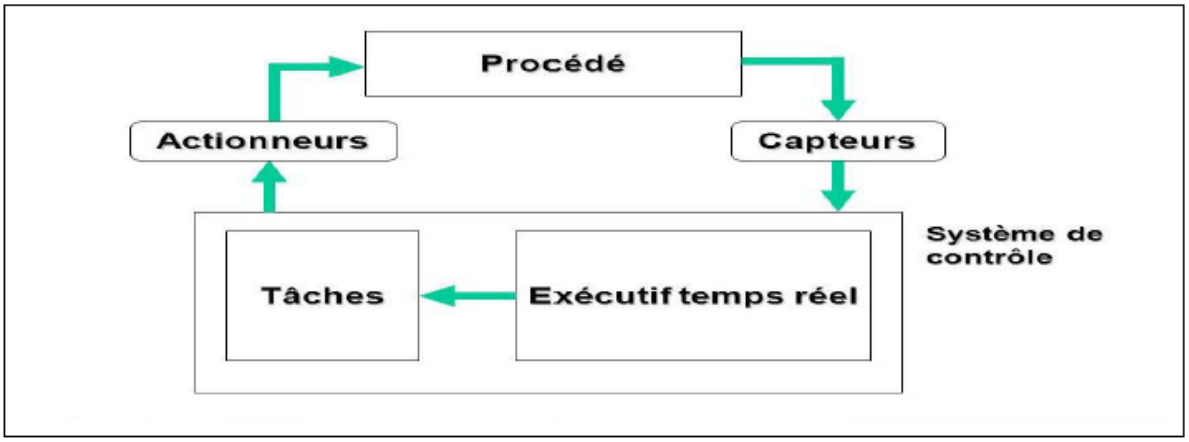

FIGURE 2.4 – Structure générale d'un système temps réel

2.2.3 Domaines d'application

Nous pouvons citer quelques d'autre exemples d'applications temps réel :

— Système de transport : que cela soit pour le transport terrestre (véhicule de tourisme, utilitaire, poids lourd, etc.), ferroviaire, aérien, ou spatial, les systèmes de transport sont caractérisés par une forte criticité et une forte complexité due à l'expansion du nombre de fonctions et la nécessité de tolérance aux fautes. Ils sont typiquement décomposés en sous-systèmes interconnectés par un ensemble de bus de terrains. (18)

— Drone volant, roulant, navigant, ou sous-marin : ce type d'applications est de plus en plus répandu, et de plus en plus d'autonomie est donnée aux drones. Qu'ils soient d'observation ou tactiques, ce type d'application trouve une large place aussi bien dans les applications militaires que civiles (exploration de zone radioactive après un accident, planétaire, sous-marine, cinéma et télévision, etc.)(18)

— Robot de production : un robot, réalisant une activité spécifique (peinture, assemblage, tri) sur une chaîne de production, doit effectuer son travail en des temps fixés par la cadence de fabrication. S'il agit trop tôt ou trop tard, l'objet manufacturier traité sera détruit ou endommagé conduisant à des conséquences financières ou humaines graves (oubli d'un ou plusieurs rivets sur un avion).(18)

— Système de vidéoconférence : ce système doit permettre l'émission et la réception d'images numérisées à une cadence de 20 à 25 images par seconde pour avoir une bonne qualité de service. Afin de minimiser le débit du réseau, une compression des images est effectuée. D'autre part la parole doit aussi être transmise. Bien que correspondant à un débit d'information moindre, la régularité de la transmission, qualifiée par une gigue temporelle, est nécessaire pour une reproduction correcte. De plus ce signal doit être synchronisé avec le flux d'images. Ces traitements (numérisations images et parole, transmission, réception, synchronisation...) sont réalisés en cascade, mais avec une cohérence précise.(18)

2.3 Systèmes critiques

2.3.1 Définition

Un système critique est un système dont une panne peut avoir des conséquences dramatiques, telles que des morts ou des blessés graves, des dommages matériels importants, ou des conséquences graves pour l'environnement. Les systèmes critiques peuvent être des systèmes de sécurité, des systèmes de mission, des systèmes informatiques, ou des systèmes électroniques. La criticité du système définit un niveau d'exigence par rapport à la tolérance aux pannes, qui aura des conséquences sur l'évaluation des niveaux d'assurance pour la sécurité. Les logiciels critiques sont des logiciels dont le mauvais fonctionnement aurait un impact important sur la sécurité ou la vie des personnes, des entreprises ou des biens.

2.3.2 Systèmes critiques pour la sécurité

La sécurité est étroitement liée à la notion de risque. Charette définit le risque comme un événement ou une action (16) :

— Avoir une perte qui y est associée ;

— Où l'incertitude ou le hasard est impliqué ;

— Certains choix est également impliqué.

La sécurité peut alors être définie comme l'absence d'exposition au danger, ou l'exemption de blessure, blessure ou perte. Mais dans la plupart des situations, on se préoccupe du degré de sécurité et donc la sécurité est une mesure subjective qui rend la fourniture et la mesure de la sécurité extrêmement difficiles et les tâches litigieuses. Les problèmes de sécurité dans les systèmes informatiques sont encore plus déroutants. De tels systèmes se composent de plusieurs sous-composants qui sont étroitement couplés et ont des interactions très complexes. La reliure de l'application au système d'exploitation à l'architecture est un excellent exemple d'un système étroitement couplé. Lorsqu'un tel système est davantage intégré dans des contextes plus larges, tels que le commandement et le contrôle systèmes, la probabilité de défaillance se rapproche rapidement de l'unité. Myers (50) estime qu'il y a environ 3,3 erreurs logicielles par millier de lignes de code dans les grands systèmes logiciels. Ce chiffre n'est pas surprenant étant donné qu'il existe jusqu'à 1020 chemins uniques de bout en bout dans un réseau de taille moyenne programme. Ce qui est pire, c'est que toutes les erreurs dans les logiciels ne sont pas égales, comme le font les petites erreurs. n'ont pas nécessairement de petits e ets...
L'image devient beaucoup plus sombre lorsque le logiciel/matériel ,les interactions dans les systèmes informatiques sont prises en compte. Dans deux études (36), près de 10 % de tous les erreurs logicielles et 35 % de toutes les pannes logicielles identifiées se sont révélées par la suite être liées au matériel, tels que les défauts transitoires corrompant les données. En fait, il semble que le matériel puisse échouer trois fois plus souvent sous forme de logiciel dans certaines circonstances. Le moyen le plus efficace pour éviter les accidents lors du fonctionnement d'un système est d'éliminer ou réduire les dangers pendant la conception et le développement, pas après quand la complexité devient accablant. La sécurité ne peut pas être considérée comme un ajout après le développement du système de la même manière qu'il n'est pas logique de concevoir un avion et de réfléchir ensuite à son poids. Nous croyons fermement que la sécurité doit être conçue dans un système et que les dangers doivent être conçu. Nous estimons également que la sécurité des logiciels et du matériel sont inextricablement liées et doivent être considérés dans leur ensemble avec une attention particulière portée aux interfaces.

2.3.3 Utilisation du système

1. La plupart des systèmes critiques sont désormais des systèmes informatisés.
2. systèmes critiques se répandent à mesure que la société devient plus complexe et que des activités plus complexes sont automatisées. (Cependant, les procédures et les pratiques qui ont évolué pour intégrer les systèmes critiques dans la société sont basées sur des systèmes beaucoup moins complexes. Dans de nombreux cas, nous ne savons pas vraiment comprendre quel sera l'impact global de ces systèmes critiques (par exemple, les systèmes automatisés de négociation d'actions)).
3. Les personnes et les processus opérationnels sont des éléments très importants des systèmes critiques - ils ne peuvent pas être simplement considérés en termes de matériel

et de logiciel (systèmes sociotechniques).

2.3.4 Domaines d'application

— Systèmes de communication tels que les systèmes de commutation téléphonique, les systèmes radio d'avion, etc ;

— Systèmes de contrôle embarqués pour usines de traitement, dispositifs médicaux (primaires) ;

— Systèmes de commandement et de contrôle tels que les systèmes de contrôle du trafic aérien, les systèmes de gestion des catastrophes ;

— Systèmes financiers tels que les systèmes de transactions de change, les systèmes de gestion de comptes (secondaires).

2.4 Systèmes hétérogènes

2.4.1 Déférentes définitions :

Définition 01 :
L'informatique hétérogène apparaît comme une exigence pour la conception de systèmes économes en énergie : les plates-formes modernes ne reposent plus sur un seul processeur à usage général, mais bénéficient plutôt de processeurs/accélérateurs dédiés adaptés à chaque tâche. Traditionnellement, ces processeurs spécialisés ont été difficiles à programmer en raison d'espaces mémoire séparés, d'interfaces au niveau du pilote du noyau et de modèles de programmation spécialisés. L'architecture système hétérogène (HSA) vise à combler cette lacune en fournissant une architecture système commune et une base pour la conception de modèles de programmation de niveau supérieur pour tous les appareils (y compris les appareils système sur puce largement utilisés, tels que les tablettes, les smartphones et autres appareils mobiles).(31)

Définition 02 :
Le niveau d'hétérogénéité dans les systèmes informatiques modernes augmente progressivement a mesure que la mise a l'échelle des technologies de fabrication permet aux composants autrefois discrets de devenir des parties intégrées d'un système sur puce, ou SoC. en logique pour l'interfaçage avec d'autres appareils (SATA, PCI, Ethernet, USB, RFID, radios, UART et contrôleurs de mémoire), ainsi que des unités fonctionnelles programmables et des accélérateurs matériels (GPU, coprocesseurs de cryptographie, processeurs de réseau programmables, A /V encodeurs/décodeurs, etc).

2.4.2 Domaines d'application

— Conception de réseaux multimédias ;

— Logiciel embarqué temps réel ;

— Co-conception matériel/logiciel ;

— Contrôle et traitement des appels dans les réseaux de télécommunications ;

— Prototypage rapide de nouveaux services de télécommunications ;
— Simulation matérielle en mode mixte ;
— Cartographier les applications sur des systèmes multiprocesseurs hétérogènes ;
— Traitement du signal mixte et contrôle en temps réel.

Par exemple, dans la conception de réseaux multimédias, nous nous intéressons à l'étude de l'interaction entre le traitement du signal de transport, de compression ou de composition, et les logiciels de contrôle en voix ou des services vidéo sur des réseaux de relais cellulaires. Dans les systèmes de télécommunication qui nous intéressent étudier l'interaction entre le logiciel de traitement des appels et les éléments matériels du commutateur. Pour développer de nouveaux services de télécommunication,nous devons concevoir conjointement des logiciels de contrôle, de traitement du signal, le transport et les éléments matériels.
Dans la conception matérielle, nous nous intéressons à la modélisation de composants avec des détails variés, tels que le comportement, la logique, la synchronisation et le circuit. Dans la conception au niveau du système, nous pouvons souhaiter concevoir conjointement l'infrastructure de communication et les éléments de traitement. Nous pouvons également souhaiter synthétiser (en combinaison) le microcode pour les processeurs spécialisés, le code C pour les processeurs génériques, les tables de routage pour les réseaux de portes programmables sur site et le VLSI personnalisé. Nous pouvons souhaiter automatiser les mappages sur processeurs parallèles en mélangeant des ordonnanceurs spécialisés, tels que ceux qui ciblent les tableaux systoliques, ou ceux qui adressent uniquement les applications avec un flux de contrôle prévisible(15).

2.4.3 Exemples pour les systèmes hétérogènes

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

prendre profiter des avantages de performance et d'efficacité de l'informatique hétérogène, le logiciel d'infrastructure cloud doit reconnaitre et gérer cette hétérogénéité. Dans le passé, le calcul en grille et la planification par lots ont tous deux couramment utilisés pour le calcul a grande échelle. Le cloud computing présente une allocation de ressources différente paradigme que les grilles ou les planificateurs de lots. En particulier, Amazon EC2 est équipe pour gérer de nombreuses allocations de ressources de calcul plus petites, plutôt que quelques requêtes volumineuses comme est normalement le cas avec le calcul en grille. L'introduction d'hétérogénéité permet aux cloud d'être compétitifs avec les systèmes informatiques distribues traditionnels, qui consistent souvent en également différents types d'architectures. Lorsqu'il est combine avec économies d'échelle, provisionnement dynamique et comparativement des dépenses en capital plus faibles, les avantages de l'hétérogénéité les nuages sont nombreux. Le cloud computing permet aux utilisateurs individuels d'avoir un accès administratif a une instance de machine virtuelle d'éditée. Le la capacité a séparer les utilisateurs est supérieure par rapport a un lot approche de planification, où il est courant pour plusieurs travaux pour partager un seul système d'exploitation. Les avantages de ce ressortent du point de vue de la sécurité ainsi que flexibilité pour les utilisateurs, offrant une variété de systèmes d'exploitation. Certaines implémentations de planification par lots, telles que LSF, dépendent d'une configuration presque identique de nœuds de calcul a travers un cluster, une tache potentiellement ardue pour le système administrateurs (47).

L'Internet des objets L'Internet des objets (ou IdO) se traduit à l'heure actuelle par l'accroissement du nombre d'objets connectés, c'est-à-dire d'appareils possédant une identité propre et des capacités de calcul et de communication de plus en plus sophistiquées : téléphones, montres, appareils ménagers, etc. Ces objets embarquent un nombre grandissant de capteurs et d'actionneurs leur permettant de mesurer l'environnement et d'agir sur celui-ci, faisant ainsi le lien entre le monde physique et le monde virtuel. Spécifiquement, l'Internet des objets pose plusieurs problèmes, notamment du fait de sa très grande échelle, de sa nature dynamique et de l'hétérogénéité des données et des systèmes qui le composent (appareils puissants/peu puissants, fixes/mobiles, batteries/alimentations continues, etc.). Nous avons pu remarquer que tous les objets n'étaient pas égaux. En effet, conformément aux usages pour lesquels ils ont été conçus, les propriétés et les capacités des objets varient significativement, contribuant à faire de l'Internet des objets un écosystème certes riche, mais aussi hétérogène. Aux caractéristiques des objets correspondent des contraintes qui conditionnent la façon dont ces objets sont utilisés et interagissent avec leur environnement. Par exemple, les objets mobiles sont plus susceptibles de souffrir d'une connectivité intermittente que les objets statiques, les objets alimentés par une batterie sont limités par leur durée de vie et les objets possédant de faibles ressources matérielles sont restreints à des tâches simples. De la même façon, les objets embarquent différents capteurs et actionneurs conformément à leurs fonctions, ce qui a un impact direct sur leurs aptitudes à mesurer et à influer sur leur environnement.

Hétérogénéité fonctionnelle Les objets possèdent des capacités spécifiques (statique ou mobile, alimenté en continu ou par une batterie, ressources matérielles, capteurs et actionneurs disponibles, etc.) et à chacune d'elle correspond des contraintes particulières (connecti-

vité intermittente, durée de vie, tâches réalisables, etc.). Différentes approches et techniques doivent être considérées pour gérer les objets en fonction de leurs contraintes et de leurs différences propres.

Hétérogénéité technique Les technologies matérielles et logicielles utilisées pour construire les objets sont multiples et compromettent l'idéal de collaboration autonome entre objets. De plus, le développement d'applications est complexifié, nécessitant des développeurs des connaissances spécifiques sur le fonctionnement de chaque objet. (9)

L'architecture système hétérogène (HSA) Elle permet d'accéder à une bande passante élevée vers la mémoire et offre des performances applicatives élevées tout en consommant peu d'énergie. HSA définit des interfaces pour le calcul parallèle utilisant des CPU, des GPU et d'autres dispositifs programmables, ainsi que le support d'un ensemble diversifié de langages de programmation de haut niveau. Cela crée la prochaine base d'informations à usage général, plaçant les CPU, les GPU et autres accélérateurs sur un pied d'égalité sur les plateformes informatiques, où les applications peuvent circuler entre les types de processeurs en fonction de leur charge de travail. L'objectif de HSA est de permettre aux développeurs de travailler de manière hétérogène en définissant et en promouvant une approche ouverte basée sur les normes de l'industrie. Pour faciliter la programmabilité du calcul, il est possible de tirer parti des langages de programmation existants en proposant une spécification matérielle commune et un large écosystème de support, ce qui permet aux développeurs de logiciels de fournir plus facilement des applications innovantes qui peuvent tirer pleinement parti des processeurs modernes.

Dans le passé, l'informatique hétérogène signifiait que différents ISA devaient être gérés différemment, tandis que dans un exemple moderne, les systèmes a architecture de système hétérogène (HSA) éliminent la différence (pour l'utilisateur) tout en utilisant plusieurs types de processeurs (généralement des CPU et des GPU), généralement sur le même circuit intègre, pour offrir le meilleur des deux mondes : le traitement GPU général (en plus des capacités de rendu graphique 3D bien connues du GPU, il peut également effectuer des calculs mathématiquement intensifs sur de très grands ensembles de données), tandis que les processeurs peuvent exécuter le système d'exploitation et effectuer des tâches sérielles traditionnelle. Les systèmes hétérogènes nous permettent de cibler notre programmation sur l'environnement approprie. La programmable des FPGA doit s'améliorer s'ils doivent faire partie de l'informatique grand public (47).

Les réseaux ad hoc Les réseaux ad hoc hétérogènes (HANET) jouent un rôle essentiel dans le domaine de l'Internet des objets, qui est devenu une tendance inévitable dans les recherches et les applications futures. Au cours des dernières années, les réseaux ad hoc ont été largement utilisés dans de nombreux domaines tels que la surveillance de l'environnement, le contrôle des armes, le transport intelligent, les villes intelligentes, et bien d'autres. Les HANET comprennent des réseaux de capteurs sans fil, des réseaux ad hoc intelligents, des réseaux de fidélité sans fil, des réseaux de télécommunication, des réseaux ad hoc véhiculaires,

etc (47).

Ces réseaux intègrent des informations numériques et des objets physiques grâce à des méthodes de communication appropriées, ce qui ouvre la voie à de nouvelles applications et de nouveaux services. Les différentes applications utilisent des structures de réseau indépendantes, formant ainsi une plateforme de réseau hétérogène qui complexifie les opérations de communication entre eux.

Le réseau ad hoc est une sorte de peer to peer réseaux et chaque nœud a des fonctions de collecte de données, de stockage, traitement et transmission. C'est la solution économique pour la communication a courte portée dans certains scenarios particuliers, tels que comme champ de bataille, sauvetage en cas de catastrophe, détection de l'environnement, etc. Afin d'améliorer la qualité de service (QoS) entre les différentes unités de réseau hétérogènes,les HANET deviennent le centre de recherche ces dernières années .
Les HANET consistent généralement en des réseaux de capteurs sans fil (WSN), réseaux ad hoc intelligents, réseaux de fidélité sans fil, réseaux de télécommunication , réseaux ad hoc véhiculaires (VANET), etc.

Les unités de réseau hétérogènes sont accessibles et interconnectées via de nœuds passerelles. Les WSN comprennent un grand nombre de nœuds de capteurs specialisés, qui peuvent mettre en place dynamiquement un réseau de communication auto-organisé. Les utilisateurs peuvent accéder a les données de détection qui contiennent des informations précieuses et fiables collectées par nœuds capteurs. Les réseaux intelligents ad hoc sont largement appliqués entre différents dispositifs de communication temporaire. Avec l'avancement de l'Internet des objets, les réseaux hétérogènes de capteurs sans fil ont connu une croissance rapide. Ces réseaux comprennent deux types de nœuds : les nœuds réguliers et les super nœuds. Bien que ces nœuds aient des chances égales d'accéder au réseau, ils ont des fonctions différentes. Les nœuds réguliers sont responsables de la surveillance de l'environnement, de l'envoi et de la transmission des données de détection aux super nœuds. Les super nœuds collectent principalement les données provenant des nœuds réguliers et connectent d'autres types de réseaux via des nœuds passerelles.

Les données de détection peuvent être envoyées vers un centre de données basé sur le cloud, où elles sont traitées et gèrent le réseau.

2.5 Les diverses méthodes de vérification du système

2.5.1 Analyse statique(static analysis)

L'analyse statique correspond à l'ensemble des techniques qui permettent de déduire algorithmiquement des propriétés sur le comportement d'un logiciel à partir de l'analyse de son code source et/ou de son code assembleur. Typiquement, on a recours à l'analyse statique lors de la compilation afin de détecter des bogues usuels ou afin d'optimiser le code assembleur obtenu, sans altérer le comportement du programme. Le compilateur va construire un arbre de syntaxe abstraite à partir du code source qui représente toutes les exécutions possibles. Cette structure étant facilement convertible en code assembleur par la suite. C'est sur cet arbre que le compilateur va vérifier les propriétés. Et c'est aussi cet arbre qu'il va modifier en vue d'optimiser le code assembleur. En prenant soin de vérifier qu'aucune de ces modifications n'altère le comportement du programme du point de vue de l'utilisateur (le

programme d'origine et le programme optimisé doivent rester bisimilaires). L'analyse statique permet de vérifier des propriétés telles que : (26)

— La variable à est-elle réutilisée par la suite ?

— N'essaye-t-on jamais d'accéder à l'élément 'p+1' d'un tableau qui n'en contient que 'p' ?

— Les données contenues dans une variable peuvent-elles être corrompues par des accès sauvages à la mémoire ?

— ...

Cependant, cette technique a ses limites. Comme son nom l'indique, il s'agit d'analyse statique et non dynamique. C'est à dire que l'on ne peut réellement vérifier que les variables qui sont initialisées de façon statique dans le programme. Ce qui élimine d'emblée toutes les variables dont l'initialisation est dynamique. Pourquoi ne serait-il pas possible d'appliquer la même méthode aux variables dynamiques ? Tout simplement parce que l'analyse statique revient à explorer toutes les exécutions possibles et que si certaines variables sont dynamiques, le nombre des exécutions possibles est infini. Mais, peut-être existe-t-il un algorithme qui réduise le nombre de ces exécutions possibles à un nombre fini ? En fait, il a été prouve que cela n'est pas possible. C'est un résultat connu sous le nom de théorème de Rice .

Théorème 2.1 : (Rice) Toute propriété extensionnelle non triviale de programmes écrits dans un langage récursivement énumérable est indécidable. (26)

2.5.2 Preuve assistée(theorem proving)

La vérification par preuve assistée ((55),(54)) correspond à l'ensemble des techniques qui permettent de déduire, à travers l'utilisation d'un assistant de preuve ((19), (34)), des propriétés sur le comportement d'un logiciel, algorithme ou protocole à partir de l'analyse d'un modèle mathématique de celui-ci. La méthode consiste à exprimer le programme et son environnement sous la forme d'un modèle mathématique. Ce modèle est obtenu soit par un traitement automatique du code source du programme, soit par une interprétation qu'en a faite un opérateur humain lorsque le programme est trop complexe pour être traité automatiquement. Quelle que soit la méthode utilisée pour concevoir ce modèle mathématique, il est important de prouver que les propriétés que l'on veut démontrer se comportent de façon identique sur le programme et sur le modèle mathématique que l'on obtient. Il s'agit ensuite de traduire les propriétés voulues dans ce même formalisme, puis de les vérifier avec l'aide de l'assistant de preuve. On considère alors que les propriétés sont les énonces d'un théorème que l'on essaye de prouver avec le modèle mathématique du programme et celui de son environnement comme axiomatique.

Théorème 2.2 : environnement + program $\vdash$propréties

2.5.3 Vérification du modèle (model-cheking)

La méthode du model-checking (vérification du modèle) est fondée sur trois étapes : (26)

— **La phase de modélisation du système** a pour but de fournir une représentation formelle du système étudié. De nombreux modèles ont été proposés, dont la sémantique est définie sous la forme de systèmes de transitions où les nœuds représentent les états du système, tandis que les transitions décrivent les évolutions possibles d'un état à un autre. On peut citer les structures de Kripke, les automates finis, les automates temporisés, les réseaux de Pétri,. . .

— **La phase de spécification de la propriété** consiste à traduire dans un langage formel de spécification la propriété à vérifier, écrite en langage naturel. Parmi les nombreux formalismes proposés, citons les logiques temporelles de temps linéaire ou de temps arborescent (LTL,CTL,TCTL,. . .) ,...

— **La phase de vérification** applique à ces données un algorithme qui vérifie si le modèle du système satisfait ou non le modèle de sa spécification. Cet algorithme dépend de la nature des modèles choisis pour le système et la propriété.

Parmi les modèles possibles pour décrire un système et une propriété donnés, le choix est souvent un compromis entre l'expressivité et la facilité d'analyse (la complexité). Les travaux présentés dans cette thèse s'inscrivent dans le cadre des techniques de model checking "temporisé", qui concernent plus particulièrement les automates temporisés comme formalisme de modélisation .

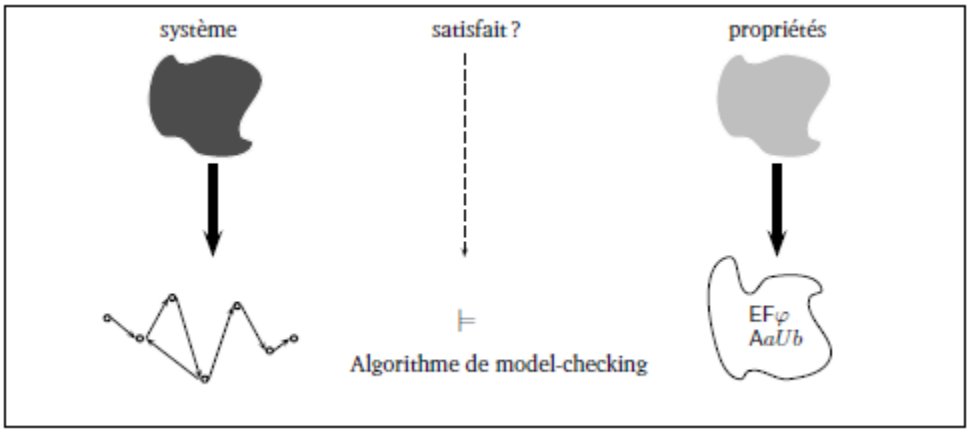

FIGURE 2.5 – Présentation du model-cheking

2.6 Les avantages de la vérification formelle

Les avantages de la vérification formelle par rapport aux autres méthodes de vérification sont les suivants :

— La vérification formelle fournit une preuve mathématiquement rigoureuse de l'exactitude qui est plus fiable que les tests ou les méthodes basées sur la simulation(Kesraoui).

— La vérification formelle peut détecter toutes les erreurs possibles, alors que les essais ne peuvent détecter que les erreurs qui se produisent pendant les cas d'essai.

— La vérification formelle peut être utilisée pour vérifier des systèmes complexes comportant de nombreux composants en interaction, qu'il peut être difficile de tester ou de simuler (49).

— La vérification formelle peut aider à identifier les défauts de conception dès le début du processus de développement, ce qui permet d'économiser du temps et de l'argent en évitant des retouches coûteuses par la suite (23).

— La vérification formelle peut fournir un niveau élevé de confiance dans la précision des systèmes critiques pour la sécurité, tels que ceux utilisés dans l'aérospatiale ou les applications médicales.

Dans l'ensemble, la vérification formelle est un outil puissant pour garantir la précision et la fiabilité des systèmes de sécurité. Bien qu'elle puisse être plus longue et plus complexe que d'autres méthodes de vérification, les avantages de la vérification formelle en font un élément important du processus de développement des systèmes de sécurité critiques.

2.7 Rapport de stage

La région saharienne de l'Algérie est reconnue pour son potentiel en hydrocarbures, notamment en gaz et en pétrole, ce qui est d'une importance capitale pour le pays. Après la nationalisation des hydrocarbures, SONATRACH et d'autres sociétés étrangères ont collaboré pour entreprendre l'extraction et l'exploitation de ces richesses. La complexité de l'extraction des hydrocarbures a nécessité l'utilisation d'installations et de dispositifs qui évoluent en synchronisation avec les avancées technologiques afin d'optimiser le temps et le coût des processus, tout en réduisant considérablement les risques associés à la nature des hydrocarbures.

Le stage que nous avons effectué avait pour objectif d'explorer un système d'une importance critique et d'étudier en détail le fonctionnement d'un système critique par exemple : SCADA. Cette expérience a été l'occasion de se familiariser avec les enjeux liés à la gestion et à la surveillance de ce type de système, ainsi que de découvrir les principes de base de la collecte et de la transmission des données .

Ce stage a eu lieu de 27/03/2022 jusqu'au 02/04/2022, dont le programme est le suivant :

— introduction sur le sonatrch ;

— des informations générale sur l'entreprise ;

— visite du département informatique ;

— visite du département de télécommunication.

2.7.1 La société Sonatrach

SONATRACH est l'abréviation de la "Société nationale pour la recherche, la production, le transport, la transformation et la commercialisation des hydrocarbures". Il s'agit d'une entreprise pétrolière et gazière algérienne fondée le 31 décembre 1963 dans le but principal de répondre à l'exploitation de la rente pétrolière, perçue dès le départ comme un moteur

essentiel pour le développement du pays. Au fil des années, SONATRACH est devenue la première entreprise en Afrique, contribuant ainsi au développement économique et social du pays. En 1967, elle a adopté son logo reconnaissable avec ses couleurs orange et noir.

2.7.2 Description générale sur l'usine

SONATRACH branche transport par canalisations (TRC) a pour objectif l'acheminement des hydrocarbures liquides et gazeux des champs de production vers les centres de consommation, de transformation, les frontières et les ports pétroliers pour l'exportation. La direction régionale de Skikda est L'une des sept (07) régions de la division exploitation (EXL) de la branche transport par canalisations de Sonatrach. Son siège, implanté au sein de la zone industrielle des hydrocarbures, est situé à l'Est de la ville de Skikda.

2.7.3 Le système SCADA

Le système SCADA est devenu populaire dans les années 1960, avec l'augmentation de la nécessité de surveiller et de contrôler l'équipement. Les premiers systèmes intégrés utilisant des ordinateurs centraux étaient chers car ils ont été opérés et contrôlés manuellement. Mais les récents progrès technologiques ont avancé, les systèmes SCADA automatisés avec une efficacité maximale à un coût réduit, selon les exigences alarmantes de la société.

1. **Définition :**
   Le système SCADA est un acronyme pour Supervisory Control and Data Acquisition, qui permet la centralisation et la présentation semi-graphique des données sur des postes de pilotage. Ce système collecte des données provenant de différents appareils d'installations, les transmet à un ordinateur central, et le supervise pour assurer un contrôle efficace. Les systèmes SCADA sont utilisés dans une variété d'industries pour surveiller des données telles que les flux, les courants, les tensions, les pressions, les températures, les niveaux d'eau, etc. Lorsque des conditions anormales sont détectées, des alarmes sont déclenchées pour alerter les opérateurs à travers le HMI. Les industries de transformation, le pétrole et le gaz, la production d'électricité, la distribution et les services publics, l'eau et le contrôle des déchets, l'agriculture/l'irrigation, la fabrication, les systèmes de transport, etc., sont les applications les plus courantes pour les systèmes SCADA.

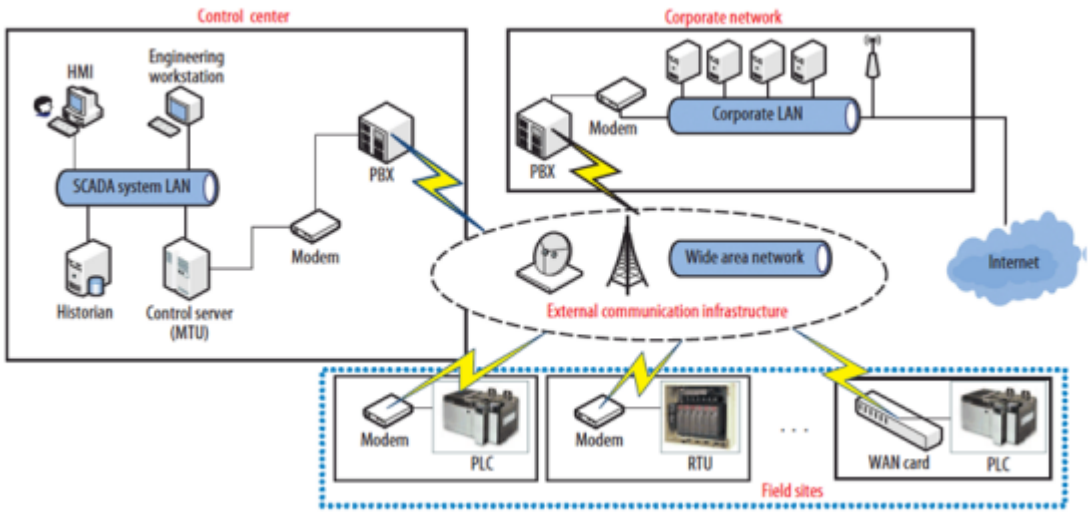

FIGURE 2.6 – Système SCADA

2. **Fonctionnement :**
   Le système SCADA n'a pas vocation à se substituer entièrement à l'homme : le pilotage et la prise de décision restent dévolus à l'opérateur. C'est pourquoi les logiciels SCADA sont fortement dédiés à la surveillance et aux alarmes. Imaginons par exemple un API pilotant l'écoulement de l'eau de refroidissement d'un processus industriel. Le système SCADA permet à un opérateur :

   — De modifier la consigne d'écoulement (litres par seconde) ;

   — Enregistre et affiche l'évolution des mesures ;

   — Détecte et affiche des conditions d'alarme (perte d'écoulement, etc.).

3. **Notions de base :**
   Le système SCADA se compose des différents blocs, à savoir l'interface homme-machine (HMI), le système de surveillance, des unités terminales à distance, les automates, les infrastructures de communication et de programmation SCADA tel que :

   Interface d'E/S
   L'interface d'E/S reçoivent les signaux d'entrées (transmetteur, interrupteur,) et les transforment en numérique pour être adaptés à l'API. Ces signaux sont envoyés au CPU via un bus d'E/S pour être traités. Le CPU fait toute sorte de traitement en temps réel (logique, séquence, calcule, ...) et élabore les signaux de sortie qui retraversent l'interface d'E/S via le même bus pour être adaptés à l'actionneur correspondant (vanne, électrovanne, ...).

   Interface homme-machine (IHM)
   Est un dispositif d'entrée-sortie qui contient les données de processus pour être contrôlé par un opérateur humain. Il est utilisé par un lien vers les programmes et les bases de données de logiciels du système SCADA pour fournir les informations de gestion, y

compris les procédures du programmées de maintenance, schémas détaillés, des informations logistiques, des tendances et des maintenance, schémas détaillés, des informations logistiques, des tendances et des données de diagnostic pour un capteur ou d'une machine spécifique. Systèmes IHM facilitent le personnel d'exploitation pour voir les informations sous forme graphique.

Système de surveillance
Système de surveillance est utilisé en tant que serveur de communication entre l'équipement du système SCADA tels que RTU, automates et capteurs, etc., et le logiciel IHM utilisé dans les postes de travail de la salle de contrôle. Station maître ou d'une station de surveillance comprend un seul PC dans les systèmes SCADA plus petits et, en cas de grands systèmes SCADA, système de contrôle comprend distribuer des applications logicielles, des sites de reprise après sinistre et de multiples serveurs.

Remote Terminal Unit (RTU)
Les objets physiques dans les systèmes SCADA sont interfacés avec les appareils électroniques contrôlés par microprocesseur appelé comme Remote Terminal Unit (RTU). Ces unités sont utilisées pour transmettre des données de télémesure du système de surveillance et de recevoir les messages du système maître pour contrôler les objets connectés. Par conséquent, ceux-ci sont également appelés unités de télémétrie à distance.

Automate Programmable
Dans les systèmes SCADA, les automates sont connectés aux capteurs pour collecter les signaux de sortie des capteurs afin de convertir les signaux de capteur en données numériques. Automates sont utilisés à la place du RTU en raison des avantages des automates tels que la flexibilité, la configuration, polyvalent et abordable par rapport à RTU.

Infrastructure Communication
En général, la combinaison de la radio et de connexions câblées directes est utilisé pour les systèmes SCADA, mais en cas de grands systèmes comme les centrales électriques et les chemins de fer SONET / SDH sont fréquemment utilisés. Parmi les protocoles SCADA très compacts utilisés dans les systèmes SCADA, quelques protocoles de communication, qui sont normalisés et reconnus par les fournisseurs SCADA, envoyer des informations uniquement lorsque la station de surveillance interroge le RTU ou API.

Programmation SCADA
Programmation SCADA dans un maître ou IHM est utilisé pour créer des cartes et des diagrammes qui donneront une importante information de la situation en cas d'échec de l'événement ou de l'échec du processus. Interfaces standard sont utilisées pour la programmation de la plupart des systèmes SCADA commerciaux. La programmation de SCADA peut être faite en utilisant un langage de programmation.

4. **Les points critique du système SCADA :**
   Le système SCADA est souvent utilisé pour contrôler des processus industriels critiques, tels que les centrales électriques, les installations de traitement de l'eau, les pipelines de gaz et de pétrole, etc. Par conséquent, un dysfonctionnement ou une défaillance du système SCADA peut potentiellement causer des dégâts humains, environnementaux et économiques importants. Voici quelques exemples de points critiques qui peuvent conduire à des dégâts humains :
   (a) Les défauts de conception : les erreurs de conception du système SCADA, telles que la configuration incorrecte des seuils d'alarme ou des seuils de sécurité, peuvent entraîner des défaillances du système et des situations dangereuses.
   (b) Les erreurs humaines : les erreurs commises par les opérateurs ou les techniciens du système SCADA, telles que l'oubli de vérifier une alarme ou de suivre les procédures de sécurité, peuvent causer des défaillances du système et des accidents.
   (c) Les cyberattaques : les attaques informatiques sur le système SCADA peuvent entraîner la prise de contrôle du système, la falsification de données, la modification de l'état des équipements et des processus, etc. Ces attaques peuvent avoir des conséquences graves sur la sécurité humaine et environnementale.
   (d) Les défaillances matérielles : les défaillances matérielles, telles que les pannes de l'alimentation électrique, les pannes de la communication ou les défauts des équipements de contrôle, peuvent également entraîner des défaillances du système et des accidents.

2.8 Conclusion

Dans ce chapitre, nous avons procédé à une introduction des concepts de base des systèmes temps réel, des systèmes critiques et des systèmes hétérogènes. Nous avons commencé par définir ces différents types de systèmes, en fournissant des exemples concrets pour chacun d'entre eux. Ensuite, nous avons présenté diverses méthodes de vérification utilisées pour ces systèmes.

Dans le prochain chapitre, nous aborderons spécifiquement les formalismes de vérification formelle appliqués aux systèmes temporisés critiques.

Chapitre 3 Les automates temporisés

Sommaire

3.1 Introduction : 25
3.2 Notions préliminaires : 26
3.3 Langages temporisés 27
3.3.1 Notions : 27
3.4 Automates temporisés 27
3.4.1 Définition 01 : 27
3.4.2 Définition 02 : 28
3.5 Sémantique d'un automate temporisé 28
3.6 Propriétés des automates temporisés 29
3.7 Problème du vide 30
3.7.1 Description du problème 30
3.7.2 Décidabilité des automates temporisés 31
3.7.3 Les régions 32
3.8 Les zones 35
3.8.1 Les DBMs 36
3.8.2 Graphe des zones 38
3.8.3 Approximations et améliorations 39
3.9 L'approche d'analyse en arrière 39
3.10 L'approche d'analyse en avant 40
3.11 Outils de vérification : 41
3.12 Conclusion 43

3.1 Introduction :

La vérification formelle des systèmes critiques a connu un succès incontestable depuis la fin des années 1970. Au cours des quinze dernières années, ces techniques de vérification ont été étendues afin de prendre en compte des notions quantitatives, permettant en particulier de spécifier le délai qui sépare différentes actions du système. Dans ce cadre, le modèle des automates temporisés (3) a été particulièrement bien étudié, et plusieurs outils de vérification ont été développés pour ces modèles. Un automate temporisé est un système de transitions disposant d'horloges, qui évoluent continûment, toutes à la même vitesse, et qui servent à spécifier quand une transition est autorisée.

Dans ce chapitre, nous allons présenter le modèle des automates temporisés, introduit par Alur et Dill.

3.2 Notions préliminaires :

Tout d'abord nous commençons par la présentation de quelques notions de base. Nous notons $\mathbb{N}$ l'ensemble des entiers naturels et $\mathbb{R}^+$ l'ensemble des réels positifs ou nuls. Dans tout le document, le domaine du temps est $\mathbb{R}^+$ (27). Soit X un ensemble d'horloges, qui prennent leurs valeurs dans $\mathbb{R}^+$. Nous supposons donner aussi un alphabet fini d'action Act.

— **Alphabet, chaine**

On appelle alphabet un ensemble fini de symbole. Une chaine sur un alphabet Σ est une séquence éventuellement vide de symboles de Σ. La séquence vide est notée ϵ(epsilon). Les autres séquences sont notées par la juxtaposition des symboles qui les composent.

— **Langage**

Un langage est un ensemble de chaines sur un alphabet.

— **Automate fini**

Un automate fini est un quintuple A=(Σ,S,σ,X,s_0,F) où :

- Σ est un ensemble fini de symboles appelé alphabet ;
- S est un ensemble fini dont les éléments sont appelés états ;
- σ est une relation de $S \times \Sigma \times S$ appelée transition ou ensemble des transitions de A ;
- s_0 est un état de S appelé état initial ;
- F est un sous-ensemble de S appelé ensemble des états finals de A.

L'ensemble des transitions σ est une relation, c'est-à-dire un ensemble de triplets. Cet ensemble est nécessairement fini puisque S et Σ sont finis. Un automate fini est fait de composantes qui sont toutes finies (Σ, S, σ, F), d'où le qualificatif de fini.

— **Représentation graphique**

Un automate fini peut être représenté graphiquement comme un graphe orienté dont les sommets sont les états et les arcs sont les transitions. Une transition (s_1, x, s_2) est représentéepar un arc reliant les sommets s_1 et s_2, étiqueté par x.

— **Chemin** Soit A= (Σ, S, σ, F) un automate. Un chemin de cet automate est une séquence $(s_0, x_0, s_1)(s_1, x_1, s_2)$ ...(s_{n-1}, x_{n-1}, s_n) de trasitions de σ telle que n $\geq$0 (Barthélemy,4 janvier 2012).

— **Une valuation des horloges**

Une valuation des horloges v est une fonction, v : X —>$\mathbb{R}^+$, qui associe chaque horloge x $\in$X sa valeur v (x).L'ensemble de toutes les valuations des horloges de X est noté V_X.

— **Ecoulement du temps**

Etant donnée une durée $d \in \mathbb{R}^+$ et v une valuation, la notation v + d désigne la valuation défini par (v + d) (x) = v (x) + d pour tout horloge x ∈X.

— **Remise à zéro des horloges**

Si $\lambda \subseteq X$, l'opération de remise a zéro de λ dans v, notée v $[\lambda := 0]$, définit une nouvelle valuation v 0 obtenue partir de v en assignant la valeur 0 a toutes les horloges de λ, lavaleur des autres horloges reste inchangée.

— **Contraintes sur les horloges**

Une contrainte d'horloge est une conjonction de contraintes atomiques qui comparent la valeur d'une contrainte x, appartenant à l'ensemble fini d'horloges, à uneconstante rationnelle. Étant donné un ensemble d'horloges X, l'ensemble des contraintes d'horloges C(X) est l'ensemble engendré par la grammaire :
$g := x \sim | \; g \wedge g \mid \neg g$. ou $x \in X, c \in \mathbb{N}$ et $\sim \in \{<, \leq, =, \geq, >\}$.

3.3 Langages temporisés

Un langage temporisé associe à chaque lettre (action) d'un mot, une date. Les mots sont donc formés d'une suite de couples comprenant une lettre choisie dans un alphabet fini Σ et un temps à valeur dans un domaine de temps T , qui sera égal à $\mathbb{N}$, $\mathbb{Q}^+$, $\mathbb{R}^+$.

3.3.1 Notions :

Soit Z un ensemble quelconque, alors Z^* (resp. Z^ϖ) est l'ensemble des séquences finies (resp. infinies) d'éléments de Z. On note $Z^\infty = Z^* \bigcup Z^\varpi$.

Une séquence temporisée sur T est une séquence croissante de T^∞ .Un mot temporisé ϖ est une séquence de couples $(a_i, t_i)_{(i>0)} \in (\Sigma \times T)^\infty$ telle que (t_i) soit une séquence temporisée.

3.4 Automates temporisés

3.4.1 Définition 01 :

Un automate temporisé est un automate à états finis comportant une représentation du temps continu par l'intermédiaire de variables à valeurs réelles, positives ou nulles,appelées horloges, qui permettent d'exprimer des contraintes temporelles.

De façon générale, un automate temporisé est représenté par un graphe où chaque sommet correspond à un état du système et les arcs aux transitions entre ces états. Les contraintes temporelles s'expriment au travers des contraintes d'horloges et peuvent porter sur les états et sur les transitions. Une contrainte d'horloge est une conjonction de contraintes atomiques qui comparent la valeur d'une contrainte x, appartenant à l'ensemble fini d'horloges, à une constante rationnelle.(35)

3.4.2 Définition 02 :

On peut définir un automate temporisé d'une manière formelle comme suit : 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 $\{G, a, R\}s'$) 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 ;
- $I : S \to C(H)$ associe un invariant à chaque localité ;
- Act est un alphabet d'actions.

Exemple :

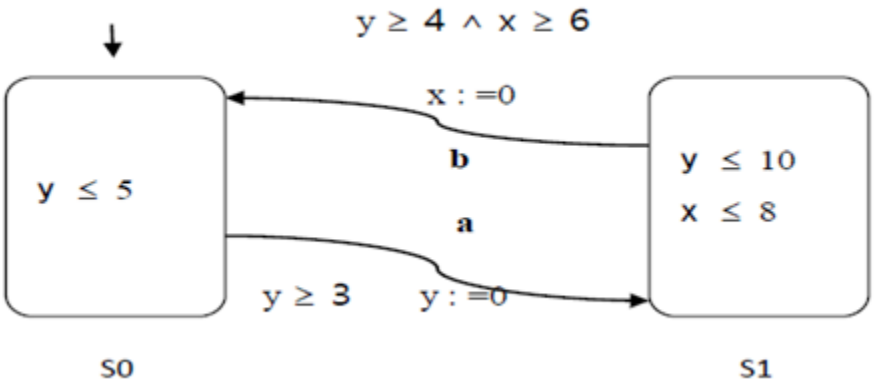

FIGURE 3.1 – Automate temporisé simple

Quelques trajectoires : Les trajectoires de l'automate de l'exemple précédent 3.1 sont :

1. $(s_0, 0, 0) \xrightarrow{3} (s_0, 3, 3) \xrightarrow{a} (s_1, 3, 0) \xrightarrow{4} (s_1, 7, 4) \xrightarrow{b} (s_0, 0, 4) \xrightarrow{1} (s_0, 1, 5) \xrightarrow{a} (s_1, 1, 0)...$
2. $(s_0, 0, 0) \xrightarrow{3.2} (s_0, 3.2, 3.2) \xrightarrow{a} (s_1, 3.2, 0) \xrightarrow{4.1} (s_1, 7.3, 4.1) \xrightarrow{b} (s_0, 0, 4.1) \xrightarrow{0.1} (s_0, 0.1, 4.2) \xrightarrow{a} (s_1, 0.1, 0)$ . . .

3.5 Sémantique d'un automate temporisé

La sémantique d'un automate temporisé est définie par un système de transitions temporisé (STT). À partir d'une configuration donnée de l'automate temporisé, deux types de transitions sont possibles : une transition d'action si la valuation courante vérifie la garde, ou une transition de durée qui consiste à rester dans le même état de contrôle et à incrémenter les valeurs des horloges uniformément en respectant l'invariant.(48)

Informellement : Le système part de la configuration initiale (état de contrôle s_0 et toutes les horloges à zéro),puis effectue deux types de transitions : Les transitions d'action si la valeur courante des horloges le permet (ce type de transition s'effectue de manière instantanée et certaines horloges peuvent être remises à zéro) et les transitions de temps qui augmentent toutes les horloges d'une même durée (les horloges sont synchrones) en respectant l'invariant associé à l'état courant.

3.6 Propriétés des automates temporisés

Le modèle que nous venons de définir a certaines propriétés intéressantes que nous résumons brièvement ci-dessous.

Problème du vide :
Pour tout automate temporisé A, le problème du vide consiste à décider de la valeur de vérité de lassertion $L(A) = \emptyset$.
Ce problème a été montré Pspace-complet pour la classe des automates temporisés classiques (38).

Universalité :
Ce problème est le dual du problème du vide. Pour tout automate temporisé A, le problème de l'universalité consiste à décider la valeur de vérité de l'assertion $(A) = (\Sigma \times T)^{\infty}$.
Ce problème a été montré indécidable pour la classe des automates temporisés classiques.(38)

Inclusion de langage :
Pour tout couple d'automates temporisés A et B, le problème de l'inclusion des langages est de décider la valeur de vérité de l'assertion $(A) \subseteq L(B)$ Ce problème a été montré indécidable pour la classe des automates temporisés classiques (38) (On se ramène au problème de l'universalité),

Équivalence de langage :
Pour tout couple d'automates temporisés A et B et leurs Langages associés (A) et $L(B)$, le problème de l'équivalence des langages est de décider la valeur de vérité de l'assertion $L(A) = L(B)$.
Ce problème a été montré indécidable pour la classe des automates temporisés classiques (38) (on se ramène à un problème de double inclusion des langages),

Traces non temporisées :
Pour tout automate temporisé A et un mot σ tel que $\sigma = \sigma_1\sigma_2\sigma_3\ldots \in \Sigma^*$. Le problème des traces non temporisées est de vérifier que l'on a $\sigma \in$Untimed(A). Autrement dit, qu'il existe une exécution temporisée r sur A telle que :

$$r = s_0 \frac{g_1,\sigma_1,R_1}{t_1} s_1 \frac{g_2,\sigma_2,R_2}{t_2} s_2 \frac{g_3,\sigma_3,R_3}{t_3} s_3 \ldots$$

Ce problème à été monté NP-complet pour la classe des automates temporisés classiques (26) et Pspace-complet pour la classe des automates temporisés avec $\varepsilon-$ transitions .

Traces temporisées :
Pour tout automate temporisé A, un mot $\sigma = \sigma_1\sigma_2\sigma_3\ldots \in \Sigma^*$ et une séquence temporisée $\tau = t_1t_2\ldots$ Le problème des traces temporisées est de vérifier que l'on a $(\sigma, t) \in L(A)$. Autrement dit, qu'il existe une exécution temporisée r sur A telle que :

$$r = s_0 \frac{g_1,\sigma_1,R_1}{t_1} s_1 \frac{g_2,\sigma_2,R_2}{t_2} s_2 \frac{g_3,\sigma_3,R_3}{t_3} s_3 ...$$

Ce problème à été montré NP-complet pour la classe des automates temporisés classiques (26) et Pspace-complet pour la classe des automates temporises contenant des $\varepsilon-$ transitions (26).

Génération d'estampillage temporèl :
Pour tout automate temporisé A, et un chemin p sur A, tel que :

$$s_0 \xrightarrow{g_1,\sigma_1,R_1} s_1 \xrightarrow{g_2,\sigma_2,R_2} s_2 \xrightarrow{g_3,\sigma_3,R_3} s_3 \xrightarrow{g_n,\sigma_n,R_n} s_n$$

avec $s_0 \in I$,et $\forall i \leq n, (s_{i-1}, g_i, \sigma_i, R_i, s_i) \in T$
Le problème de la génération d'estampillage temporel est de vérifier s'il existe une exécution temporisée r sur A telle que :

$$r = s_0 \frac{g_1,\sigma_1,R_1}{t_1} s_1 \frac{g_2,\sigma_2,R_2}{t_2} s_2 \frac{g_3,\sigma_3,R_3}{t_3} s_3 ... \xrightarrow{g_n,\sigma_n,R_n} s_n$$

Ce problème à été montré de complexité $O(n, m^2)$, avec n la longueur du mot et m le nombre des horloges, pour la classe des automates temporisés classiques et celle des automates temporisés avec ε-transitions. (26) (4)

3.7 Problème du vide

3.7.1 Description du problème

Une fois que les systèmes sont modélisés, le problème de tester le vide du langage accepté par le modèle est fondamental. En effet, le problème de l'accessibilité d'un état (c'est-à-dire tester s'il existe une exécution dans le modèle qui permet d'atteindre un état donné) est strictement équivalent à la non-vacuité du langage accepté par ce même modèle en prenant comme état final l'état dont on cherche à tester l'accessibilité. Par exemple, pour assurer une propriété d'exclusion mutuelle, il faut tester que deux processus ne peuvent pas aller simultanément dans leur section critique, ce qui revient à tester l'accessibilité de l'un des états du système où deux processus sont simultanément en section critique. Le problème de tester le vide d'un langage accepté par un automate est appelé de manière plus synthétique le problème du vide. Nous disons alors qu'une classe de modèles est décidable si le problème du vide est décidable pour chacun de ces modèles.

Nous allons maintenant présenter l'étude de la décidabilité des automates temporisés faite par Alur et Dill dans ((2),(3)).

3.7.2 Décidabilité des automates temporisés

Le résultat suivant est fondamental :

Théorème 3.1 :

La classe des automates temporisés est décidable ((2),(3)).

Étant donné un mot temporisé de $(\Sigma \times T)^\infty$, en effaçant toutes les dates, nous obtenons un mot de Σ^∞. De cette manière , nous pouvons associer à chaque langage temporisé L un langage classique sur l'alphabet Σ . Ce langage est noté Untime (L) et est défini formellement par :

$$\text{Untime}(L) = \{a_1...a_n \mid \exists t_1...t_n \textbf{ t.q. } (a_1,t_1)...(a_n,t_n) \in L\}$$

La preuve du théorème repose alors sur la construction, à partir d'un automate temporisé A, d'un automate classique β qui reconnaît le Untime du langage accepté par l'automate A. Or, comme pour tout langage temporisé nous avons bien évidemment :

$$L = \emptyset \text{ est équivalent à } Unitime(L) = \emptyset$$

nous obtenons que :

«le langage accepté par A est vide si et seulement si le langage accepté par β est vide»

Ainsi, comme le test du vide est décidable pour les automates finis (ou de Büchi) (32), il est aussi décidable pour les automates temporisés.
Nous allons décrire le principe de la preuve de ce théorème. Tout d'abord, grâce au lemme suivant, nous pouvons nous restreindre aux constantes entières :

lemme 3.1 : Soit A un automate temporisé.Si $\lambda \in Q^+$, nous définissons l'automate λA comme étant le même automate que A sauf que les constantes apparaissant sur les transitions (au niveau des contraintes) sont multipliées par λ . Soit $\upsilon = (a_1,t_1)...(a_n,t_n)...$ un mot temporisé. Nous avons alors que :
$\upsilon \in L(A)$ est équivalent à $\lambda.\upsilon \in L(\lambda A)$
Si $\lambda \cdot \upsilon$ représente le mot temporisé $(a_1,\lambda t_1)$ ... $(a_n,\lambda t_n)$ ... (toutes les dates sont multipliées par λ).
Ainsi, pour tester le vide d'un automate temporisé, nous pouvons multiplier toutes les constantes apparaissant sur les transitions, sans changer le résultat du test du vide. En particulier, si nous prenons comme facteur multiplicatif le « plus petit commun multiple » (i.e. le ppcm) des dénominateurs de toutes les constantes (qui sont par hypothèse toutes rationnelles) apparaissant dans l'automate.
Nous pouvons aussi supposer que les contraintes utilisées sont non diagonales. En effet, nous avons le résultat suivant :

lemme 3.2 : Soit A un automate temporisé. Il existe un automate temporisé A' avec uniquement des contraintes non diagonales tel que $L(A) = L(A')$.
Une preuve complète de ce lemme peut par exemple être trouvée dans (8).

3.7.3 Les régions

Alur et Dill ont démontré (2) que le problème du vide est décidable pour les automates temporisés. Leur preuve est basée sur la construction de l'automate des régions. Celui-ci consiste en fait en un graphe fini simulant l'automate temporisé basé sur une relation d'équivalence d'indice fini définie sur les valuations d'horloges. Il s'agit d'identifier deux valuations v et v' telles que le comportement de l'automate temporisé soit le même à partir des configurations $\langle s, v\rangle$ et $\langle s, v'\rangle$ pour tout état s de l'automate. Le graphe quotient de cette relation d'équivalence est appelé graphe des régions. C'est simplement une partition finie de l'ensemble T^X des valuations d'horloges possibles, pour laquelle chaque élément est dénommé "région" (53), Et c'est expliqué dans la suite.

Le graphe des régions

Soit $A = (S, X, \Sigma, s_0, F, R, T)$ un automate temporisé classique avec des constantes entières et des contraintes non diagonales. Pour toute horloge $x \in X$, nous notons max la plus grande constante c telle qu'une contrainte d'horloges $x \sim c$ apparaisse dans A.
L'équivalence des régions est définie sur T^X par :

$v \equiv v'$ est équivalent à :

— $Ent(v(x)) = Ent(v'(x))$ ou $(v(x) > max_x$ et $v'(x) > max_x)$

— si $(v(x) \leq max_x$ et $v(y) \leq max_y)$

— alors $[frac(v(x)) < frac(v(y))$ est équivalent à $frac(v'(x)) < frac(v'(y))]$

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

1. pour tout contrainte g de A,$v \models g$ est équivalent à $v' \models g$
2. $\forall t \in T, \exists t' \in T$ tel que $v + t \equiv v' + t'$

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 $T^X/_{\equiv}$ est appelée une région.

Exemple 1 :
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 figure 3.2.

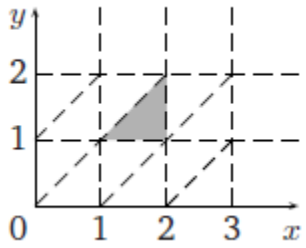

FIGURE 3.2 – Représentation un région

La région grisée est définie par la contrainte : $1 < x < 2 \bigwedge 1 < y < 2 \bigwedge x > y$

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 :
$R' \in Succ(R)$ est équivalent à $\exists v \in R. \exists t \in T$ tel que $v + t \in R'$

Alors 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.

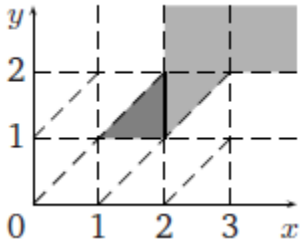

FIGURE 3.3 – Représentation des autres régions

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

$$S' = S \times T^X / \equiv, s'_0 = (s_0, 0), F' = F \times T^X / \equiv, R' = R \times T^X / \equiv,$$
$$T' = \left\{ (s, R) \xrightarrow{a} (s', R') \middle| \begin{array}{l} \exists s \xrightarrow{g,a,C} s' \in T \text{ et } \exists R" \in Succ(R) \\ \\ tel\ que\ R \subseteq g\ et\ R' = R"|C \leftarrow 0| \end{array} \right\}$$

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

Exemple 2 :

Nous traitons un exemple de construction de l'automate des régions. Considérons donc l'automate A dessiné sur la figure 3.4.

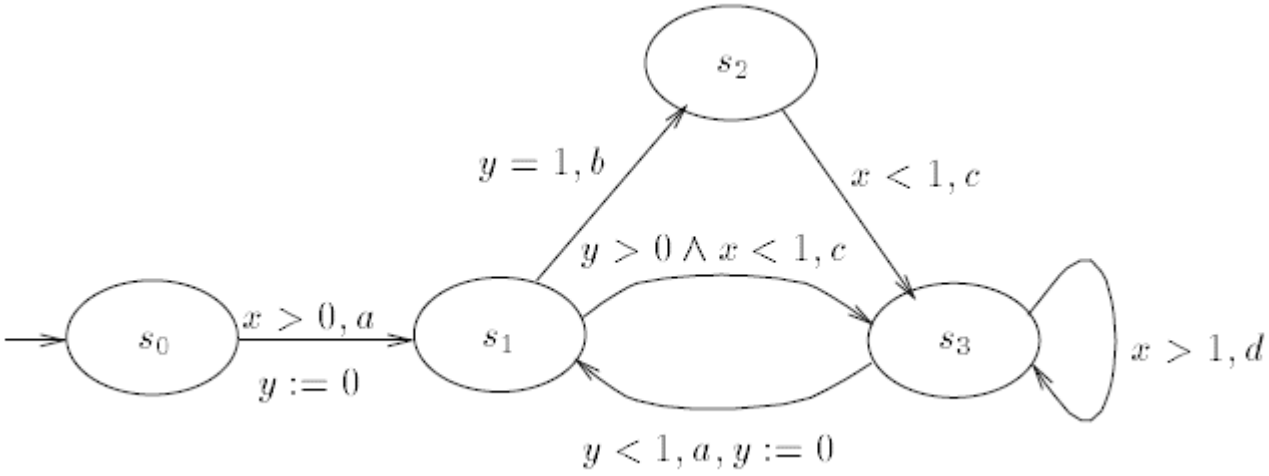

FIGURE 3.4 – Automate temporisé A

L'automate des régions que nous obtenons à partir de A en prenant comme constantes maximales 1 pour les deux horloges x et y est dessiné sur la figure 3.5.

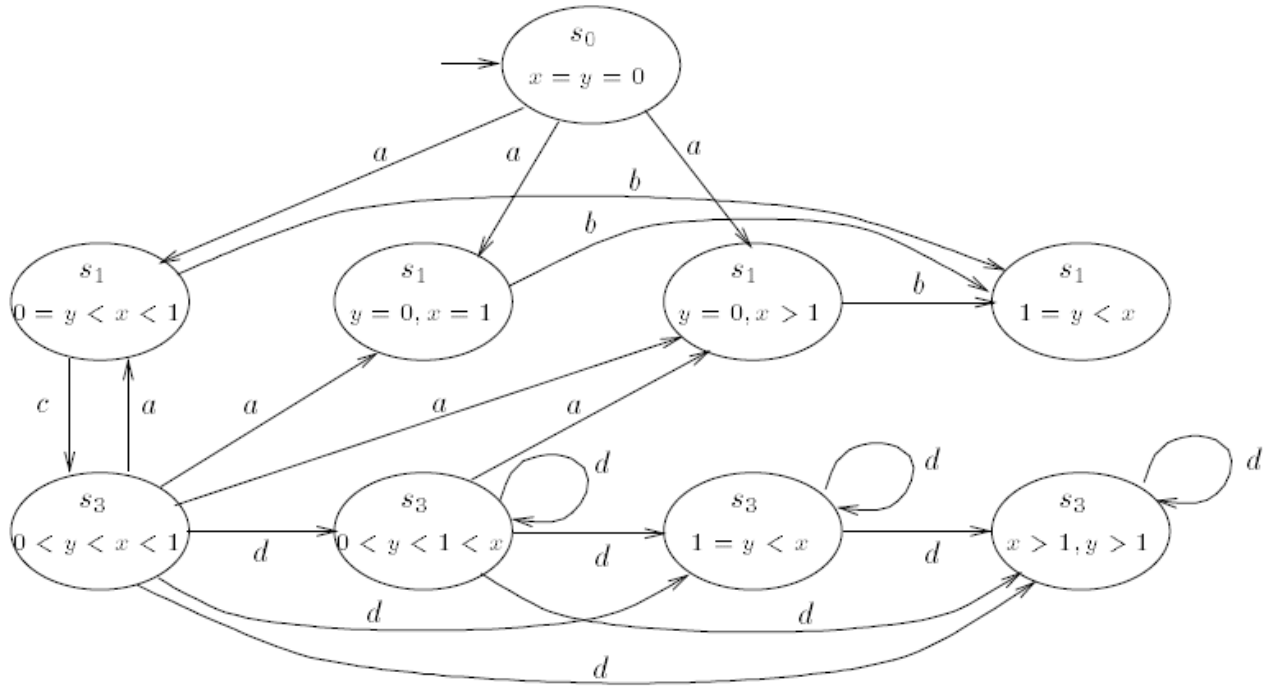

FIGURE 3.5 – Automate des régions associé à A

L'automate des régions vérifie la propriété suivante :

Propriété 3.1 :
Le langage accepté par β correspond au Untime du langage accepté par A.

Nous avons alors le théorème plus précis suivant :

Théorème 3.2 : Tester le vide d'un langage accepté par un automate temporisé est un problème PSPACE-complet. (3)

3.8 Les zones

Les régions, introduites dans la sous-section 3.7.3, constituent une représentation symbolique. Cependant, le nombre de régions est exponentiel dans la taille de l'automate. Pour vérifier ce modèle par model-checking, il est nécessaire de maitriser sa taille même s'il est un modèle fini. La plus communement utilisée est la représentation symbolique appelée zone qui est utilisée dans la majorité des algorithmes développés pour les systèmes temporisés et qui est plus compacte que les régions (22).

Une zone d'horloges (10) est un espace convexe défini par une conjonction d'inégalités qui comparent soit une valeur d'horloge a un nombre entier ou la différence entre deux valeurs d'horloge a un nombre entier. Elle est sous la forme : $\bigwedge_{0 \leq i \neq j \leq nb}$

$x_i - x_j \prec n_{ij}$ ou $\prec \in \{<, \leq\}$, nb est le nombre d'horloges, $n_{ij} \in N$ et x_i sont des horloges. On introduit une horloge spéciale x_0 qui est toujours égale a zéro.

Exemple

La figure ci-dessous 3.6 donne un exemple d'une représentation graphique de la zone $x - x_0 > 1 \wedge y \leq 3 \wedge x - y < 2$ a deux horloges x et y avec x_0 qui est une horloge constante à valeur 0.

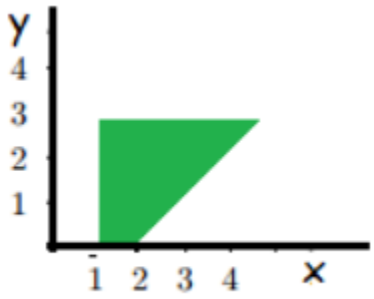

FIGURE 3.6 – Représentation graphique de zone

3.8.1 Les DBMs

Une DBM (pour Difference Bounded Matrix) (24) est une structure de données qui permet de représenter des systèmes de contraintes de différences (17), mais elle a un intérêt particulier pour la vérification des systèmes temporisés car elle permet de représenter les zones. Chaque zone d'horloges peut être représentée par une DBM. Les DBM peuvent être utilisées pour évaluer la satisfiabilité d'une zone donnée.

Une DBM d'une zone d'horloges z est une structure de données permettant de représenter une zone par une matrice carrée M de dimension $nb \times nb$ ou nb est le nombre d'horloges de z, incluant une horloge spéciale x_0 qui est toujours égale à zéro.
Un élément $M_{i,j}$ est sous la forme $(n_{ij}, \prec)$ou $n_{i,j} \in N$ et $\prec \in \{<, \leq\}$.
Le tableau 3.1 montre une DBM M qui représente la zone définie ainsi : $1 \leq x_1 \leq 4 \wedge 1 \leq x_2 \leq 3 \wedge x_1 - x_2 \leq 1$. Par exemple $M_{2,1} = x_2 - x_1 = (2, \leq)$. La valeur de 2 est obtenue : on a $1 \leq x_1 \leq 4$ et $1 \leq x_2 \leq 3$ donc $1 - 4 \leq x_2 - x_1 \leq 3 - 1$.

$x_i - x_j$	x_0	x_1	x_2
x_0	$(0, \leq)$	$(-1, \leq)$	$(-1, \leq)$
x_1	$(4, \leq)$	$(0, \leq)$	$(1, \leq)$
x_2	$(3, \leq)$	$(2, \leq)$	$(0, \leq)$

TABLEAU 3.1 – DBM de la contrainte $1 \leq x_1 \leq 4 \wedge 1 \leq x_2 \leq 3 \wedge x_1 - x_2 \leq 1$

Les DBM peuvent être utilisées aussi pour évaluer la satisfiabilité d'une zone. Une DBM est insatisfaisante si et seulement si la condition suivante est vérifiée :

$$\sum_{i=0}^{nb-1}(M_{i,i+1}) + M_{nb,1} < (0, \leq)$$

L'addition de deux éléments $M_{i,j} = (n, e)$ et $M_{i_1,j_1} = (n_1, e_1)$ est égale à $(n+n_1, min(e, e_1))$ ou $<$ est représente par 0 et $\leq$ par 1.

Les opérations d'intersection de deux zones, de remise a zéro d'une zone, de calcul du successeur d'une zone sont définies sur la structure de données DBM.

Intersection de deux zones $z \cap z'$: Soit M et M' deux DBM qui représentent les zones z et z' et$M'' = M \cap M'$, l'intersection de M et M' est une DBM M" telle que $M''_{i,j} = min(M_{i,j}, M'_{i,j})$.

Remise à zéro d'une zone : La remise à zéro d'un ensemble d'horloges X' dans une zone z est égale $z' = z[X' := 0]$. La zone z est représentée par la DBM M. La zone z' est représentée par la DBM M' telle que :

- $M'_{i,0} = (0, \leq)$ pour tout $x_i \in X'$
- $M'_{0,i} = (0, \leq)$ pour tout $x_i \in X'$
- $M'_{i,j} = M'_{0,j}$ pour tout $x_i \in X'$ et $x_j \notin X'$

— $M'_{j,i} = M'_{j,0}$pour tout $x_i \in X'$ et $x_j \notin X'$

Successeur d'une zone $\overrightarrow{z}$: la zone $z' = (z \cap g)[X' := 0]$ est la zone suivante de la zone z en franchissant une transition (l, a, g, X', l').

Exemple

On prend comme exemple la zone z définie ainsi :$1 \leq x_1 \leq 4 \wedge 1 \leq x_2 \leq 3 \wedge x_1 - x2 \leq 1$ qui est représentée par une DBM M_g dans le tableau 3.1. La zone z' est le successeur de la zone z en franchissant la transition$(l, a, x_1 \leq 4 \wedge x_2 \leq 2, \{x_2\}, l')$. On obtient la zone $z' = (z \cap g)[X' := 0]$ telle que $g = x_1 \leq 4 \wedge x_2 \leq 2$. La zone $0 \leq x_1 \leq 4 \wedge 0 \leq x_2 \leq 2$ est représentée par la DBM de tableau 3.2. L'intersection entre z et g donne une DBM M' qui est présentée dans tableau 3.3. Ainsi, $z'[x_2 := 0]$ est représenté par la DBM M" qui est présentée dans tableau 3.4. Pour appliquer la remise à zéro de l'horloge x_2 sur $z \cap g$, on applique les règles suivantes :

— $M''_{2,0} = (0, \leq)//i = 2$

— $M''_{0,2} = (0, \leq)//i = 2$

— $M''_{2,0} = M''_{0,0}$ et $M''_{0,2} = M''_{0,0}//$ si i = 2 et j = 0

— $M''_{2,1} = M''_{0,1}$ et $M''_{1,2} = M''_{1,0}//$ si i = 2 et j = 1

On obtient finalement la zone $z'' = 1 \leq x_1 \leq 4 \wedge 1 \leq x_1 - x_2 \leq 4$.

$x_i - x_j$	x_0	x_1	x_2
x_0	$(0, \leq)$	$(0, \leq)$	$(0, \leq)$
x_1	$(4, \leq)$	$(0, \leq)$	$(4, \leq)$
x_2	$(2, \leq)$	$(2, \leq)$	$(0, \leq)$

Tableau 3.2 – DBM de la garde g

$x_i - x_j$	x_0	x_1	x_2
x_0	$(0, \leq)$	$(-1, \leq)$	$(-1, \leq)$
x_1	$(4, \leq)$	$(0, \leq)$	$(1, \leq)$
x_2	$(2, \leq)$	$(2, \leq)$	$(0, \leq)$

Tableau 3.3 – DBM de $z \cap g$

$x_i - x_j$	x_0	x_1	x_2
x_0	$(0, \leq)$	$(-1, \leq)$	$(0, \leq)$
x_1	$(4, \leq)$	$(0, \leq)$	$(4, \leq)$
x_2	$(0, \leq)$	$(-1, \leq)$	$(0, \leq)$

Tableau 3.4 – DBM de $z \cap g[\{x_2\} := 0]$

3.8.2 Graphe des zones

L'abstraction par graphe des zones propose une solution pour reduire l'espace d'etats tout en préservant des propriétés d'accessibilité équivalente au graphe des régions. Chaque état du graphe est un couple localité et zone. Il est formellement defini dans la définition suivante :

Définition :(22)
Soit $T = (L, l_0\Sigma, X, T, F)$ un TA, son graphe des zones Z(T) $\langle S^T, s_0^T, \Sigma, \Delta^T \rangle$ est un LTS défini ainsi :

- Les états de S^T sont des couples (l, z) ou $l \in L$ et z est une zone.
- L'état initial est $s_0^T = (l_0, z_0)$ ou l_0 est la localité initiale de T et z_0 est la zone initiale telle que toutes les horloges de X sont initialisées à zéro.
- Il y a un arc ((l, z), a, (l', z')) dans Δ^T si et seulement s'il existe une transition (l, a, g, X', l') dans T et une zone z" telles que :
  - z" = est le successeur de z,
  - z'' satisfait g,
  - $z' = z''[X' := 0]$,
  - $z' = K - approximation(z')$,
  - z' est une zone non vide.

L'opération K-approximation est une opération d'extrapolation. Elle permet d'assurer la terminaison de la construction du graphe des zones et que le nombre des états du graphe des zones soit fini. Cette opération est décrit dans la sous-section suivante. On a besoin d'une structure de données DBM (sous-section 3.8.1) pour représenter les zones.

Exemple :

L'automate de zone de la figure suivante 3.7 correspond à l'automate présenté précédemment figure 3.4

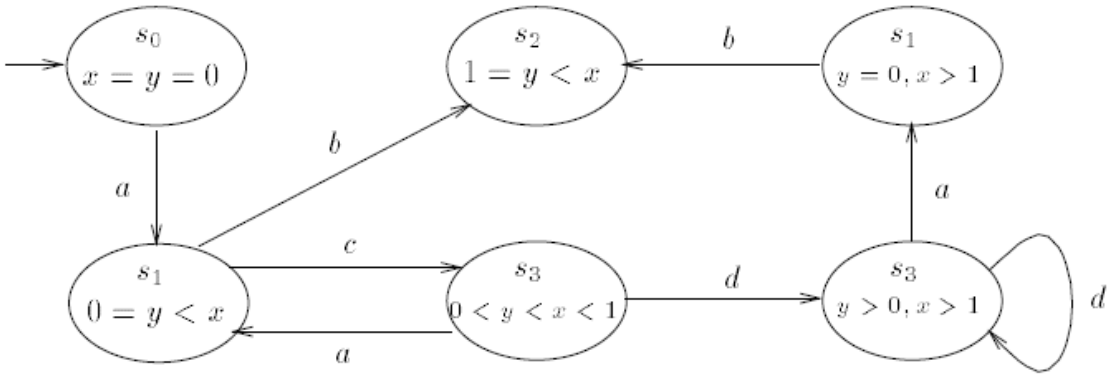

FIGURE 3.7 – Automate de zone

3.8.3 Approximations et améliorations

Dans les automates temporisées un inconvénient de la méthode est le recours nécessaire à des méthodes d'approximation (k-approximation) dans le cas où l'infini est utilisé dans les bornes des intervalles temporels, et dans ce cas, la finitude du graphe n'est pas garantie. Pour résoudre ce problème, une étape supplémentaire est alors introduite. Celle ci consiste à appliquer des approximations sur les zones admettant des bornes infinies. Ci-dessous, nous présentons l' approximation qui a été proposée.
Soit Z une zone sur un ensemble d'horloges X et $k \in Z$ (46).

k-approximation : L'idée est que, si une horloge va au-delà de la valeur k, sa valeur précise n'a pas d'importance et les comportements résultant de la zone obtenue sont les mêmes. Formellement, la k-approximation est obtenue comme suit :

$$\forall\, t_i; t_j \in Enable(m) \cup \{t_0\}; c_{ij} = \begin{cases} \infty \; si \; c_{ij} > k \\ -k \; si \; c_{ij} < -k \\ c_{ij} \; sinon \end{cases}$$

3.9 L'approche d'analyse en arrière

L'analyse en arrière se fait de la manière suivante : à partir des états qu'on veut atteindre, on calcule les états prédécesseurs en un pas, ensuite ceux en deux pas, ... et à chaque étape on teste si l'état initial est parmi les états calculés. Si c'est le cas, cela veut dire qu'on peut atteindre les états recherchés à partir de l'état initial. Si on a terminé le calcul et ce n'est pas le cas, cela veut dire qu'on ne peut jamais atteindre ces états recherchés.(43) Le principe des méthodes basées sur l'analyse en arrière est illustré sur Figure 3.8.

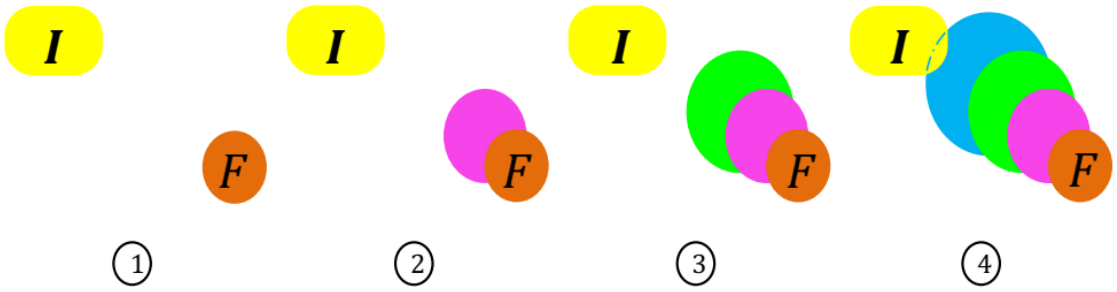

FIGURE 3.8 – Le calcul, pas à pas, des prédécesseurs de l'état final dans l'analyse en arrière.

L'utilisation des zones facilite le calcul d'un pas de l'analyse en arrière. Si on prend une transition de l'automate temporisé $t = s \xrightarrow{G,a,Y} s'$ et si Z' est une zone, alors l'objectif est de calculer l'ensemble des prédécesseurs en un pas de (s', Z') suivant la transition t qui est exactement l'ensemble des configurations (s, v) où v est une valuation d'horloges appartenant à la zone $Z = \overleftarrow{g \cap (Z' \cap Y = 0)[Y]^{-1}}$.

La terminaison du calcul itératif de l'analyse en arrière est une propriété très importante. La preuve de cette propriété est basée sur le fait qu'une zone se compose en réalité de plusieurs régions d'horloges. De ce fait il est facile de montrer que si Z' est une zone, alors la zone

Z qu'on vient de décrire est elle-même une union de régions. Comme il a été montré que le nombre de régions associées à un automate temporisé est fini, alors le nombre de paires (q, Z) qu'on est en train de calculer dans une analyse en arrière est lui-même fini.

Pratiquement, l'analyse en avant est implémentée dans la plupart des outils malgré les indéniables avantages de l'analyse en arrière. Les raisons peuvent se résumer en :

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

— l'analyse en arrière ne convient pas avec l'analyse des systèmes qui utilisent des structures de données de haut niveau par exemples les variables entières bornées ou bien les morceaux de programmes C, ... On trouve ces structures de données dans des outils qui implémente l'analyse en avant, par exemple l'outil Uppaal qui va être présenté dans la section 3.11 suivante.

3.10 L'approche d'analyse en avant

L'analyse en avant se fait de la manière suivante : à partir de l'état initial, on calcule les états successeurs en un pas, ensuite ceux en deux pas, ... et à chaque étape on teste si les états recherchés sont parmi les états calculés. Si c'est le cas, cela veut dire qu'on peut atteindre certains de ces états recherchés à partir de l'état initial. Si on a terminé le calcul et ce n'est pas le cas, cela veut dire qu'on ne peut jamais atteindre ces états recherchés (43). Le principe des méthodes basées sur l'analyse en avant est illustré sur Figure 3.9.

L'utilisation des zones facilite le calcul d'un pas de l'analyse en avant. Si on prend une transition de l'automate temporisé $t = s \xrightarrow{G,a,Y} s'$ et si Z est une zone, alors l'objectif est de calculer l'ensemble des successeurs en un pas de (s, Z) suivant la transition t qui est exactement l'ensemble des configurations (s', v') où v' est une valuation d'horloges appartenant à la zone $Z' = (g \cap \overrightarrow{Z})[Y]$.

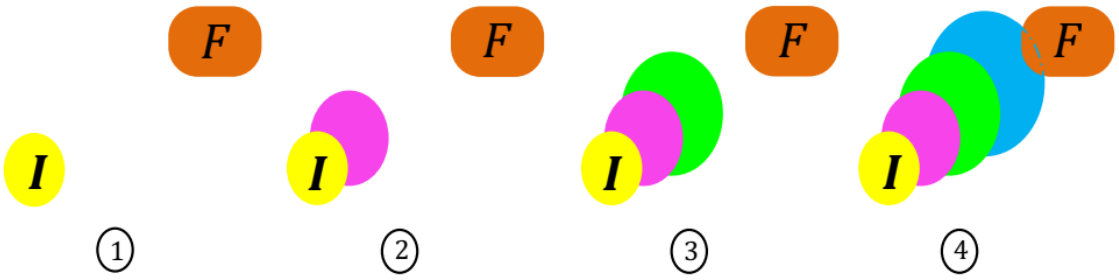

FIGURE 3.9 – Le calcul, pas à pas, des successeurs de l'état initial dans l'analyse en avant.

Généralement le calcul itératif de l'analyse en avant ne se termine pas, contrairement au calcul de l'analyse en arrière. Figure 3.10 et Figure 3.11 présentent un automate temporisé qui illustre cet inconvénient de l'analyse en avant. La valeur de l'horloge x de cet exemple s'incrémente avec une unité de temps pour chaque itération du calcul en avant, par conséquent ce calcul ne va jamais se terminer.

Généralement un opérateur d'abstraction est utilisé à chaque étape du calcul afin de faire face à ce problème de terminaison. Cette opération d'abstraction s'appelle normalisation ou

encore extrapolation. Soit k la plus grande constante avec laquelle une horloge de l'automate temporisé a été comparée dans les contraintes. Maintenant soit Z une zone, l'extrapolation de Z par rapport à k est la plus petite zone comprenant Z et qui est définie par des contraintes qui utilisent que des constantes entre $-k$ et $+k$. Cette opération d'extrapolation a comme objectif : Comme toutes les contraintes d'horloges de l'automate temporisé sont bornées par k, alors ce qui est important est de savoir uniquement que la valeur d'une horloge a dépassé la borne k car on n'a pas vraiment besoin de sa valeur exacte. Notons que l'utilisation de cette opération d'extrapolation pendant toute l'analyse en avant assure la terminaison du calcul itératif parce que le nombre de zones définies par des contraintes qui n'utilisent que des constantes de l'intervalle $[-k, +k]$ est fini.

En outre, cette approche d'analyse en avant peut engendrer un autre problème à chaque itération du calcul : une sur-approximation calculée des états accessibles. Donc ce calcul itératif peut nous retourner des états déclarés comme accessibles par contre en réalité ils ne le sont pas.

Il a été montré dans (12) que cette approche d'analyse en avant est correcte si on considère les automates temporisés sans contraintes diagonales, par contre elle est incorrecte pour la classe des automates avec contraintes diagonales.

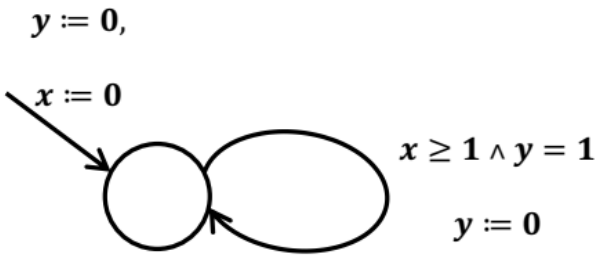

FIGURE 3.10 – Un automate temporisé qui illustre la non terminaison de l'analyse en avant.

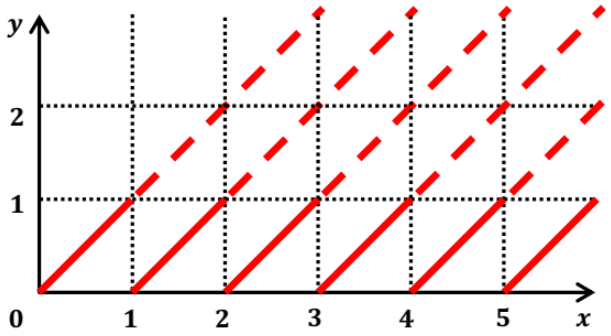

FIGURE 3.11 – Le calcul itératif en avant associé.

3.11 Outils de vérification :

Voici un panorama de certains outils existants et de leurs fonctionnalités :
Uppaal : développé à Uppsala (Suède) et Aalborg (Dannemark) (41)

— accessibilité, bloquage, fragment simple de TCTL

— analyse en avant
→ http ://www.uppaal.com

Uppaal est un outil développé par Université d'Uppsala (Suède) et Université d'Aalborg (Danemark) (42). Il sert à vérifier les systèmes temporisés. Cet outil est libre à l'adresse http ://www.uppaal.com. Il est conçu pour faire de la vérification sur une variante des automates temporisés. Cette variante est plus large syntaxiquement parce qu'elle autorise une représentation directe des

— contraintes d'urgence : dans le cas d'une transition qui doit être tirée immédiatement, sans attente,

— contraintes d'atomicité : quand on est obligé de tirer instantanément une séquence de transitions,

— etc.

Ces facilités de modélisation améliorent la concision et la lisibilité du modèle original mais n'ajoutent pas en effet de l'expressivité. L'outil Uppaal offre la possibilité de vérifier un sous-ensemble de $TCTL_h$ qui regroupe surtout les propriétés d'accessibilité, de sûreté et aussi les propriétés de réponse. Pour plus de détails sur l'utilisation de cet outil, un tutoriel est disponible dans l'adresse web mentionnée plus-haut (6).

Uppaal comprend principalement trois modules concernant la construction du modèle, la vérification et la simulation. Ce dernier module permet de voir l'évolution du modèle construit.

HyTech : développé à Berkeley (USA) (29)

— pas de logique de spécification, riche langage de calcul, automates hybrides

— analyse en avant et en arrière
→ http ://www-cad.eecs.berkeley.edu/ tah/HyTech/

L'outil HyTecH (HyTecH signifie Hybrid Technology Tool) a été développé à l'uni- versité de Berkeley par Tom Henzinger, Pei-Hsin Ho et Howard Wong-Toi. Cet outil vise à vérifier des réseaux d'automates hybrides très généraux. Il permet même d'analyser des systèmes paramétrés. Il vérifie principalement des propriétés d'accessibilité et de sûreté. La représentation symbolique qui est utilisée dans HyTecH est celle des polyèdres convexes, c'est-à-dire des intersections d'hyperplans (13), (28), (29).

CMC : développé à Cachan (France) (40)

— logique modale L_v

— méthode compositionnelle, analyse en avant
→ http ://www.lsv.ens-cachan.fr/ fl/cmcweb.html

L outil CMC (CMC signifie Compositional Model-Checking) a été développé au Laboratoire de Spécification et Vérification à l ENS de Cachan par François Laroussinie. Cet outil vise à vérifier des réseaux d'automates temporisés. Il utilise comme langage de spécification la logique Lv (56) (1) (c'est une logique modale temporisée). L'algorithme qui est implémenté est basé sur l'approche compositionnelle présentée dans [LL98]. Des stratégies de simplification de formules ont été aussi étudiées (44), (39), (40).

Kronos : développé à Grenoble (France) (14)

— Timed CTL

— analyse en avant et en arrière
→ http ://www-verimag.imag.fr/TEMPORISE/kronos/

L'outil Kronos a été développé au laboratoire Vérimag à Grenoble principalement par Sergio Yovine, Alfredo Olivero, Conrado Daws et Stavros Tripakis. Cet outil vise à vérifier des réseaux d'automates temporisés. Il permet de vérifier des propriétés énoncées dans la logique TCTL. Dans cet outil, plusieurs algorithmes sont implémentés, tout d'abord l'analyse en avant, puis l'analyse en arrière, enfin l'algorithme pour TCTL. De multiples articles présentent l'outil Kronos et ses applications, par exemple (60), (30), (21), (61), (14), (62).
Remarque Les grands avantages de l'outil UPPAAL par rapport aux autres outils que nous avons décrits précédemment sont d'une part, son interface graphique très conviviale et d'autre part, son module de simulation qui permet, lors de la phase de modélisation de faire des tests du modèle et de détecter éventuellement des erreurs dans la modélisation.

3.12 Conclusion

Dans ce chapitre, nous proposons les automates temporisés. Ensuite, nous examinons le problème de décidabilité du vide (Le problème du vide est-il toujours décidable ?). Nous présentons les régions et les zones comme une solution envisagée à cette problématique, qui constitue un sujet central de notre mémoire. Enfin, nous abordons la présentation d'outils qui facilitent la vérification et la modélisation des systèmes.

Chapitre 4

Automate temporisé avec temp relatif r-TA

Sommaire

4.1 Introduction 45
4.2 Automates temporisés avec temp relatif 46
4.2.1 Définition 46
4.2.2 Sémantique 47
4.3 Véréfication automatique 47
4.3.1 Paramètre *slope* et les régions d'horloges 48
4.3.2 Classes d'équivalence sur les valuations d'horloges 49
4.3.3 Successeurs des régions d'horloges 50
4.4 Automate de régions 52
4.4.1 Définition 52
4.5 Automate de zones avec temps relatif 53
4.5.1 Définition 53
4.5.2 Algorithme de zones classique 54
4.5.3 Algorithme de zones avec temps relatif 55
4.5.4 Les r-DBMs 55
4.5.5 Exemple : 56
4.6 Conclusion 62

4.1 Introduction

Les automates temporisés sont un outil puissant pour la modélisation, l'analyse et la vérification des systèmes temps réel critiques et hétérogènes. Ils permettent de capturer les contraintes temporelles spécifiques de ces systèmes complexes et de garantir leur bon fonctionnement.

les automates temporisés sont confrontés à des défis dans les systèmes hétérogènes en raison des différentes propriétés des composants qui interagissent. Les composants peuvent avoir des horloges internes fonctionnant à des fréquences différentes, des délais de communication variables ou des comportements de synchronisation spécifiques. Les automates temporis peuvent désormais modéliser ces différences et expliquer la complexité temporelle des systèmes hétérogènes.Nous parlons maintenant sur les automates temporisés avec temps relatif qui est une variante du modèle classique des automates temporisés, qui est le modèle étudié par notre encadrant Mr Layadi.Alors ce chapitre est basé sur les résultats publiés dans (Layadi et al. 2016).

Notre contribution se concentre spécifiquement sur le développement d'un automate de zone avec temps relatif, une avancée majeure dans le domaine de l'automatisation industrielle. En intégrant la notion de temps relatif, notre système permet une synchronisation précise et dynamique des différentes tâches effectuées par les automates de zone, offrant ainsi une performance optimale et une flexibilité accrue.

4.2 Automates temporisés avec temp relatif

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. (43)

Le produit asynchrone B= $(S, Act, Z, T, I, s_0, F)$ 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 :

4.2.1 Définition

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 $r-TA$.

4.2.2 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 τ= $(\tau_p)_{p\in Proc}$. 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 $\tau : R_{\geq 0} \rightarrow R_{\geq 0}^{Proc}$ 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 $(Q, q_0, \rightarrow)$ où $Q \subseteq S \times R_{\geq 0}^{Z}$ 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 $\rightarrow$ entre les configurations est définie par les règles suivantes :

— $< s \mid v > \xrightarrow{d} < s \mid v + \tau(d) >$

si $\forall d' , 0 \leq d' \leq d \Rightarrow (v + \tau(d')) \models I(S)$ pour tout $d \in R_{\geq 0}$, avec $v + \tau(t) = (v(x) + \tau_p(t))_{x\in Z_p \; and \; p\in Proc}$

— $< s \mid v > \xrightarrow{a} < s[s'_p/s_p] \mid v' >$
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 Vérification automatique

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

Comme le nombre de configurations $< s \mid v >$ dans le système de transitions est infini, la construction de ce dernier est impossible.

Dans la littérature (Alur & Dill 1994), des relations d'équivalence ont été proposées pour agréger les configurations des systèmes de transitions temporisés telles que chaque classe d'équivalence représente un ensemble de configurations $< s_i \mid v_i >$. Dans ce cas, les relations d'équivalence sont construites en respectant les exécutions. Les classes d'équivalence des valuations d'horloges sont appelées régions d'horloges.(43)

4.3.1 Paramètre *slope* et les régions d'horloges

D'habitude, le temps est mesuré par des dispositifs physiques, appelés horloges, qui offrent un comportement presque régulier au cours du temps.

La fonction $v(x)$ donne la valeur de l'horloge x. Pour obtenir les valeurs de la valuation d'horloges x à l'instant du temps absolu t, le tuple des fonctions continues de vitesses $\tau(t)$ est utilisé.

Pour une paire d'horloges x et y (qui appartiennent respectivement aux processus p et q), leurs propres fréquences lui feront diverger de la référence du temps absolu avec un certain degré qui est égale au rapport entre leurs propres fréquences. Il représente la pente de la ligne droite sur Figure 4.1. Ce rapport de vitesses est appelé *slope*, tel que $slope_{xy} = \tau_q(t)/\tau_p(t)$. Il représente la pente de la droite de Figure 4.2.

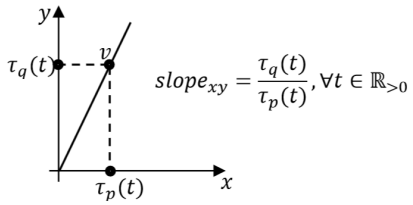

FIGURE 4.1 – Evolution de deux horloges avec différentes fréquences

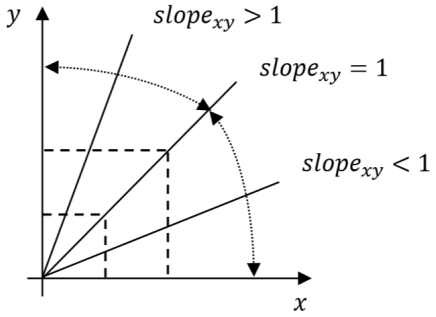

FIGURE 4.2 – Différentes possibilités de l'évolution de deux horloges

Lorsqu'une horloge est remise à zéro, la progression de sa valeur doit reprendre de zéro avec la même fonction de vitesse (voir Figure 4.3).

La conception des systèmes devient cohérente si les composants partagent une perception conjointe du temps (Verssimo 1994 ; Lenzen et al. 2010). Par conséquent, il est important que la perception générale soit consistante. Cette perception prend sa pleine dimension quand on construit le graphe de la sémantique du modèle.

À tout moment, le comportement futur du système modélisé est déterminé par sa localité et les valeurs des horloges de tous les processus composant ce système, ceci motive la redéfinition de quelques concepts, comme les régions d'horloges et l'automate de régions. On se concentrera ci-après sur l'effet des fréquences relatives d'horloges.

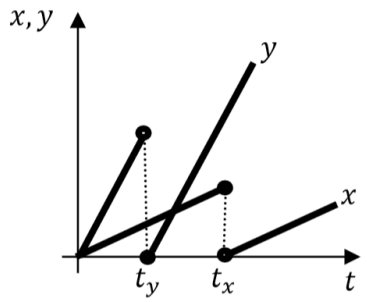

FIGURE 4.3 – Réinitialisation de deux horloges avec différentes fonctions linéaires de vitesses

4.3.2 Classes d'équivalence sur les valuations d'horloges

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

Définition 4.1

Soit x (respectivement y) une horloge qui appartient au processus p (respectivement q)et qui évolue selon la fonction de fréquence τ_p (respectivement τ_q). On définit $slope_{xy} = \tau_q/\tau_p$. Pour les r-TA, nous nous limitons à l'utilisation des constantes entières dans les contraintes temporelles. Cela peut être fait en multipiliant toutes les constantes apparaissant dans les contraintes temporelles par leur plus petit commun multiple des dénominateurs, voir la preuve dans (Alur & Dill 1994). Dans le reste de ce chapitre, on suppose que les contraintes temporelles ne comportent que des constantes entières. Comme il n'y a qu'un nombre fini de contraintes temporelles sur toute horloge x, on peut déterminer l'entier le plus grand $c_x \in \mathbb{N}$ avec lequel x a été comparée dans une contrainte temporelle (garde ou invariant) du 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 4.2

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.

4.3.3 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 $\alpha', t \in \mathbb{R}_{>0}$ de telle sorte que $v' = v + \tau(t)$ avec $v + \tau(t) = (v(x) + \tau_p(t))_{\pi^{-1}(x)=p}$.

Par exemple dans Figure 4.9, les cinq successeurs de la région $\alpha = [(1.5 < x < 2), (1 < y < 2x - 2)]$ sont : elle même, $[(x = 2), (1 < y < 2)], [(x > 2), (1 < y < 2)], [(x > 2), (y = 2)]$ et $[(x > 2), (y > 2)]$. Ces régions sont celles couvertes par une ligne tracée, d'un point quelconque en α, en parallèle à la ligne $y = slope_{xy} x = 2x$ (avec une direction vers le haut).

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

— **étape 1.** Pour chaque horloge x, une contrainte de la forme $(x = c)$, $c < x < c + \frac{1}{slope_{max(x)}})$ ou bien $(x > c_x)$ et

— **étape 2.** Pour toute paire x et y de telle sorte que $(c < x < c + \frac{1}{slope_{max(x)}}$ et $(d < y < d + \frac{1}{slope_{max(y)}})$ apparaissent dans l'étape 1, une relation d'ordre entre $slope_{xy}(x - c)$ et $y - d$.

Pour calculer les successeurs possibles, trois cas sont distingués :

Premier Cas

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

Deuxième Cas

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

1. Un ensemble de contraintes temporelles qui peut être donné comme suit :

   (a) Pour toute horloge $x \in Z_0$:

      i 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

Troisième Cas

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

1. Un ensemble de contraintes temporelles qui peut être donnée comme suit :

   (a) Pour toute horloge $x \in Z_0$, si α satisfait $(c < x < c + \frac{1}{slope_{max(x)}})$ alors β satisfait $(x = c + \frac{1}{slope_{max(x)}})$;

(b) Pour toute horloge $x \notin Z_0$, la contrainte temporelle de α reste la même que celle dans β

2. Pour toute paire f'horloges x et y tel que $(c < x < c + \frac{1}{slope_{max(x)}}$ et $(d < y < d + \frac{1}{slope_{max(y)}}$ 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

4.4 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 B t 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 $< s'_0, [v'_0] >$ et l'étiquette α si et seulement s'il existe une transition étiquetée par α partir d'une localité s avec une valuation d'horloges $\upsilon \in \alpha$ vers une localité s' avec une valuation d'horloges $\upsilon' \in \alpha'$ ans le rd-TA (B) Pour une région d'horloges α et un ensemble d'horloges $R \subseteq Z$ la région $\alpha[R]$ dénote la réinitialisation de toutes les horloges de R dans toute valuation υ de α Formellement $\forall \upsilon \in \alpha, \forall x \in Z$:(43)

— si $x \in R$ alors $\upsilon(s) = 0$ dans $\alpha[R]$.

— sinon $\upsilon(x)$ dans $\alpha[R]$ est la meme que dans α.

4.4.1 Définition

Soit B= $(S, Act, Z, T, I, s_0, F)$ r-TA sur un ensemble de processus $Proc$ un tuple de fonctions de vitesses locales du temps τ L'automate de régions$R(B)$ est un automate sur l'alphabet Act tel que :

— Les configurations de $R(B)$ sont de la forme $< s|\alpha >$où s est une localité de B et α est une région d'horloges.

— La configuration initiale est de la forme $< s_o|[\upsilon_0] >$ ou $\upsilon_0(x) = 0$ 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 $c_x \in N$ 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 :

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

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

4.5 Automate de zones avec temps relatif

L'algorithme d'accessibilité basé sur le graphe des régions n'est pas utilisé en pratique dans les outils de model checking en raison de la taille du graphe des régions. En effet, la taille de $(\mathbb{R}_+^X)_{/\equiv M}$ est exponentielle dans le nombre d'horloges ($|X|$) et dans le codage de M ($|(\mathbb{R}_+^X)_{/\equiv M}$ est en $O(|X|! \cdot M^{|X|})$). Cette explosion combinatoire empêche la construction explicite de RA. En fait, l'accessibilité dans un automate temporisé est un problème difficile.

Le saut de complexité induit par les contraintes d'horloges a nécessité le développement de structures de données spécifiques pour manipuler les ensembles de valuations. L'idée générale est d'éviter de construire le graphe des régions et plutôt d'utiliser des structures de données symboliques, permettant de représenter des ensembles plus grands de valuations. Il existe plusieurs solutions mais, la structure de données DBM (Difference Bound Matrix) la plus utilisée dans le domaine. Nous avons abordé cette structure dans la sous-section 3.8.1 et vous pouvez consulter les références [(24), (11)] pour des explications détaillées.

Les DBM sont utilisées pour représenter les zones d'horloge dans les automates temporisés. L'objectif d'une telle partition en zones est souvent de réduire la complexité de l'automate en regroupant des états équivalents dans une même zone. Cela facilite l'analyse et la vérification formelle de l'automate.

4.5.1 Définition

Dans le contexte des automates temporisés avec temps relatif, une définition formelle des zones peut être donnée comme suit :

Soit $A = (L, X, Act, T, I, F)$ un automate temporisé avec temps relatif, où :
- L est l'ensemble fini des localités ou états,
- X est l'ensemble fini des horloges,
- *Act* est l'alphabet des symboles d'entrée,
- T est l'ensemble fini des contraintes temporelles,
- I est la fonction d'initialisation qui associe à chaque localité un ensemble de valuations initiales des horloges,
- F est l'ensemble fini des transitions.

Une zone dans un automate temporisé avec temps relatif est un sous-ensemble de l'ensemble des localités L qui regroupe des localités partageant des propriétés temporelles similaires.

Plus précisément, une zone peut être définie en termes de contraintes temporelles satisfaites par les localités qu'elle contient. Les localités d'une zone peuvent avoir des propriétés temporelles communes, telles que des limites supérieures ou inférieures sur les valuations des horloges spécifiques, des délais spécifiques avant de franchir une transition, ou d'autres contraintes temporelles spécifiques.

La création de zones dans un automate temporisé permet de regrouper des localités qui partagent des propriétés temporelles similaires, ce qui facilite l'analyse, la vérification et la synthèse du système. Les zones peuvent être utilisées pour simplifier la complexité de l'automate, pour identifier des régions de comportement similaire, ou pour faciliter la planification et la gestion du temps dans le système modélisé.

4.5.2 Algorithme de zones classique

Nous avons utilisé cet algorithme comme point de départ pour créer un algorithme robuste répondant à nos besoins spécifiques ("le temps relatif").

Algorithm 1 Algorithme des zones pour les automates temporisés classiques

Algorithme des zones $(A : TA)$ {
  Définir k ;
  Visités := $\emptyset$; (* Visités contient les états visités *)
  Attente := $\{(s_0, Approx_k(Z_0))\}$;
  Répéter
    Prendre (s, Z) dans Attente et le retirer ;
    Si s est final alors {Retourner «~Oui~» ;}
      sinon {S'il n'y pas de $(s, Z') \in$ Visités t.q. $Z \subseteq Z'$
        alors {Visités := Visités $\cup \{(s, \overrightarrow{Z})\}$;
          Successeur := $\{(s', Approx_k(Post(Z, e)))|$
            e transition de s à $s'\}$;
          Attente := Attente $\cup$ Successeur ;}}
  Jusqu'à (Attente = $\emptyset$) ;
  Retourner «~Non~» ; }

Voici une explication étape par étape de l'algorithme :

1. Définir k : Cela correspond à la précision ou à la granularité de l'approximation utilisée dans l'algorithme. Classiquement, on lui donne la valeur la plus élevée dans les horloges de r-TA.

2. Initialiser les ensembles Visités et Attente : L'ensemble Visités contient les états déjà visités, et l'ensemble Attente contient les états en attente d'être explorés. Au départ, l'ensemble Attente contient le premier état initial s_0 associé à l'approximation de la zone Z_0 et l'ensemble Visités est vide .
3. Boucle principale :
   (a) Prendre un état s et sa zone Z à partir de l'ensemble Attente, puis le retirer de l'ensemble.
   (b) Vérifier si l'état s est un état final. Si c'est le cas, cela signifie que l'automate temporel atteint un état final. Dans ce cas, l'algorithme retourne la réponse "Oui".
   (c) Sinon, s'il n'existe aucun état (s, Z') dans l'ensemble Visités tel que Z est inclus dans Z', cela signifie que la zone Z n'a pas été visitée auparavant. Dans ce cas, ajouter l'état s et sa future de zone Z "$(s, \overrightarrow{Z})$" à l'ensemble Visités.
   L'opération de futur Z et défini par $\overrightarrow{Z} = \{v + t) \mid v \in Z \text{ et } t \geq 0\}$
   (d) Calculer les successeurs de l'état s : Pour chaque transition e de s à un état successeur s', approximer la post-condition de la zone Z en utilisant l'opération $Post(Z, e)$ ou elle représente l'ensemble $[C \leftarrow 0](g \cap \overrightarrow{Z})$. Ces successeurs sont ajoutés à l'ensemble Attente.
4. Répéter l'étape 3 jusqu'à ce que l'ensemble Attente soit vide. Cela signifie que tous les états accessibles ont été explorés.
5. Si l'ensemble Attente est vide, cela signifie que l'automate temporel n'a pas atteint d'état final. Dans ce cas, l'algorithme retourne la réponse "Non".

4.5.3 Algorithme de zones avec temps relatif

Il convient de noter que bien que l'algorithme soit fondamentalement identique, des adaptations significatives ont été apportées à la structure des opérations, notamment l'opération futur . Les détails des opérations à modifier sont donnés dans la sous-section suivante.Ces modifications ont été mises en œuvre dans le but de répondre aux besoins spécifiques du contexte actuel "le temps relatif". Cette adaptation nous permet d'obtenir des résultats plus fiables et plus pertinents.

4.5.4 Les r-DBMs

Une matrice de différences bornées avec temps relatif , ce que nous abrégerons en r-DBM pour n horloges est une matrice carrée $(m_{i,j}, \prec_{i,j})_{i,j=0\ldots n}$ de taille $n+1$ de paires

$$(m; \prec) \in V = (Z \times \{<, \leq\}) \cup (\infty; <).$$

Une r-DBM $M = (m_{i,j}, \prec_{i,j})_{i,j=0\ldots n}$ définit l'ensemble suivant de T^n (l'horloge x_0 est supposée être onstamment égale à zéro, i.e. pour toute valuation $v, v(x_0) = 0$) :

$$\{v : \{x_1\ ,\ \ldots\ ,\ x_n\} \to T | \forall\ 0 \leq i, j \leq n, v(x_i) - v(slope_{x_i x_j} x_j) \prec_{i,j} m_{i,j} \text{ tel que } slope_{x_i x_j} = \tau_q/\tau_p \text{ et } x_i \in T(q), x_j \in T(p)\}$$

Opérations sur r-DBM

Intersection

$z \cap z'$: Soit M et M' deux r-DBMs qui représentent les zones z et z' et$M'' = M \cap M'$, l'intersection de M et M' est une r-DBM M'' tel que $M''_{i,j} = min(M_{i,j}, M'_{i,j})$.

Futur
$\overrightarrow{z} = \{v + \tau(t) \mid v \in Z \text{ et } t \geq 0 \text{ et } \tau(t) = (\tau_p(t))_{p \in Proc}\}$

Dans les DBM en définir M' = future(M) comme suit :

— $M'_{i,0} = \infty$ pour tout $i \neq 0$
— $M'_{i,j} = M_{i,j}$ si $i = 0$ou $j \neq 0$

Remise à zéro

La remise à zero d'un ensemble d'horloges X' dans une zone z est egale $z' = z[X' := 0]$. La zone z' est représentée par le r-DBM M' tel que :

— $M'_{i,0} = (0, \leq)$ pour tout $x_i \in X'$
— $M'_{0,i} = (0, \leq)$ pour tout $x_i \in X'$
— $M'_{i,j} = M'_{0,j}$ pour tout $x_i \in X'$ et $x_j \notin X'$
— $M'_{j,i} = M'_{j,0}$pour tout $x_i \in X'$ et $x_j \notin X'$

4.5.5 Exemple :

Un automate temporisé avec temps relatif est représenté par le tuple :$B = (S, Act, Z, T, I, s_0, F)$ Nous donnons un exemple de r-TA tel que :

— $S = \{l0, l1, l2\}$
— $Act = \{a, b\}$
— $Z = \{x, y\}$
— $T = \{(l0, a, [x \leq 2, y := 0], l1); (l1, b, [y \leq 1, x := 0], l2)\}$
— $s0 = \{l0\}$
— $F = \{l2\}$

Les étapes d'exécution :

1. Définir k tel que $k = max(v)$ alors $k = 2$
2. Initialiser les ensembles Visités et Attente : Au départ, l'ensemble Attente contient la première localité de l'état initial l_0 associé à l'approximation de la zone Z_0 et l'ensemble Visités est vide .
3. Boucle principale :

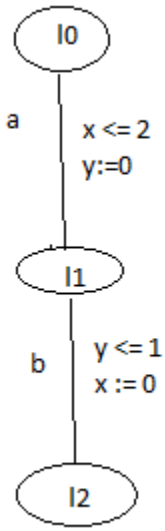

FIGURE 4.4 – Exemple d'un r-TA

(a) Prendre la localité $l0$ et sa zone Z_0 à partir de l'ensemble Attente, puis le retirer de l'ensemble.

(b) Vérifier si la localité $l0$ est un état final. Dans ce cas, l'algorithme est terminé.

(c) Sinon, s'il n'existe aucun état (s, Z') dans l'ensemble Visités tel que Z est inclus dans Z'. Dans ce cas, ajouter l'état s et sa future de zone Z "$(s, \overrightarrow{Z})$" à l'ensemble Visités.
la future : $\overrightarrow{z} = \{z + \tau(t)\}$, tel que le $\tau(t) = (\tau_q, \tau_p)$ et $\tau_q = t$ et $\tau_p = 2 * t$ avec $x \in Z(q), y \in Z(p)\}$ implique que : $slope_{xy} = \tau_q/\tau_p = 1/2$, alors : $x = 1/2 * y$ et $y = 2 * x$:

— $Z_0 =$

	x_0	x	$slope_{xy} * y$
x_0	$(0, \leq)$	$(0, \leq)$	$(0, \leq)$
x	$(0, \leq)$	$(0, \leq)$	$(0, \leq)$
$slope_{xy} * y$	$(0, \leq)$	$(0, \leq)$	$(0, \leq)$

— Ensuite, on a appliquer l'opération future Z . la matrice future Z_0 est :

	x_0	x	$slope_{xy} * y$
x_0	$(0, \leq)$	$(0, \leq)$	$(0, \leq)$
x	$(\infty, \leq)$	$(0, \leq)$	$(0, \leq)$
$slope_{xy} * y$	$(\infty, \leq)$	$(0, \leq)$	$(0, \leq)$

Et noteé que : $Z_0 = [x = 1/2y \geq 0]$
la représentation graphique de cette zone est donneé par la figure 4.5 :

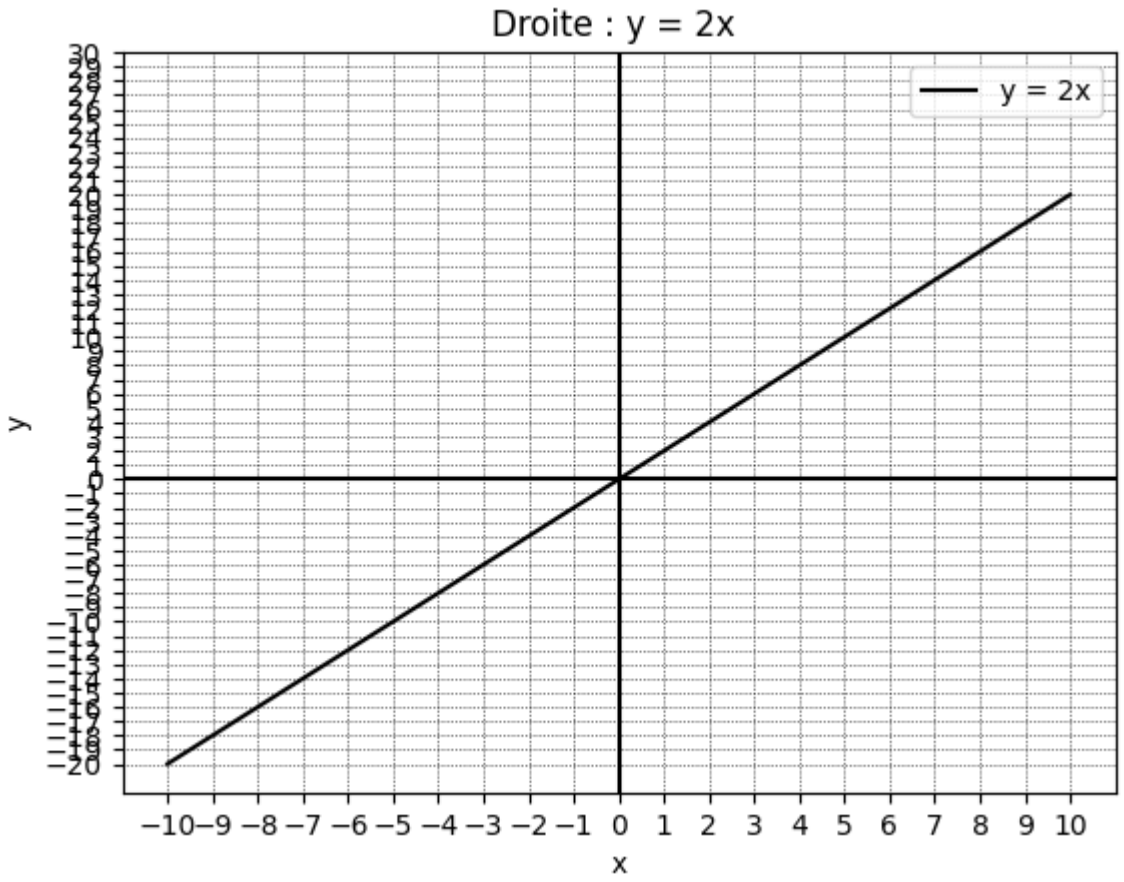

FIGURE 4.5 – Future Z_0

(d) Calculer les successeurs de l'état $s0$:

— La garde $g_a = 0 \leq x \leq 2$ et $y \geq 0$ pour la transition a de la localité l0 vers la localité l1 est donnée par le r-DBM suivant :

	x_0	x	$slope_{xy} * y$
x_0	$(0, \leq)$	$(0, \leq)$	$(0, \leq)$
x	$(2, \leq)$	$(0, \leq)$	$(2, \leq)$
$slope_{xy} * y$	$(\infty, \leq)$	$(\infty, \leq)$	$(0, \leq)$

— L'intersection de future Z_0 avec la garde ga est représentée par :

	x_0	x	$slope_{xy} * y$
x_0	$(0, \leq)$	$(0, \leq)$	$(0, \leq)$
x	$(2, \leq)$	$(0, \leq)$	$(0, \leq)$
$slope_{xy} * y$	$(2, \leq)$	$(0, \leq)$	$(0, \leq)$

— Enfin, on remet à zéro l'horloge $y := 0$ pour obtenir la matrice de Z_1 :

	x_0	x	$slope_{xy} * y$
x_0	$(0, \leq)$	$(0, \leq)$	$(0, \leq)$
x	$(2, \leq)$	$(0, \leq)$	$(2, \leq)$
$slope_{xy} * y$	$(0, \leq)$	$(0, \leq)$	$(0, \leq)$

— Et, on a appliquer l'opération future Z sur le dernier r-DBM pour obtenu la zone Z_1. le r-DBM du zone $Z1$ est :

	x_0	x	$slope_{xy} * y$
x_0	$(0, \leq)$	$(0, \leq)$	$(0, \leq)$
x	$(\infty, \leq)$	$(0, \leq)$	$(2, \leq)$
$slope_{xy} * y$	$(\infty, \leq)$	$(0, \leq)$	$(0, \leq)$

On note que le dernier r-DBM correspond à la zone Z_1 : $Z1 = [0 \leq x - 1/2y \leq 2]$ et donne la représentation graphique du zone dans la figure 4.6 :

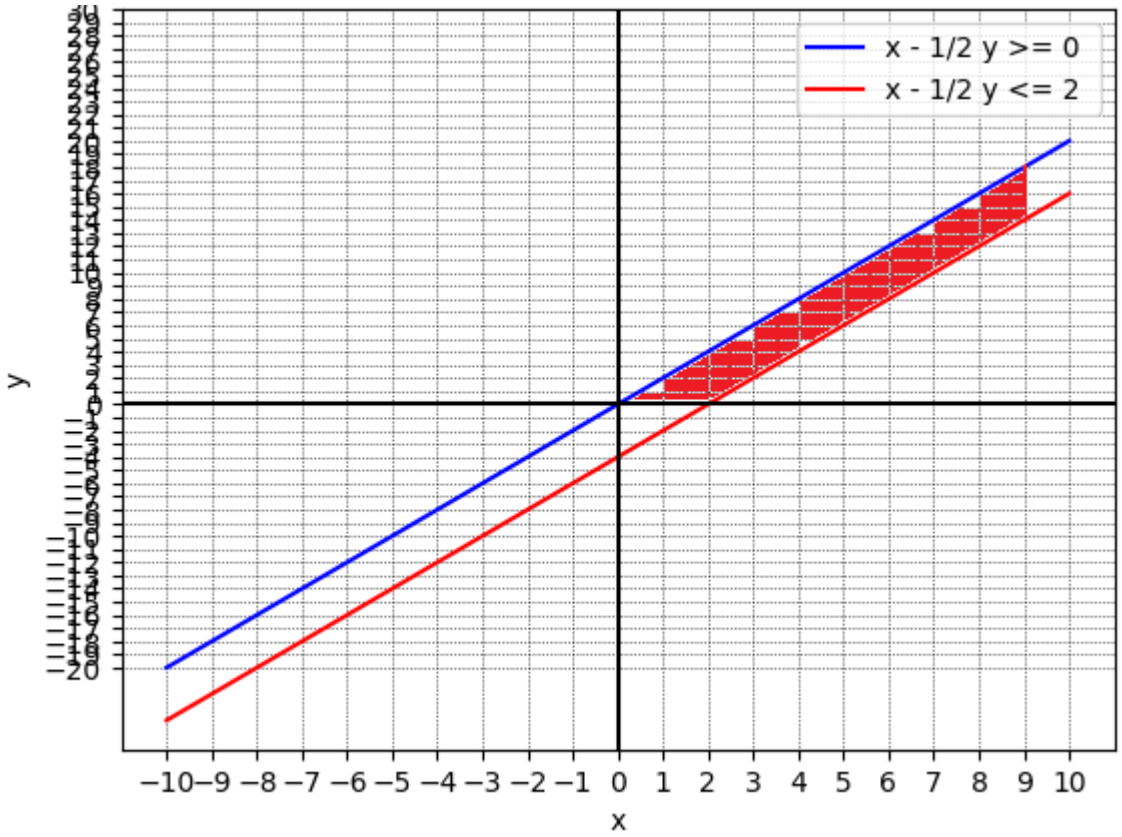

FIGURE 4.6 – Future Z_1

Ce successeur est ajouté à l'ensemble Attente.

4. Répétition :

— La garde $gb = 0 \leq y \leq 1$ et $x \geq 0$ pour la transition a de la localité l1 vers la localité l2 est donnée par le r- DBM suivant :

	x_0	x	$slope_{xy} * y$
x_0	$(0, \leq)$	$(0, \leq)$	$(0, \leq)$
x	$(\infty, \leq)$	$(0, \leq)$	$(\infty, \leq)$
$slope_{xy} * y$	$(1, \leq)$	$(1, \leq)$	$(0, \leq)$

— L'intersection de future Z_1 avec la garde gb est représentée par :

	x_0	x	$slope_{xy} * y$
x_0	$(0, \leq)$	$(0, \leq)$	$(0, \leq)$
x	$(\infty, \leq)$	$(0, \leq)$	$(2, \leq)$
$slope_{xy} * y$	$(1, \leq)$	$(0, \leq)$	$(0, \leq)$

— Enfin, on remet à zéro l'horloge $x := 0$ pour obtenir la matrice de Z_2 :

	x_0	x	$slope_{xy} * y$
x_0	$(0, \leq)$	$(0, \leq)$	$(0, \leq)$
x	$(0, \leq)$	$(0, \leq)$	$(0, \leq)$
$slope_{xy} * y$	$(1, \leq)$	$(1, \leq)$	$(0, \leq)$

— Et la future du dernier r-DBM est :

	x_0	x	$slope_{xy} * y$
x_0	$(0, \leq)$	$(0, \leq)$	$(0, \leq)$
x	$(\infty, \leq)$	$(0, \leq)$	$(0, \leq)$
$slope_{xy} * y$	$(\infty, \leq)$	$(1, \leq)$	$(0, \leq)$

On note que $Z_2 = [0 \leq y - 2x \leq 1]$ et tant que $x = 1/2y$ on peut écrire la zone Z_2 comme suivant : $Z_2 = [-1 \leq y - 2x \leq 0]$
et la représentation graphique de la zone est dans la figure 4.7 :

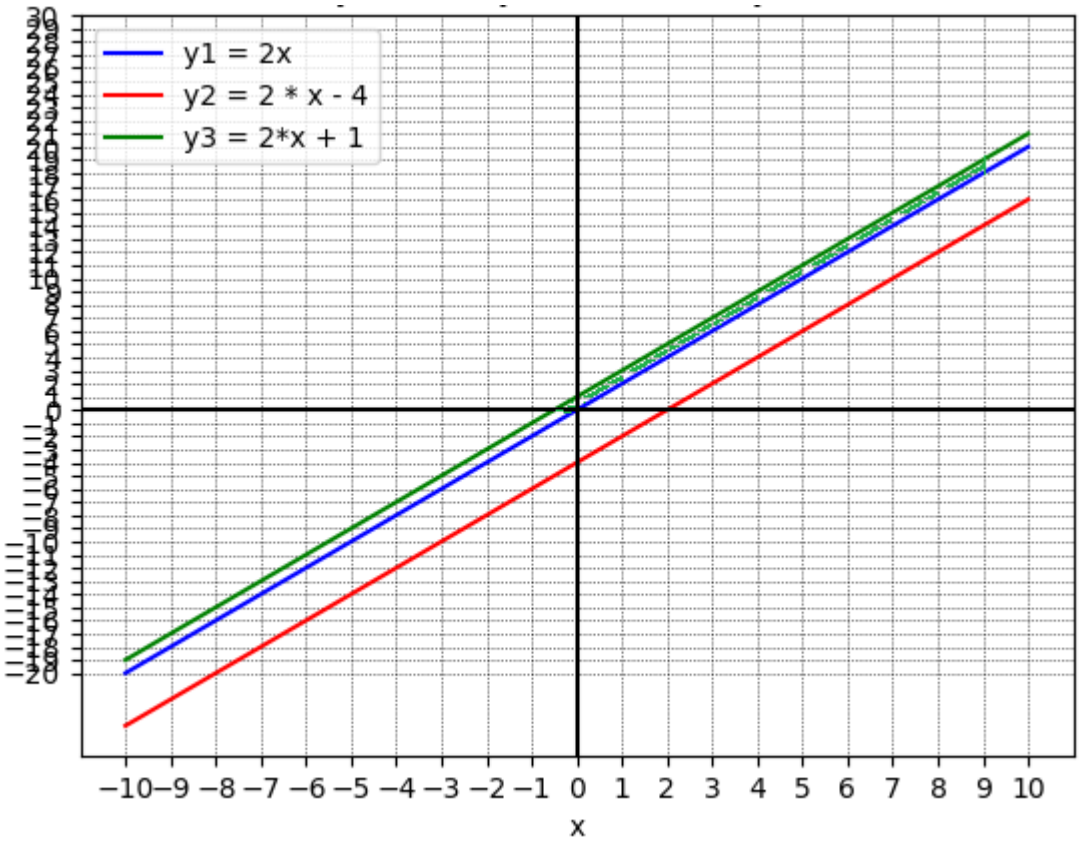

FIGURE 4.7 – Future Z_2

5. L'ensemble d'attente est maintenant vide, cela signifie que le r-TA n'a pas atteint d'état final. Dans ce cas, l'algorithme retourne l'affichage de l'ensemble visités.

Alors, a la fin en obtenu automate de zone dans "lafigure 4.8" :

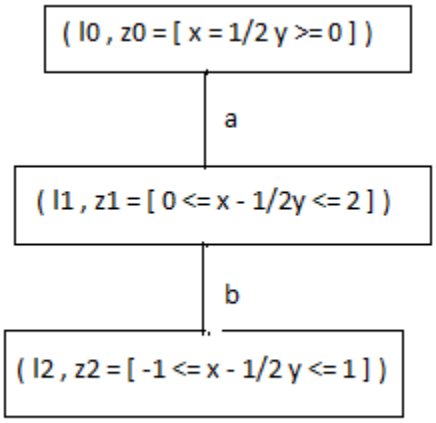

FIGURE 4.8 – Automate de zone de r-TA

4.6 Conclusion

En conclusion, notre contribution d'un automate de zone avec temps relatif représente une avancée significative dans le domaine de l'automatisation industrielle. En exploitant la puissance de la synchronisation en temps réel, notre système offre des performances optimales, une flexibilité accrue et une amélioration globale de l'efficacité des opérations industrielles. Nous sommes convaincus que cette approche trouvera des applications dans divers domaines, tels que les réseaux de capteurs, les systèmes de contrôle industriel et les véhicules autonomes, où la réactivité en temps réel est essentielle pour assurer un fonctionnement optimal et sûr.

Chapitre 5

Conception et implémentation

Sommaire

5.1 Introduction 63
5.2 Langage de modélisation UML 64
5.2.1 A quoi sert UML ? 64
5.2.2 Diagrammes de classes 64
5.3 Langage de développement java 67
5.3.1 Présentation du java 67
5.3.2 Caractéristiques de base du langage Java 67
5.4 Outils de développement 67
5.4.1 L'environnement Netbeans 67
5.4.2 L'environnement Overleaf 68
5.4.3 Pacestar UML Diagrammer 69
5.5 Présentation du l'application 69
5.5.1 Interface principale 69
5.5.2 Interface du résultat 70
5.6 Conclusion 71

5.1 Introduction

Dans ce chapitre, nous allons présenter les outils Hardware et Software (le langage java,uml,...) utilisés pour le développement et la création de notre application.
Alors on a présenté le paradigme de programmation nous avons choisi les outils Hardware , en utilise deux micro-portable :

1. Pour le premier ordinateur portable :
   - Micro portable DELL processeur Intel(R) Celeron(R) 2975U @ 1,40 GHz 1,40 GHz,
   - un SE (Windows 7) installé,64 bits,
   - La RAM 2,00 GO.
2. Pour le second ordinateur portable :
   - Micro portable HP processeur Intel(R) Core(TM) i7 CPU L640 @ 2,13 GHz 2,13GHz,
   - un SE (Windows 7) installé,64 bits,

— La RAM 4,00 GO.

On a essayé de présenter des diagramme sur l'application ainsi quelques capture de notre travail.

5.2 Langage de modélisation UML

5.2.1 A quoi sert UML ?

UML n'est pas une méthode ou un processus !

— Si l'on parle de méthode objet pour UML, c'est par abus de langage !

— Ce constat vaut aussi pour OMT ou d'autres techniques / langages de modélisation

— Une méthode propose aussi un processus, qui régit notamment l'enchaînement des activités de production d'une entreprise.

— UML a été pensé pour permettre de modéliser les activités de l'entreprise, pas pour les régir (ce n'est pas CMM ou SPICE)

— ... (52)

UML est un langage pseudo-formel

— UML est fondé sur un métamodèle, qui définit : les éléments de modélisation (les concepts manipulés par le langage) , et la sémantique de ces éléments (leur définition et le sens de leur utilisation).

— Un métamodèle est une description très formelle de tous les concepts d'un langage. Il limite les ambiguïtés et encourage la construction d'outils.

— ... (52)

UML cadre l'analyse objet, en offrant

— différentes vues (perspectives) complémentaires d'un système, qui guident l'utilisation des concept objets

— plusieurs niveaux d'abstraction, qui permettent de mieux contrôler la complexité dans l'expression des solutions objets.

— ... (52)

5.2.2 Diagrammes de classes

Un diagramme de classes est une collection d'éléments utilisés pour la modélisation statique d'un ensemble d'objets. En d'autres termes, le diagramme de classes fait abstraction des aspects dynamiques et temporels d'un modèle. Naturellement au cas où le modèle est complexe et fastidieux à établir, la nécessité d'utiliser des diagrammes de classes complémentaires s'impose. En général, lors de l'écriture du modèle, le concepteur se focalise plutôt sur les classes qui participent à un cas d'utilisation (use case), à celles qui sont associées à la réalisation d'un scénario précis, à celles qui composent un paquetage et d'une façon globale,

une attention particulière sera donnée en premier lieu à la structure hiérarchique d'un ensemble de classes. Si l'on considère deux classes appartenant au même diagramme, ces deux classes peuvent être reliées par une connexion sémantique bidirectionnelle qui est peut être instanciable sous forme de liens. Dans le même sens, on définit ce que l'on appelle le ou les rôles qui spécifient la fonction d'une classe pour une association donnée (indispensable pour les associations réflexives).

Diagramme de classes de l'automate temporisé avec temps relatif

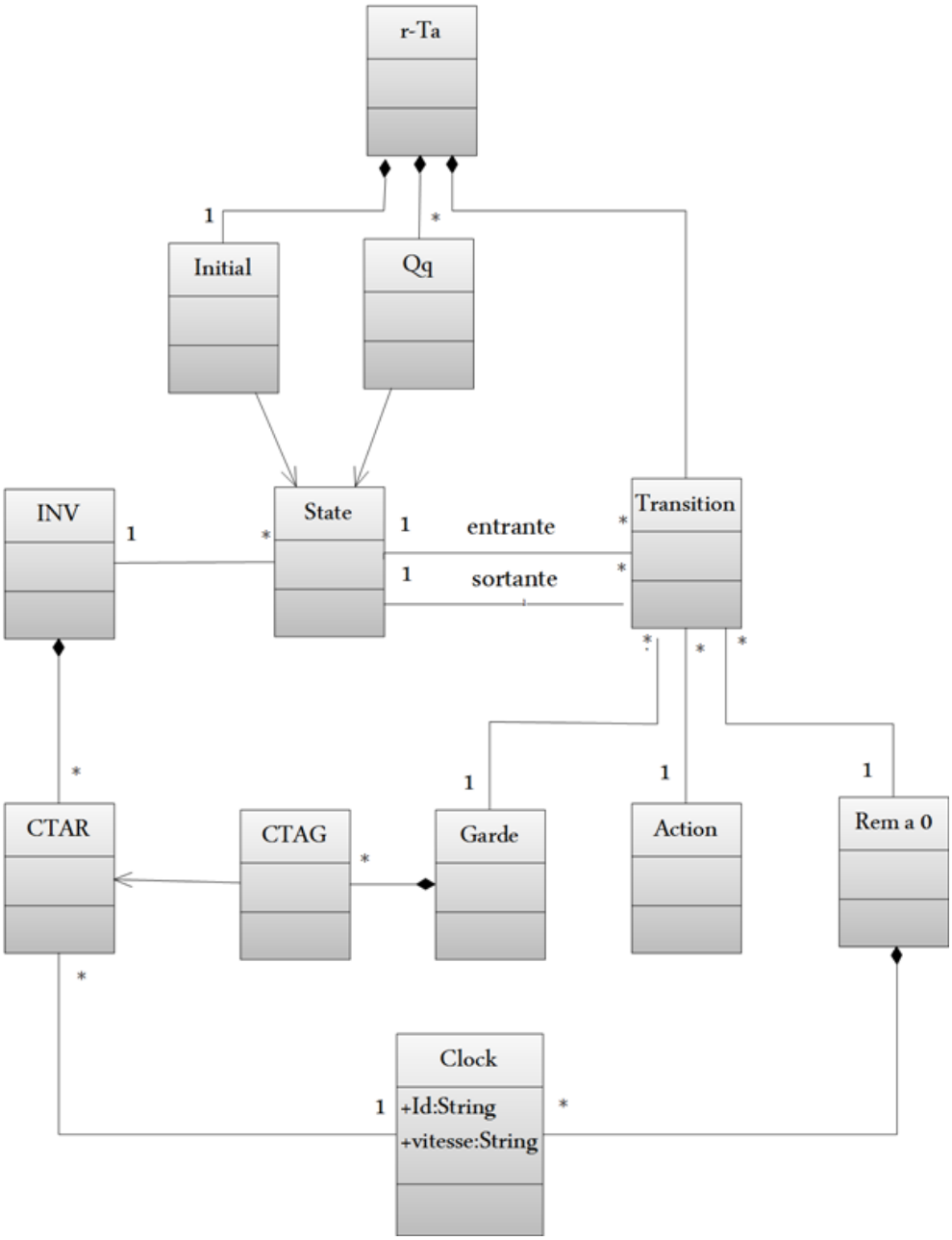

FIGURE 5.1 – Diagramme du r-TA

Diagramme de classes de l'automate de zones

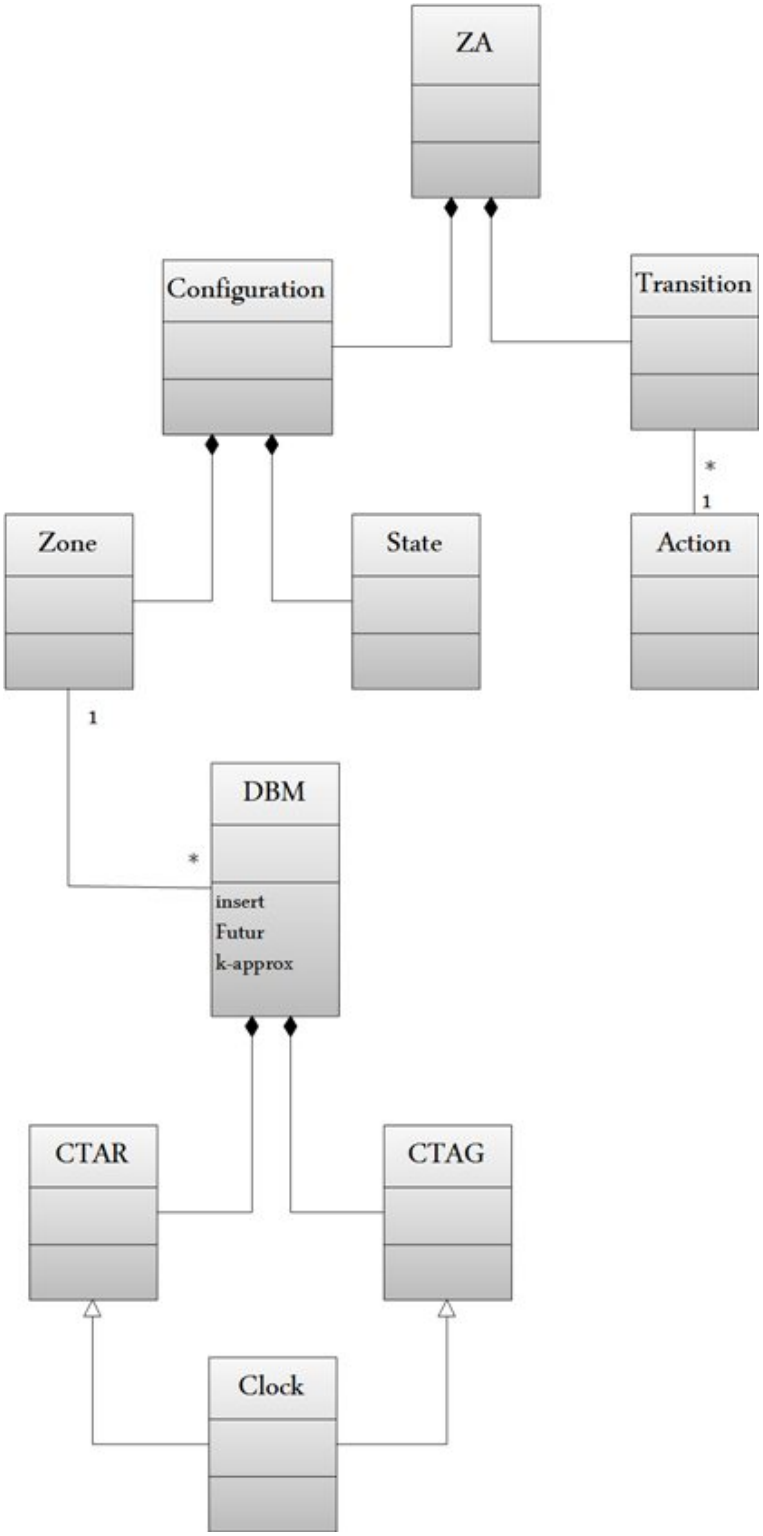

FIGURE 5.2 – Diagramme de classes de l'automate de zones

5.3 Langage de développement java

5.3.1 Présentation du java

Java est un langage de programmation orienté objet à usage général créé par James Gosling et d'autres personnes de Sun Microsystems au milieu des années 1990. Il a joué un rôle clé dans la transformation de l'internet et a fondamentalement changé la façon dont les gens programment. Il prend en charge la programmation d'applications dans un environnement informatique distribué et a été conçu dans un souci de portabilité. L'objectif est de créer un environnement informatique qui permette d'écrire un programme une seule fois et de l'exécuter n'importe où. L'indépendance de la plate-forme est obtenue en compilant le code Java en bytecode Java, où le bytecode est un ensemble optimisé d'instructions qui peuvent être exécutées sur une machine virtuelle Java.(51)

FIGURE 5.3 – Logo JAVA

5.3.2 Caractéristiques de base du langage Java

Lors de la création du langage Java, il avait été décidé que ce langage devait répondre à 5 objectifs :

1. utiliser une méthodologie orientée objet,
2. permettre à un même programme d'être exécuté sur plusieurs systèmes d'exploitation différents,
3. pouvoir utiliser de manière native les réseaux informatiques,
4. pouvoir exécuter du code distant de manière sûre,
5. être facile à utiliser et posséder les points forts des langages de programmation orientés objet comme le C++ .(en Années and Web)

5.4 Outils de développement

5.4.1 L'environnement Netbeans

NetBeans est un environnement de développement intégré qui constitue la base des outils de développement Java de Sun. Il s'agit de l'un des premiers projets open source lancés par Sun en juin 2000. Sun est le principal contributeur de NetBeans et le seul contributeur HCI. La plus grande réalisation du groupe NetBeans HCI a été d'établir un document de

spécification d'interface utilisateur dans le cadre du processus de développement . Ces documents décrivent maintenant l'interaction détaillée et la conception visuelle de la majorité des nouvelles fonctionnalités. Le groupe HCI de Sun réalise régulièrement des études de convivialité de NetBeans et publie les résultats. Les problèmes de convivialité sont traités avec la même importance que les bogues de fonctionnalité et sont suivis dans la même base de données. Les graphiques tels que les écrans d'accueil, les icônes et autres illustrations constituent une grande partie du projet et sont fournis exclusivement par les concepteurs de Sun sous une licence de source ouverte. La contribution de Sun à l'expérience utilisateur comprend également l'accessibilité, la localisation en japonais et en chinois simplifié, ainsi que la documentation.(7)

5.4.2 L'environnement Overleaf

Overleaf est un éditeur collaboratif en ligne pour la rédaction de documents scientifiques, tels que des articles et des thèses. Il simplifie et accélère le processus de rédaction et de publication scientifiques en conservant le document en un seul endroit central tout au long de son cycle de vie. Le document est stocké en toute sécurité dans le nuage, de sorte que les auteurs, les éditeurs, les réviseurs et les lecteurs peuvent tous lire, modifier ou commenter le document quand c'est leur tour, en utilisant uniquement un navigateur web. Overleaf prend en charge le suivi des modifications, les commentaires, le contrôle des versions et plusieurs gestionnaires de références populaires, et vous pouvez désormais soumettre directement depuis Overleaf à plus d'une douzaine de partenaires d'édition, dont PeerJ, Nature Scientific Reports et F1000Research. Plus de 200 000 auteurs de plus de 2 000 universités à travers le monde ont créé plus de deux millions de documents avec Overleaf, et les bibliothèques de l'université de Stanford ont lancé un essai d'Overleaf à l'échelle de l'institution pour 2015. Overleaf apporte le processus de rédaction et de publication scientifique dans le nuage, où il peut être rendu plus facile, plus rapide et plus ouvert.(45)

FIGURE 5.4 – Logo Overleaf

5.4.3 Pacestar UML Diagrammer

Pacestar UML Diagrammer est une alternative moins chère à d'autres programmes populaires qui aident les programmeurs, les ingénieurs système ou d'autres professionnels à créer des diagrammes UML. Il est doté d'un grand nombre de fonctionnalités, ainsi que d'options de personnalisation, et comprend tous les outils nécessaires pour aider les utilisateurs à concevoir des diagrammes à des fins diverses.

FIGURE 5.5 – Logo Pacestar UML Diagrammer

5.5 Présentation du l'application

5.5.1 Interface principale

Dans la figure 5.6 nous avons utilisé l'application pour implémenter l'exemple précédent dans la sous-section 4.5.5

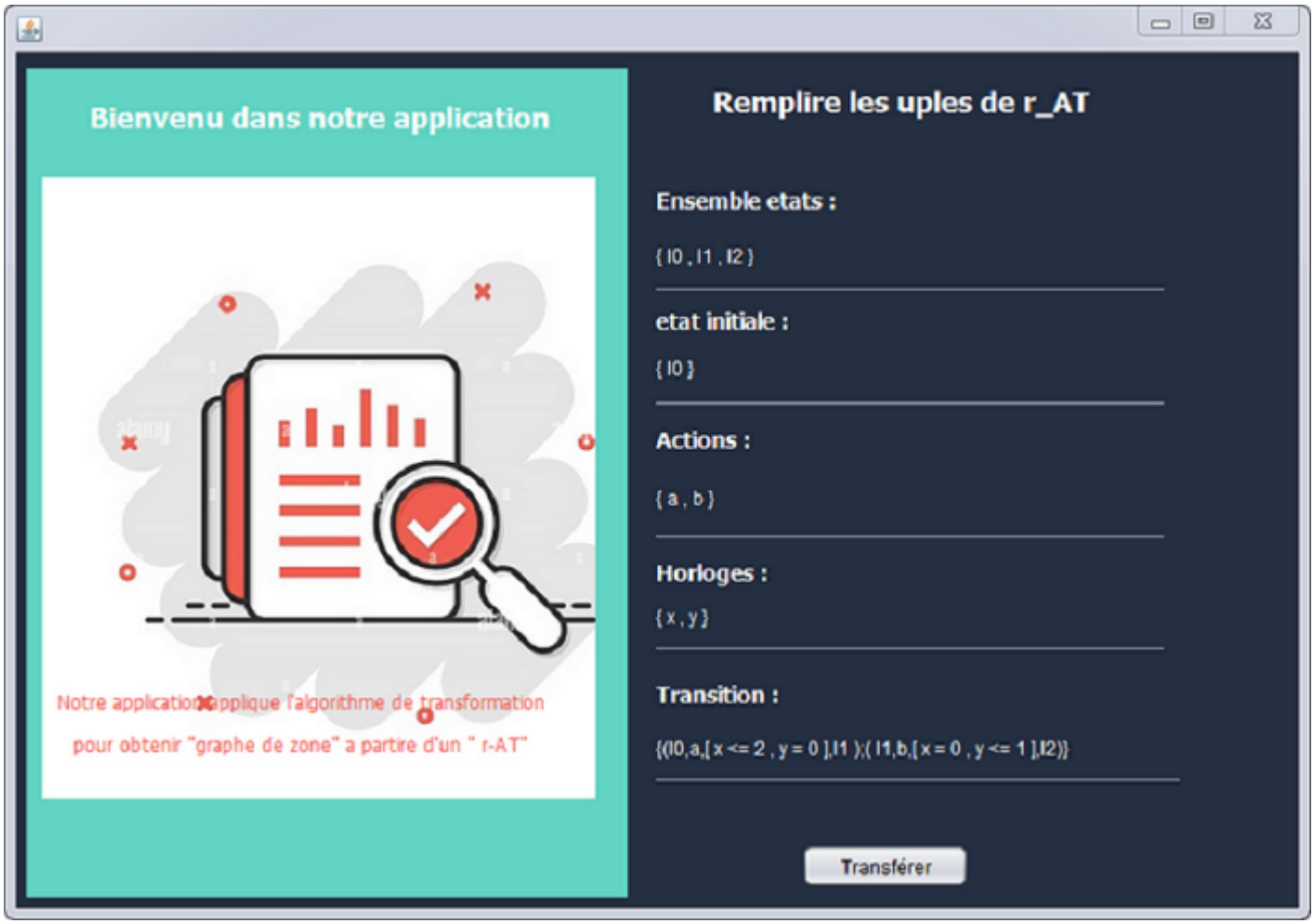

FIGURE 5.6 – Interface principale avec un exemple

5.5.2 Interface du résultat

Après utilisation de l'application pour trouver le graphe de zone de l'exemple de sous-section4.5.5 , la figure 5.7 est montre

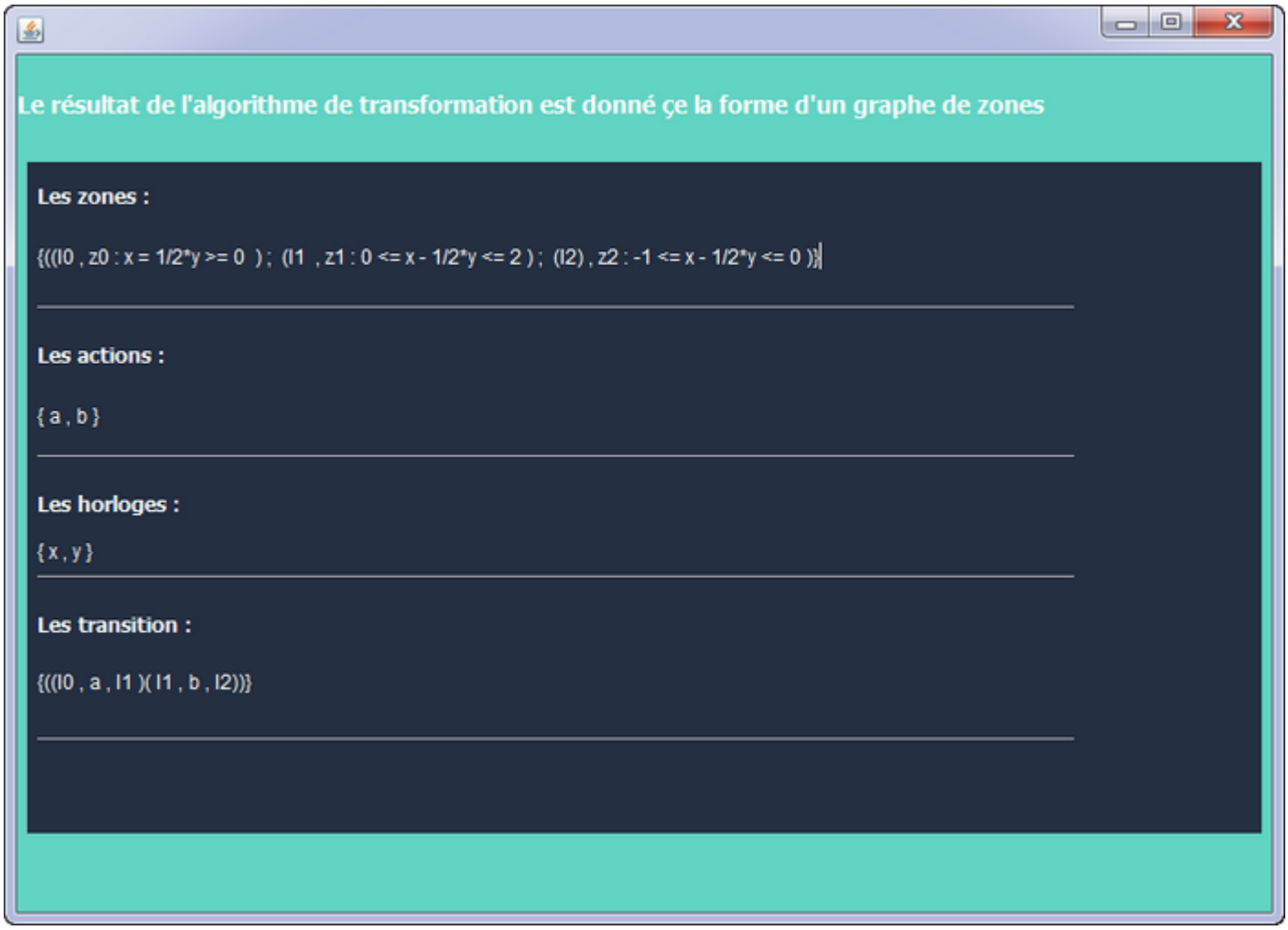

FIGURE 5.7 – Interface de résultat

5.6 Conclusion

Dans ce chapitre nous avons présenté Les outils Hardware (les deux micro-portable) et Software (langage java , neatbeans , overleaf , .U .) utilisés dans notre mémoire.
De plus, il introduit brièvement l'application étudiée, en mettant en avant son interface principale et interfacedu resultat avec un exemple .

Chapitre 6

Conclusion générale

L'objectif principal de ce travail était de participer au développement de modèles et d'algorithmes pour la modélisation et la vérification de systèmes critiques où la réactivité et l'hétérogénéité doivent être prises en compte.

La vérification formelle revêt une importance capitale pour garantir leur validité et leur conformité aux spécifications. Parmi les différentes méthodes de vérification, le model-checking est largement reconnu comme l'une des approches les plus efficaces car requiert une représentation finie de la structure du système ainsi que des propriétés à vérifier.

Dans notre mémoire, nous avons développé un algorithme de transformation des automates temporisés avec temps relatif r-TA en des structures finies appellées automates de zones qui sont utiles à la vérification par model-checking.

Il est à noter que d'autres approches existent, telles que l'utilisation d'automates de régions. Cependant, nous avons écarté cette solution en raison de sa complexité et du temps considérable qu'elle demande.

En conclusion, nous avons concentré dans ce travail sur la transformation d'un automate temporisé avec temps relatif r-TA vers un automate de zones, afin de faciliter la vérification formelle des systèmes temps réel critiques et hétérogènes. Cette approche c'est avérée pertinente, car elle permet de modéliser de manière finie les comportements du système avec des aspects temporels relatifs. Grâce à cette transformation, nous avons pu appliquer des techniques de model-checking pour vérifier les propriétés souhaitées et assurer la fiabilité des systèmes critiques hétérogènes.

Bibliographie

[1] 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.

[2] Alur, R. and Dill, D. (1990). Automata for modeling real-time systems. In *Automata, Languages and Programming : 17th International Colloquium Warwick University, England, July 16–20, 1990 Proceedings 17*, pages 322–335. Springer.

[3] ALUR, R. and DILL, D. (1994.). A theory of timed automata. *Theoretical Computer Science*, vol. 126(2) :183–235.

[4] Alur, R., Kurshan, R., and Viswanathan, M. (1998). Membership problems for timed and hybrid automata. In *19th IEEE Real-Time Systems Symposium*.

[5] Azzedine, A. (2004). *Outil d'analyse et de partitionnement-ordonnancement pour les systèmes temps réels embarqués*. PhD thesis, Lorient.

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

[7] Benson, C., Muller-Prove, M., and Mzourek, J. (2004). Professional usability in open source projects : Gnome, openoffice. org, netbeans. In *CHI'04 extended abstracts on Human factors in computing systems*, pages 1083–1084.

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

[9] Billet, B. (2015). *Système de gestion de flux pour l'Internet des objets intelligents*. PhD thesis, Université de Versailles-Saint Quentin en Yvelines.

[10] Bouyer, P. (2002). *Timed automata may cause some troubles*. BRICS.

[11] Bouyer, P. (2003). Untameable timed automata ! In *STACS*, volume 2607, pages 620–631. Springer.

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

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

[14] Bozga, M., Daws, C., Maler, O., Olivero, A., Tripakis, S., and Yovine, S. (1998). Kronos : A model-checking tool for real-time systems : Tool-presentation for ftrtft'98. In *Formal Techniques in Real-Time and Fault-Tolerant Systems : 5th International Symposium, FTRTFT'98 Lyngby, Denmark, September 14–18, 1998 Proceedings 5*, pages 298–302. Springer.

[15] Buck, J., Ha, S., Lee, E. A., and Messerschmitt, D. G. (2001). Ptolemy : A framework for simulating and prototyping heterogeneous systems. In *Readings in hardware/software co-design*, pages 527–543.

[16] Charette, R. N. (1991). *Applications strategies for risk analysis*. McGraw-Hill, Inc.

[17] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (1990). 33.3 : Finding the convex hull. *Introduction to Algorithms*, pages 955–956.

[18] Cottet, F. and Grolleau, E. (2014). *Systèmes temps réel embarqués-2e éd.-Spécification, conception, implémentation et validation tem : Systèmes temps réel embarqués*. Dunod.

[19] Crow, J., Owre, S., Rushby, J., Shankar, N., and Srivas, M. (1995). A tutorial introduction to pvs. Wift.

[20] Dasarathy, B. (1985). Timing constraints of real-time systems : Constructs for expressing them, methods of validating them. *IEEE transactions on Software Engineering*, 11(1) :80–86.

[21] Daws, C. (1997). Analyse par simulation symbolique de syst emes temporis es avec kronos.

[22] Daws, C. and Tripakis, S. (1998). Model checking of real-time reachability properties using abstractions. In *TACAS*, volume 98, pages 313–329.

[23] Diab, H. (2002). *Évaluation de méthodes formelles de spécification*. National Library of Canada= Bibliotheque nationale du Canada, Ottawa.

[24] Dill, D. L. (1990). Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems : International Workshop, Grenoble, France June 12–14, 1989 Proceedings 1*, pages 197–212. Springer.

[en Années and Web] en Années, A. and Web, S. Java (langage).

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

[27] Guellati, S. (2018). *Logic Verification of Real Time Systems in the Context of the Maximality Semantics*. PhD thesis, Laboratoire MISC, Université Abdelhamid Mehri - Constantine 2, 25000 Constantine, Algérie.

[28] Henzinger, T. A., Ho, P.-H., and Wong-Toi, H. (1995). Hytech : the next generation. In *Proceedings 16th IEEE Real-Time Systems Symposium*, pages 56–65. IEEE.

[29] Henzinger, T. A., Ho, P.-H., and Wong-Toi, H. (1997). Hytech : A model checker for hybrid systems. In *Computer Aided Verification : 9th International Conference, CAV'97 Haifa, Israel, June 22–25, 1997 Proceedings 9*, pages 460–463. Springer.

[30] 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.

[31] Hildenbrand, D., Albert, J., Charrier, P., and Steinmetz, C. (2017). Geometric algebra computing for heterogeneous systems. *Advances in Applied Clifford Algebras*, 27 :599–620.

[32] HOPCROFT, J. E. and ULLMAN, J. D. (1979.). Introduction to automata theory, languages and computation. *Addison-Wesley,*.

[33] Hu, H., Zhou, M., and Li, Z. (2009). A new class of petri nets for modeling and control of ratio-enforced resource allocation systems. In *2009 IEEE International Conference on Systems, Man and Cybernetics*, pages 199–204. IEEE.

[34] Huet, G., Kahn, G., and Paulin-Mohring, C. (1997). The coq proof assistant a tutorial. *Rapport Technique*, 178.

[35] Hutzler, G., Klaudel, H., and Wang, D. Y. (2004). Automates temporisés et systèmes multi-agents temps-réel. In *JFSMA*, pages 69–82.

[36] Iyer, R. K. and Velardi, P. (1985). Hardware-related software errors : measurement and analysis. *IEEE Transactions on Software Engineering*, 11(2) :223–231.

[Kesraoui] Kesraoui, S. M. Intégration des techniques de vérification formelle dans une approche de conception des systèmes de contrôle-commande.

[38] Krichen, M. and Tripakis, S. (2009). Conformance testing for real-time systems. *Formal Methods in System Design*, 34(3) :238–304.

[39] Kristoffersen, K. J., Laroussinie, F., Larsen, K. G., Pettersson, P., and Yi, W. (1997). A compositional proof of a real-time mutual exclusion protocol. In *TAPSOFT'97 : Theory and Practice of Software Development : 7th International Joint Conference CAAP/FASE Lille, France, April 14–18, 1997 Proceedings 22*, pages 565–579. Springer.

[40] Laroussinie, F. and Larsen, K. G. (1998). Cmc : A tool for compositional model-checking of real-time systems. In *Formal Description Techniques and Protocol Specification, Testing and Verification : FORTE XI/PSTV XVIII'98 IFIP TC6 WG6. 1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XI) and Protocol Specification, Testing and Verification (PSTV XVIII) 3–6 November 1998, Paris, France*, pages 439–456. Springer.

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

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

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

[44] Lee, I. and Smolka, S. A. (1995). Concur'95 : Concurrency theory 6th international conference philadelphia, pa, usa, august 21–24, 1995 proceedings. In *Conference proceedings CONCUR*, page 557. Springer.

[45] Lees-Miller, J. D. (2015). Overleaf : Scientific writing and publishing in the age of the cloud. In *PKP Scholarly Publishing Conference 2015*.

[46] LIPN, K. K. (2013). *Abstraction et Analyse de l'Espace des États des Réseaux de Petri Temporels*. PhD thesis, Université Pierre et Marie Curie.

[47] Maria, M. R. D. (2022). Vérification automatique basée-modèles des systèmes temps-réel hétérogènes. Mémoire de fin d'études, Université 20 Aout-1955-SKIKDA. Mémoire fin d'étude en vue de l'obtention du diplôme Master en Informatique, Spécialité : Réseaux et Systèmes Distribués (RSD).

[48] Mokadem, H. B. (2006). *Vérification des propriétés temporisées des automates programmables industriels*. PhD thesis, École normale supérieure de Cachan-ENS Cachan.

[49] Mouradian, A. (2013). *Proposition et vérification formelle de protocoles de communications temps-réel pour les réseaux de capteurs sans fil*. PhD thesis, INSA de Lyon.

[50] Myers, W. (1986). Can software for the strategic defense initiative ever be error-free ? *Computer*, 19(11) :61–67.

[51] O'Regan, G. and O'Regan, G. (2018). Java programming language. *The Innovation in Computing Companion : A Compendium of Select, Pivotal Inventions*, pages 171–174.

[52] Piechocki, L. (2007). Uml, le langage de modélisation objet unifié. *Laurentpiechnocki. developpez. com*.

[53] Reynier, P.-A. (2004). *Analyse en avant des automates temporisés*. PhD thesis, Master's thesis, DEA Algorithmique, Paris.

[54] Rushby, J. (2000). From refutation to verification. In *Formal Methods for Distributed System Development : FORTE/PSTV 2000 IFIP TC6 WG6. 1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XIII) and Protocol Specification, Testing and Verification (PSTV XX) October 10–13, 2000, Pisa, Italy*, pages 369–374. Springer.

[55] Rushby, J. (2001). Theorem proving for verification. In *Modeling and Verification of Parallel Processes : 4th Summer School, MOVEP 2000 Nantes, France, June 19–23, 2000 Revised Tutorial Lectures*, pages 39–57. Springer.

[56] Schapachnik, F., Braberman, V., and Olivero, A. (2002). An architecture-centric approach to the development of a distributed model-checker for timed automata. In *Proceedings of the 24th International Conference on Software Engineering*, pages 710–710.

[57] Stankovic, J. A. (1988). Misconceptions about real-time computing : A serious problem for next-generation systems. *Computer*, 21(10) :10–19.

[58] Upegui, A., Thoma, Y., Satizábal, H. F., Mondada, F., Rétornaz, P., Graf, Y., Perez-Uribe, A., and Sanchez, E. (2010). Ubichip, ubidule, and marxbot : a hardware platform for the simulation of complex systems. In *Evolvable Systems : From Biology to Hardware : 9th International Conference, ICES 2010, York, UK, September 6-8, 2010. Proceedings 9*, pages 286–298. Springer.

[59] Van Baelen, S., Ober, I., Espinoza, H., Weigert, T., Ober, I., and Gérard, S. (2011). Model based architecting and construction of embedded systems (aces-mb 2010). In *Models in Software Engineering : Workshops and Symposia at MODELS 2010, Oslo, Norway, October 2-8, 2010, Reports and Revised Selected Papers 13*, pages 70–74. Springer.

[60] Yovine, A. O. S. and rue Lavoisier, M.-Z. (1993). Kronos : A tool for verifying real-time systems user's guide and reference manual draft 0.0.

[61] Yovine, S. (1997). Kronos : A verification tool for real-time systems. *Int. J. Softw. Tools Technol. Transf.*, 1(1-2) :123–133.

[62] Yovine, S. (1998). Model checking timed automata. In *Lectures on Embedded Systems : European Educational Forum School on Embedded Systems Veldhoven, The Netherlands November 25–29, 1996*, pages 114–152. Springer.

[63] Zerhouni, F. Z., Zerhouni, M., Zegrar, M., Benmessaoud, M. T., Stambouli, A. B., and Midoun, A. (2010). Proposed methods to increase the output efficiency of a photovoltaic (pv) system. *Acta polytechnica hungarica*, 7(2) :55–70.