Conception, modélisation et vérification formelle d'un système temps-réel d'agents coopératifs : application aux véhicules autonomes communicants

Johan Arcile,

▶ **To cite this version:**

Johan Arcile,. Conception, modélisation et vérification formelle d'un système temps-réel d'agents coopératifs : application aux véhicules autonomes communicants. Calcul formel [cs.SC]. Université Paris-Saclay, Université d'Evry Val-d'Essonne, 2019. Français. NNT : 2019SACLE029 . tel-02480653

HAL Id: tel-02480653
https://hal.science/tel-02480653

Submitted on 16 Feb 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

université PARIS-SACLAY

NNT : 2019SACLE029

Thèse de doctorat

Conception, modélisation et vérification formelle d'un système temps-réel d'agents coopératifs

Application aux véhicules autonomes communicants

Thèse de doctorat de l'Université Paris-Saclay
préparée à Université d'Evry-Val-d'Essonne

École doctorale n°580 Sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Evry, le 13 décembre 2019, par

JOHAN ARCILE

Composition du Jury :

Saïd Mammar Professeur, Université d'Evry-Val-d'Essonne (IBISC)	Président
Olivier H. Roux Professeur, École Centrale de Nantes (LS2N)	Rapporteur
Catalin Dima Professeur, Université Paris-Est Créteil (LACL)	Rapporteur
Kaïs Klai Maître de Conférences HDR, Université Paris 13 (LIPN)	Examinateur
Hanna Klaudel Professeur, Université d'Evry-Val-d'Essonne (IBISC)	Directrice de thèse

Remerciements

Ce manuscrit n'aurait pu voir le jour sans le soutien de longue date de ma directrice de thèse, Hanna Klaudel, qui m'a fait entrer dans l'univers de la recherche. Je la remercie chaleureusement pour tout ce que notre travail ensemble m'a apporté, professionnellement comme personnellement.

Merci à Raymond Devillers pour sa collaboration toujours appréciée et riche de savoir faire. Son intérêt constant pour mes travaux est à mes yeux un gage de leur qualité.

Je remercie également Jérémy Sobieraj, Guillaume Hutzler, Bożena Woźna-Szcześniak, Witold Klaudel et Lydie Nouvelière avec qui j'ai eu la chance de travailler dans le cadre de ma thèse.

Merci aux membres du jury, Olivier H. Roux, Catalin Dima, Kaïs Klai et Saïd Mammar, pour leur disponibilité. Je souhaite souligner en particulier le rôle des rapporteurs du présent manuscrit, Olivier H. Roux et Catalin Dima. Leur lecture minutieuse et les remarques pertinentes qu'ils m'ont transmises ont été d'une grande aide.

Je tiens à exprimer ma gratitude envers les enseignants du département informatique de l'Université d'Évry, qui m'ont appris à aimer cette science et à la pratiquer. Ils sont devenus par la suite des collègues que j'ai eu plaisir à cotoyer durant ces trois dernières années. Je remercie particulièrement les membres de l'équipe Cosmo pour leur bienveillance et leur sympathie.

J'ai également une pensée pour tous mes camarades de thèse, ceux qui sont aujourd'hui docteurs comme ceux qui le deviendront bientôt. Leur présence et leur amitié ont pour moi beaucoup comptées.

Je remercie enfin ma famille et mes proches pour leurs encouragements continus.

Table des matières

Introduction générale **7**

1 Contexte et motivation **13**
1.1 Approche pour l'évaluation du système 15
1.2 Caractéristiques du système 16
1.3 Comparaison avec des approches existantes 20

2 Notions préliminaires et état de l'art **23**
2.1 Notions préliminaires sur les graphes 23
2.2 Les techniques de vérification formelle 25
2.3 Les formalismes de modélisation 29
2.3.1 Réseaux d'automates temporisés 29
2.3.2 Réseaux de Petri de haut niveau 32
2.3.3 Automates hybrides 33

3 Le cadre logiciel VerifCar **35**
3.1 Le modèle 36
3.1.1 Les composants 36
3.1.2 Choix de modélisation, réglages et précision 39
3.1.3 Structure de données et détail des fonctions 42
3.2 Méthodologie de vérification 45
3.2.1 Indicateurs utilisés pour l'analyse comportementale 46
3.2.2 Méthodologie de l'analyse 50
3.3 Utilisation du cadre logiciel 52
3.3.1 L'algorithme de décision 53
3.3.2 Impact des communications sur le comportement des CAVs 54
3.3.3 Impact de la coopération sur le comportement des véhicules 59
3.3.4 Limitations et gestion des paramètres 62

3.4 Résumé du chapitre 65

4 Exploration de MAPTs **67**
4.1 Le formalisme des MAPTs 69
4.1.1 Syntaxe et sémantique des G-MAPTs 69
4.1.2 Traduction en réseaux de Petri de haut niveau 77
4.1.3 Accélération 79
4.2 Algorithmes d'exploration 86
4.2.1 Espace d'états en couches 87
4.2.2 Exploration basée sur les variables fortes et faibles 96
4.2.3 Exploration «à la volée» d'un MAPT 99
4.3 Comparaison de performances 100
4.3.1 Efficacité de la dynamique accélérée. 101
4.3.2 Efficacité de l'exploration par couches. 102
4.3.3 Heuristiques 103
4.3.4 Comparaison avec VERIFCAR 105
4.4 Résumé du chapitre 107

5 Approche combinée avec la simulation **111**
5.1 Choix de l'outil de simulation 113
5.2 Etude d'un modèle de suivi de voiture : IDM 115
5.3 Comparaisons des valeurs des indicateurs 116
5.4 Application de la méthode 118
5.5 Résumé du chapitre 120

Conclusion **123**

A Coopération par négociation **127**

B Coopération par infrastructures **129**

C Algorithmes des requêtes pour l'exploration des MAPTs **131**

Bibliographie **135**

Introduction

Au matin du 4 juin 1996 au centre spatial de Kourou, à lieu le premier lancement de la fusée Ariane 5, aboutissement de près de 10 ans de travail de la part de l'Agence spatiale européenne (ESA). Cependant, l'engin n'atteindra jamais l'espace : il explosera à quatre kilomètres du sol, après moins d'une minute de vol. Cet évènement est bien connu des informaticiens, car c'est un problème logiciel qui déclencha la série de dysfonctionnements conduisant à l'auto-destruction de la fusée. En l'occurrence, le problème logiciel consistait en un *overflow*, une des causes les plus fréquentes d'erreurs dans un programme informatique. En informatique, une variable permet de stocker des nombres, chaque variable ayant un ensemble fini de nombres qu'elle est en mesure de stocker. Un *overflow* consiste a essayer de stocker dans une variable un nombre ne faisant pas partie de cet ensemble. Cela conduit en général à l'arrêt du programme, ou à des calculs erronés, comme ce fût le cas pour le vol d'Ariane 5.

La série d'événements conduisant à l'explosion a pu être retrouvée après coup à l'aide de la technique informatique de l'interprétation abstraite, qui permet d'analyser toutes les exécutions possibles d'un programme informatique afin de pouvoir vérifier de manière sûre des propriétés de ce programme. Cette technique aurait pu être utilisée avant le lancement pour vérifier si un *overflow* était possible ou non. Néanmoins, ce genre de vérifications n'étaient pas utilisées à cette époque au cours du processus de conception de systèmes critiques. L'accident a permis d'attirer l'attention sur les risques liés à l'utilisation de systèmes informatiques complexes dans le cadre de systèmes critiques, tels que les fusées ou les avions, mais également les machineries médicales. L'analyse automatique des programmes de la fusée Ariane, bien qu'effectuée après l'événement, était le premier cas où les méthodes formelles étaient utilisée pour vérifier le fonctionnement de systèmes critiques dans le cadre d'un projet de grande envergure. Aujourd'hui, les méthodes formelles de vérification automatique de systèmes informatiques sont largement utilisées lors de la conception de systèmes de ce type [1, 2].

L'intitulé du présent manuscrit indique que les travaux qui y sont décrits portent sur trois points distincts, bien que fortement dépendant les uns des autres : la **conception** d'un système, sa **modélisation**, et sa méthode de **vérification formelle**.

1. La **conception** du système constitue la première étape d'un processus visant à analyser un

comportement donné. Elle consiste, au vu des questions auxquelles on souhaite pouvoir répondre concernant le comportement du système, à identifier ce qu'il est important ou non de représenter. Il convient ensuite de spécifier de manière non ambiguë ce que l'on a choisit de représenter. Cela consiste, entre autre, à définir le niveau d'abstraction adéquat auquel on souhaite se positionner. En effet, certains détails seront considérés non significatifs, tandis que d'autres auront de l'importance au vu de l'analyse que l'on souhaite effectuer.

2. La **modélisation** a pour objectif de permettre l'analyse par ordinateur d'un système donné. Produire un modèle d'un système nécessite le choix d'un formalisme adéquat. Ici, on désigne par formalisme un langage pourvu d'une sémantique, laquelle permet de calculer l'ensemble des comportements possibles d'un système décrit par le langage. Le choix d'un formalisme dépend autant des outils permettant l'analyse de modèles ainsi formalisés (chaque formalisme disposant d'un panel d'outils qui lui est propre), que du type de système à modéliser (chaque formalisme étant adapté à différents types de systèmes). Il convient également de prendre en compte le type de propriétés que l'on souhaite vérifier, car chaque outil est limité au niveau de l'expressivité de ces propriétés.

3. La **vérification formelle**, aussi appelée vérification de modèle [3], ou *model-checking* en anglais, désigne l'action d'une fonction prenant en entrée un modèle d'une part, et une propriété portant sur les états du modèle d'autre part. L'outil implémentant cette fonction, et donc le modèle, est appelé un *model checker*, et renvoie une réponse positive ou négative, selon que la propriété est vraie sur le modèle ou non. On parle de vérification formelle car le modèle est formel, à savoir non ambigu et donc calculable mathématiquement, et que la propriété est vérifiée de manière exhaustive, à savoir sur l'intégralité des états dans lesquels peut se trouver le système.

Ainsi, la conception, la modélisation et la vérification formelle d'un système forment les trois étapes nécessaire au processus de vérification de modèle. La vérification de modèle est une méthode formelle de résolution de problèmes de décision complexes. Plus spécifiquement, elle permet de vérifier les propriétés comportementales d'un système donné et fournit des exemples de comportements qui respectent ou violent la propriété considérée. L'atout principal de cette méthode est qu'elle prend en compte le non-déterminisme tout en garantissant l'exhaustivité. Cependant, il peut être difficile d'obtenir une réponse en raison du nombre d'états du modèle. En effet, dans la mesure où une formalisation des caractéristiques du système est nécessaire, le modèle résultant est souvent composé d'un très grand nombre d'états (souvent plusieurs milliards). Par exemple, dans le cas de la modélisation de véhicules sur une portion de route, il faut prendre en compte :

— Les variables nécessaires pour exprimer l'état d'un véhicule (position, vitesse, accélération, direction, ...) ;

— Le nombre de possibilités résultant d'une prise de décision non déterministe ;

— Le nombre de véhicules, qui impacte de manière exponentielle le nombre d'états.

Dans un tel contexte, l'ajout d'un agent, voir même d'une variable, peut conduire à une augmentation du nombre d'états de nature exponentielle qui résulte en une augmentation de même nature des temps de calculs et de l'espace mémoire nécessaire à ces calculs. On parle alors d'explosion du nombre d'états. L'utilisation d'approximations permet de contenir ce problème d'explosion en simplifiant certains aspects du modèle. Ces simplifications ont cependant un impact négatif sur la fiabilité du système, l'accumulation d'erreurs (par exemple la discrétisation de nombre réels en nombres entiers) pouvant conduire à un comportement très différent du comportement réel. Ainsi, la vérification de modèle peut être intéressante pour résoudre des problèmes complexes, mais le modèle doit présenter un compromis satisfaisant entre taille de l'espace d'états et réalisme de la représentation.

Problématique soulevée L'objectif ici est d'appliquer la méthode du *model-checking* dans le cadre d'un système **temps-réel** d'**agents coopératifs**. Un système dit **temps-réel** est un système dans lequel les actions doivent respecter des contraintes temporelles. Par exemple, dans un certain état, il faut attendre au moins 2 secondes avant de pouvoir effectuer une action, ou encore, après avoir atteint un certain état du système, une action devra être effectuée dans un intervalle de $[1, 3]$ secondes. Un système dit à **agents coopératifs** est un système multi-agent, c'est à dire où le choix des actions possibles est réparti entre différents agents ayant chacun un point de vu local du système. Ainsi, chaque agent est une entité interagissant avec son environnement sans qu'il n'ait une connaissance globale du système. La somme de toutes les actions choisies par les agents conduit à un comportement émergent, en opposition à un modèle centralisé où les actions sont prises en ayant une connaissance globale de l'état du système. L'aspect coopératif indique ici que les agents ont un but commun et ne sont pas directement des adversaires. Pour ce faire, le cas d'étude des véhicules autonomes coopératifs a été choisi. Il s'agit de représenter plusieurs véhicules prenant en temps réel des décisions sur les manoeuvres à effectuer, et capables d'échanger des informations les uns avec les autres. Le système est ainsi composé de plusieurs agents en mouvement qui évoluent en respectant des règles précises portant sur la vitesse et l'accélération des agents, ainsi que la direction de leur mouvement. Chaque agent perçoit son environnement, c'est-à-dire, les positions des autres agents et leur paramètres de mouvement, et à partir de cette représentation, prend une décision (en temps réel) sur son propre mouvement. Nous souhaitons donc modéliser formellement un tel système et de

mettre en place une méthodologie de vérification automatique permettant d'évaluer exhaustivement des politiques de prise de décision sur différents types d'agents dans des environnements divers. Les méthodes formelles ont cependant des limitations et souffrent du problème d'explosion du nombre d'états et d'une mauvaise capacité de passage à l'échelle. En particulier, pour des systèmes d'agents temps-réel communicant, il n'existe pas de méthode de vérification suffisamment efficace. La problématique soulevée concerne le développement d'une telle méthode qui garantirait une précision et un réalisme suffisants (surtout dans la représentation des trajectoires et de leur prédiction) pour que les résultats obtenus soient exploitables par les concepteurs de véhicules autonomes.

Contributions Les contributions de ce travail sont, par ordre d'apparition dans le manuscrit :

- La spécification du système (Section 1.2). Le système est modulaire et permet l'utilisation d'algorithmes de prise de décision réalistes dans des environnements divers. La spécification prend en compte la particularité de l'approche, qui se concentre sur l'impact des communications sur la prise de décision. En particulier, le caractère non déterministe de la latence est représenté.
- Le cadre logiciel VERIFCAR, constitué par :
  - Une modélisation du système (Section 3.1). Le formalisme utilisé pour le modèle est un réseau d'automates temporisés. Il a comme avantage de permettre une représentation exhaustive d'un système temps réel, tout en ayant la possibilité d'être analysé via des langages de logique temporelle. Il a été nécessaire de définir le bon niveau d'abstraction permettant d'obtenir un compromis entre espace d'états et réalisme. Ces abstractions sont par ailleurs capables de se mettre à l'échelle en fonction des paramètres du modèle (taille de la route, fréquence de la prise de décision, précision souhaitée, $\cdots$).
  - Une méthodologie de vérification (Section 3.2). Une méthodologie de vérification basée sur la logique CTL (Computational Tree Logic) [4] est proposée, l'expressivité de ce langage de requêtes est généralement suffisante pour le type d'informations que nous souhaitons obtenir. On y propose des séquences de requêtes portant sur divers indicateurs. En particulier, l'indicateur du *time-to-collission* (TTC) [5, 6] est adapté à un espace à deux dimensions.
- La formalisation d'une classe de modèles appelée *Multi-Agent with timed Periodic Tasks* (MAPTs), systèmes multi-agent à tâches périodiques, en français, et d'abstractions dédiées à la vérification de propriétés d'accessibilité sur cette classe de modèles. (Chapitre 4). Ce formalisme tire profit des particularités observées sur les systèmes de vé-

hicules autonomes communicants pour construire un environnement de vérification formelle dédié aux problèmes d'accessibilité dans lequel il est possible d'exprimer tout CTL, y compris des requêtes imbriqués (ce que ne peut pas faire le cadre logiciel VERIFCAR, limité en termes d'expressivité). Le modèle et les algorithmes ont été implémentés en utilisant ZINC [7], un compilateur pour réseaux de Petri de haut niveau qui donne la possibilité d'explorer aisément l'espace d'états.

— La définition d'une approche d'analyse de comportement de véhicules autonomes combinant vérification formelle et simulation (Section 5). Cette méthode permet de tirer parti des avantages respectifs de la vérification de modèle (l'exhaustivité) et de la simulation (le réalisme). Elle permet de détecter, par confirmation des traces d'exécutions par la simulation, que la vérification ne renvoie pas de faux positifs dus à son niveau d'abstraction.

Plan du manuscrit

Chapitre 1 : Ce premier chapitre tend à éclairer le lecteur sur le contexte et les motivations du travail présenté. On y définit l'étude de cas et l'approche du problème, et le système est conceptualisé en conséquence. On se penche ensuite sur l'utilisation des méthodes formelles dans ce contexte.

Chapitre 2 : Le second chapitre donne au lecteur les notions préliminaires nécessaires à la compréhension du manuscrit et un état de l'art des techniques de vérification formelle est proposé. Les formalismes et outils existants utilisés dans le cadre de la thèse sont alors présentés.

Chapitre 3 : Ce chapitre décrit le cadre logiciel VERIFCAR [8], composé d'un modèle formel et d'une méthodologie de vérification permettant l'évaluation de la robustesse des véhicules autonomes à l'aide du *model checker* UPPAAL [9, 10]. Cet outil de vérification de modèles supporte le formalisme des réseaux d'automates temporisés et permet la vérification de propriétés exprimées dans un sous-ensemble de CTL (Computational Tree Logic) [4]. Des exemples d'utilisation de VERIFCAR sont fournis.

Chapitre 4 : Le quatrième chapitre du manuscrit présente le formalisme des MAPTs (*Multi-Agent with timed Periodic Tasks*. On commence par définir formellement la syntaxe et la sémantique des MAPTs et comment ils peuvent être traduits en réseaux de Petri de haut niveau permettant une implémentation dans ZINC [7]. On propose ensuite des abstractions de la sémantique, en particulier portant sur la représentation du temps. Les algorithmes de vérifications permettant une exploration dynamique de l'espace d'états d'un MAPT sont alors présentés.

Enfin, des expériences dans lesquelles des heuristiques sont utilisées sur les algorithmes de vérifications, mettent en évidence les avantages de cette approche en termes d'expressivité, de niveau d'abstraction et de temps de calcul.

Chapitre 5 : Le dernier chapitre décrit une méthode alliant simulation et vérification formelle qui a été conceptualisée et appliquée à l'analyse de comportement de véhicules autonomes en situations non déterministes. Après une description générale de la méthode ainsi qu'une introduction au domaine de la simulation et de l'outil GAMA [11], le simulateur de systèmes multi-agents utilisé ici, un cas d'étude est défini, avec une prise de décision utilisant le modèle de suivi de voiture (*car following model*) IDM [12]. IDM (*Intelligent Driver Model* en anglais) est un algorithme décisionnel pour les véhicules autonomes dans le cadre d'un comportement sans changement de voie. Des indicateurs pertinents sont ensuite choisis et appliquées afin d'appliquer la méthode sur le cas d'étude.

Conclusion et perspectives : La conclusion du manuscrit résume les travaux présentés et les applications concrètes dans lesquelles ils peuvent être utilisés ou sont en cours d'utilisation. Y sont également développées des pistes de recherche qui permettraient de généraliser ou d'améliorer l'efficacité des méthodes présentés.

Chapitre 1

Contexte et motivation

Sommaire

1.1 Approche pour l'évaluation du système 15
1.2 Caractéristiques du système 16
1.3 Comparaison avec des approches existantes 20

Les **véhicules autonomes** sont des entités rationnelles sophistiquées (appelées agents) qui, comme le terme le suggère, agissent de manière autonome dans des environnements ouverts et répartis. Ils peuvent avoir des perceptions différentes de leur environnement, en raison des informations qu'ils possèdent, et des intérêts divergents en termes d'objectif à atteindre. Les communications entre véhicules affectent les perceptions et, à leur tour, les décisions et les comportements individuels. Un système de véhicules autonomes **communicants** ou **CAVs** (pour *Communicating Autonomous Vehicles*) est donc à la fois un système multi-agents [13] et un système temps réel [14]. Plus précisément, un système de CAVs est un réseau de véhicules qui communiquent avec leurs voisins pour atteindre un objectif le plus rapidement possible tout en respectant le code de la route et en évitant les accidents. De plus, un tel système se doit également de respecter un ensemble de contraintes temporelles. L'analyse comportementale des véhicules autonomes est un défi pour les modélisateurs depuis des années [15, 16, 17, 18]. Elle peut être abordée via l'une des approches suivantes :

— Un essai sur route consiste à tester des véhicules autonomes sur des routes ou des circuits existants afin d'étudier leur comportement dans les différentes situations qu'ils peuvent être amenés à gérer. Malgré des avantages évidents liés à son réalisme, cette méthode présente cependant de sérieuses limitations. En effet, des prototypes coûteux sont nécessaires et le temps passé à l'essai est long puisqu'il correspond au temps réel passé sur la route. Enfin, certains scénarios ne peuvent pas être étudiés avec cette approche, typiquement les

plus dangereux, qui peuvent potentiellement conduire à un crash. En conséquence, les approches informatiques sont généralement préférées ;

— Les simulations informatiques permettent de modéliser le comportement des véhicules dans un environnement choisi, de sorte que différents types de scénarios puissent être étudiés [19, 20, 21]. Cependant, par rapport à la méthode précédente, on perdra en réalisme en modélisant les véhicules et leur environnement (on parle alors d'abstraction, car on s'abstrait du réel). De plus, lorsque les véhicules présentent des comportements non déterministes, les outils de simulation ne sont généralement pas exhaustifs, étant donné que chaque simulation correspond à un seul chemin dans le graphe de tous les comportements possibles. Cela est particulièrement vrai dans le contexte des agents communicants, où les agents interagissent pendant des intervalles de temps non déterministes, permettant plusieurs actions possibles à un moment donné. Il est donc intéressant dans ce genre de cas de vérifier formellement les comportements des CAVs afin d'être confiant dans l'intégration des véhicules autonomes dans le trafic routier ;

— Pour améliorer la confiance dans certains modèles, la technique de vérification de modèles ou *model-checking*, qui est une méthode exhaustive, peut être utilisée. Les outils de *model-checking* permettent d'obtenir automatiquement des réponses binaires (oui ou non) aux questions sur le comportement dynamique des véhicules. La modélisation et la vérification formelle des systèmes de CAVs nécessitent non seulement la définition des états du véhicule et de la route (l'environnement), mais également la spécification des interactions entre les véhicules ainsi qu'un langage de requêtes capable d'exprimer les propriétés que l'on souhaite vérifier. Pour ce faire, des modèles spécifiques de véhicules dans un environnement donné doivent être fournis. Cependant, l'utilisation de ces outils est généralement limitée, car elle est assez exposée au phénomène bien connu d'explosion de l'espace d'états.

Les politiques de décision des CAVs peuvent avoir un impact sur la sécurité, la fluidité du trafic et la consommation d'énergie. Elles s'appuient souvent en pratique sur des algorithmes de planification de trajectoire étudiés en simulation 3D, souvent en conjonction avec des expériences sur route [22, 23, 24, 25, 26]. Alors que la fiabilité de tels systèmes est une préoccupation majeure des concepteurs [27], la vérification formelle des politiques décisionnelles des CAVs est un domaine sous-explorée dans lequel il est difficile de trouver des travaux publiés. C'est dans ce contexte que se situe cette thèse, qui est à notre connaissance la première approche de ce type ayant l'ambition de permettre une vérification formelle du comportement des véhicules autonomes communicants.

1.1 Approche pour l'évaluation du système

Les modèles et algorithmes présentés dans ce manuscrit visent à permettre une vérification formelle de la prise de décision dans un système de CAVs. L'angle d'approche consiste à étudier l'impact des aspects temporels non déterministes, tels que la latence dans les communications entre véhicules, sur le comportement des véhicules autonomes communicants.

Afin de conceptualiser le système, il est nécessaire de déterminer quels sont les indicateurs sur lesquels portent les propriétés que l'on souhaite vérifier. Nous nous concentrons en particulier sur deux types de propriétés principales. D'une part celles portant sur la sécurité des véhicules, qui garantissent qu'un véhicule respecte toujours les distances de sécurité et minimise les risques d'accident. D'autres part celles portant sur la fluidité du trafic, qui consistent à optimiser la vitesse de chaque véhicule et à éviter les embouteillages, ce qui permet entre autres de réduire les émissions atmosphériques et la consommation de carburant. La première peut être abordée, par exemple, avec le *Time-To-Collision* (TTC), qui calcule le temps avant qu'un véhicule entre en collision avec un véhicule qui le précède sur la même voie, s'ils gardent tous deux leurs vitesses actuelles. La valeur TTC_i de *Time-To-Collision* du véhicule i à un moment donné dépend des vitesses et positions de deux véhicules :

$$TTC_i = \frac{x_i(t) - x_j(t) - l(i)}{v_i(t) - v_j(t)} \, \forall v_i(t) > v_j(t),$$

où i représente le véhicule, j représente le véhicule qui précède i, et pour un véhicule donné k, x_k correspond à sa position, l_k à sa longueur et v_k à sa vitesse. La seconde peut être adressée, par exemple avec le *Travel Time* [28, 29], ou temps de parcours, qui correspond au temps total qu'un véhicule met dans un scénario donné pour parcourir une distance souhaitée. En complément du temps de parcours, l'ordre de passage des véhicules à un endroit nous donne des informations supplémentaires, notamment sur les possibilités de dépassement. D'autres propriétés portant sur le confort ou l'équité peuvent être vérifiées à l'aide d'indicateurs tels que les variations d'accélération ou de décélération, ou le temps d'attente. Ces indicateurs seront généralement analysés par leurs extremums, de sorte que des questions comme «Le véhicule est-il toujours dans une situation sûre ?» ou «Est-il possible pour le véhicule de dépasser celui qui le précède ?» peuvent se ramener à chercher s'il existe un état du système tel que l'indicateur concerné dépasse une certaine valeur.

Le système doit avoir un niveau de réalisme suffisamment élevé pour pouvoir y appliquer divers algorithmes de prise de décision susceptibles d'être utilisés sur des véhicules réels. Le type de système de CAVs qu'on étudie s'attache ainsi à représenter avec précision la position de véhicules sur la route et la temporalité des communications entre ceux-ci, mais ne s'attarde

pas à représenter les capteurs de perception du véhicule (caméras, LIDARs, ultras sons, ...) lui permettant de connaître son environnement, cet aspect n'étant pas ciblé par notre approche. En revanche on représentera les récepteurs ou les émetteurs sevrant à la communication, de sorte à pouvoir altérer leur fonctionnement et observer comment les véhicules se comportent en cas de mauvais fonctionnement ou de panne des communications.

1.2 Caractéristiques du système

Les systèmes de CAVs que nous considérons sont définis comme suit : Un système décrit une section de route composée de plusieurs voies unidirectionnelles avec plusieurs agents (des véhicules), chacun réalisant un objectif qui lui est propre. Cet environnement (la route et les agents) est représenté par une structure de données contenant l'état de chaque véhicule ainsi qu'un ensemble de contraintes sur leurs actions possibles. L'état d'un véhicule est conservé sous la forme d'un ensemble de valeurs comprenant la position, la vitesse, la direction, la connaissance de l'environnement, etc. La position actuelle est exprimée à l'aide de valeurs discrètes, mais avec suffisamment de précision pour modéliser la progression du véhicule sans entraîner de comportement anormal dû à la perte d'informations. Plus précisément, la position d'un véhicule est considérée comme un point sur une grille orthogonale bi-dimensionnelle. Nous supposons que les véhicules sont équipés de divers capteurs de perception permettant d'observer le comportement de leurs voisins. Notre approche étant axée sur l'impact des communications entre véhicules sur les décisions, nous partons du principe que les capteurs de perception du véhicule sont parfaits, en ce sens que les informations obtenues sont toujours précises et immédiates. Les véhicules sont également capables de communiquer et de recevoir des informations qui ne sont pas directement observables, telles que le changement de voie prévu des autres véhicules. Ces communications sont programmées de manière à représenter de manière réaliste les délais entre les émissions et les réceptions de données. Le comportement de chaque véhicule consiste à répéter indéfiniment, à une fréquence qui lui est propre :

— un calcul permettant de prendre une décision sur l'action immédiate à exécuter (il s'agit de l'algorithme de prise de décision) sous la forme d'une accélération (accélération, freinage, pas de changement) et d'une direction (gauche, droite, pas de changement), suivi par

— la communication de son intention, sous la forme de la trajectoire qu'il souhaite suivre, à tous les véhicules pouvant recevoir cette information. Les informations reçues provenant des autres véhicules sont stockées dans la base de données du véhicule et utilisées lors de l'exécution de l'algorithme de décision.

L'algorithme de prise de décision peut être différent pour chaque véhicule, ces derniers étant

considérés comme des agents indépendants. Les vitesses et les positions des véhicules sont mises à jour de façon simultanée pour tous les véhicules, ce qui signifie que l'observation du système est indépendante de la fréquence de prise de décision des véhicules.

La route. Nous considérons une section de route composée d'une ou de plusieurs voies unidirectionnelles sur lesquelles les véhicules circulent et nous mémorisons à tout moment la position de chaque véhicule sous forme de coordonnées. La position actuelle (x, y) (ainsi que la position initiale) est exprimée par la distance entre le début du tronçon de route (x) et l'une des bordures latérales de la route (y). Pour des raisons de vérification nécessitant des domaines de valeurs discrètes, ces distances sont exprimées à l'aide de valeurs entières, mais suffisamment précises pour modéliser de manière satisfaisante la progression du véhicule sur la route (voir la section 3.1.2 pour plus de détails sur les aspects de discrétisation). En conséquence, les différents objets présents sur la route, dont les véhicules, auront une position représentée par un point sur une grille orthogonale bi-dimensionnelle sur laquelle les espaces longitudinaux et latéraux entre points adjacents peuvent être différents.

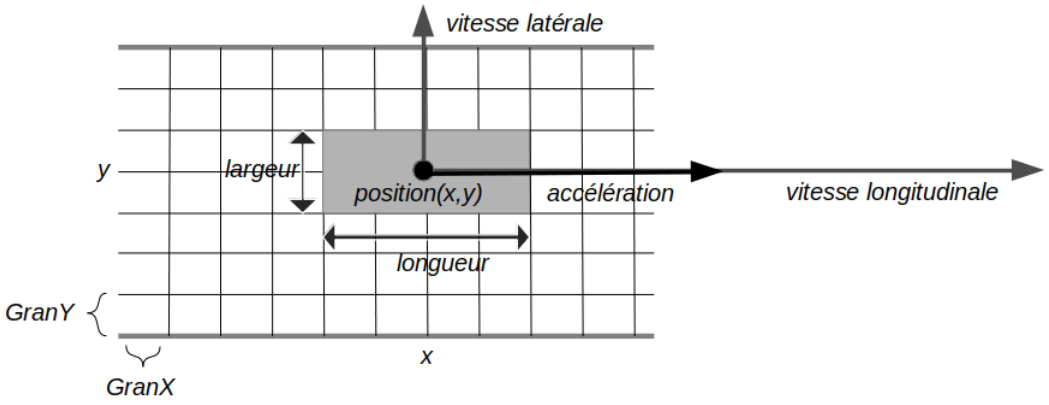

FIGURE 1.1 – Une portion de route discrétisée avec une granularité Gran_x en coordonnée x et Gran_y en coordonnée y, avec un véhicule en mouvement sur la position (x, y).

Les véhicules. Chaque véhicule est représenté sur cette grille orthogonale par un rectangle de longueur et de largeur données, centré sur sa position (x, y), comme illustré en Fig. 1.1. Son état est ainsi décrit par un ensemble de variables contenant sa position (le centre du rectangle), ses vitesses longitudinales et latérales actuelles, son accélération longitudinale (dans un intervalle donné allant de valeurs négatives à positives) et sa connaissance des autres véhicules (par exemple sous la forme de trajectoires temporelles). Notez que la taille des ensembles contenant

ces variables est critique et doit être choisie avec soin, car elle affecte directement la taille de l'espace d'états et la précision de la modélisation. Dans la réalité, un module de contrôle se charge d'appliquer la prise de décision afin que les trajectoires choisies soient respectées. Notre approche ne portant pas sur cette partie du comportement des CAVs, nous faisons abstraction des lois physiques telles que la rotation ou l'inertie, impliquées dans les manœuvres en situation réelle. Par conséquent, le rectangle représentant le véhicule ne tourne jamais et le changement de direction est appliqué directement sur la valeur de la vitesse latérale. En d'autres termes, le fait de tourner le volant a un impact sur la valeur de vitesse qui fera bouger le rectangle sur l'axe y, tandis que sa vitesse longitudinale le déplace toujours sur l'axe x.

La prise de décision. L'algorithme de prise de décision du véhicule suit une politique de décision donnée. Ce dernier peut notamment définir des paramètres tels que des distances de sécurité à respecter en fonction d'une vitesse. Plusieurs véhicules peuvent avoir des politiques de prise de décision différentes. Pour prendre une décision, l'algorithme prend en compte ses propres informations d'état, y compris ce qu'il sait de son voisinage (par exemple, une représentation des trajectoires temporelles d'autres véhicules) ainsi que son propre itinéraire (ses objectifs). L'itinéraire du véhicule est défini comme une séquence de positions que le véhicule doit atteindre. Plus précisément, nous considérons une série d'ensembles de positions à atteindre, avec la nécessité d'atteindre au moins une position de chaque ensemble. Ces ensembles peuvent contenir plusieurs positions latérales adjacentes à la même distance de l'origine. Concrètement, cela permet de fixer l'itinéraire en tant que paramètre du véhicule et qui influera les choix de décision en fonction de la sortie que le véhicule doit prendre.

La mise à jour de l'environnement. A une fréquence constante, la vitesse longitudinale et la position de chaque véhicule sont mises à jour en fonction de l'accélération du véhicule et des valeurs de vitesse longitudinale et latérale (la modification des autres variables dépend d'événements tels que une prise décision ou une réception de message). Nous appelons ce processus la mise à jour de l'environnement, car il met à jour l'état de tous les agents du système simultanément. La fréquence de mise à jour de l'environnement doit être supérieure à la fréquence de l'algorithme de décision de chaque véhicule. Cela permet d'éviter qu'il n'y ait pas de mise à jour entre deux prises de décision du même véhicule, ce qui signifierait que la première prise de décision n'a pas d'impact sur l'état du véhicule. En général, afin de représenter les actions des véhicules de manière réaliste, la fréquence de mise à jour de l'environnement doit être aussi élevée que possible. Cependant, la taille de l'espace d'états augmente de manière exponentielle par rapport à la valeur de cette fréquence. Il convient donc de choisir le bon compromis entre réalisme et réduction de l'espace d'état.

Les trajectoires temporelles. L'algorithme de décision vise à prévoir les futurs conflits entre véhicules et à choisir la manœuvre la mieux adaptée à leurs besoins. Nous proposons pour cela des trajectoires prévisionnelles mais d'autres algorithmes spécifiques pourraient être utilisés.

Les trajectoires prévisionnelles des véhicules sont des abstractions temporisées représentées sous forme de séquences de positions que le véhicule atteindra à certaines dates jusqu'à un horizon temporel (généralement 10 s), comme indiqué dans la Fig 1.2. Elles sont censées représenter l'intention d'un véhicule à un moment donné et peuvent être communiquées à d'autres véhicules. Afin d'optimiser l'espace mémoire, ces trajectoires ne sont jamais stockées mais elles sont recalculées à partir des informations d'un véhicule donné sur son voisinage. Ces informations doivent être encodées de manière aussi compacte que possible. Pour ce faire, nous encodons ces trajectoires avec deux variables possédant un intervalle limité de valeurs possibles :

— La voie que le véhicule souhaite atteindre actuellement.

— Le délai prévu par l'algorithme de décision avant le début d'un changement de voie (le cas échéant).

Ce dernier est exprimé en utilisant un intervalle et une durée maximale. Par exemple, un pas de 100 ms et une durée maximale de 5 secondes donneraient 51 valeurs possibles ([0.0,0.1, ..., 4.9,5.0]) pour la variable de délai. Pour indiquer le prochain changement de voie, cela donne 103 possibilités : 51 possibilités pour un changement à gauche, 51 pour un changement à droite et 1 lorsqu'il n'y a pas de changement de voie.

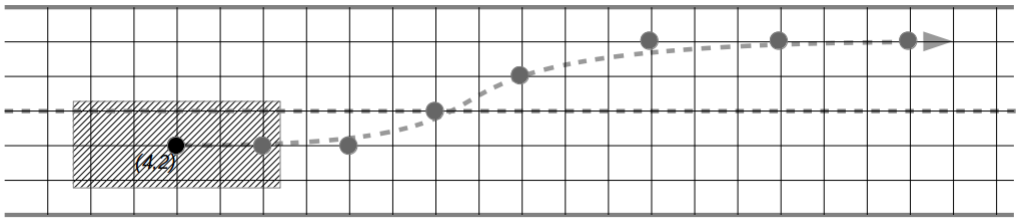

FIGURE 1.2 – Un tronçon routier à deux voies où un véhicule (le rectangle hachuré) est centré en $(4, 2)$ et sa trajectoire prévue $(6, 2); (8, 2); (10, 3); (12, 4); (15, 5); (18, 5); (21, 5)$ pour les sept pas de temps suivants est indiqué par des points gris (la progression dans x varie en fonction de la vitesse).

Dans un système réel, le pas de temps séparant deux points dans de telles trajectoires prévisionnelles serait plus petit, afin d'avoir davantage de précision. Cependant, comme dans notre modélisation la précision temporelle de l'environnement dépend de la fréquence de mise à jour de l'environnement, il est inutile de représenter des états qui ne peuvent être observés, ni par les

véhicules ni par l'environnement. En conséquence, le pas de temps séparant deux points dans les trajectoires temporelles sera identique au temps entre deux mises à jour de l'environnement.

Les autres formes de communications Étant donné que nous visons à modéliser des véhicules coopératifs, en plus de l'échange d'informations entre véhicules, nous proposons d'étudier deux autres formes de communication susceptibles d'être utilisées dans le futur pour les CAVs : la négociation et la prise de décision basée sur les infrastructures.

La négociation se comporte comme un algorithme réparti dans lequel les agents peuvent interagir les uns avec les autres, chaque agent essayant d'influencer les autres pour tendre vers un comportement global qui lui permette d'atteindre ses objectifs de navigation le plus rapidement possible. Typiquement, un véhicule peut «demander» à autre d'attendre quelque temps avant d'effectuer une action, afin de minimiser l'impact négatif que cette action pourrait avoir sur la sécurité et / ou la fluidité du trafic. Les algorithmes de décision prennent alors en compte de telles demandes lors de la prise de décision.

La décision basée sur les infrastructures consiste à utiliser des bornes le long de la route, qui observent les mouvements et les intentions des véhicules, calculent et leur envoient des ordres, en contrôlant le trafic de manière centralisée. Ici, ces bornes sont modélisés de la même manière que les véhicules, c'est-à-dire comme des agents qui reçoivent les données diffusées par les véhicules et calculent les ordres à envoyer selon une fréquence donnée.

Notre objectif est la modélisation de systèmes de CAVs, afin de montrer comment évaluer la robustesse d'une politique de décision donnée. Pour illustrer notre approche, nous avons conçu un algorithme de prise de décision pour les systèmes de CAVs avec plusieurs variantes, avec et sans l'implémentation des formes de communications susmentionnées. Cependant, nous n'avons pas l'ambition de présenter et de promouvoir un algorithme de prise de décision efficace.

1.3 Comparaison avec des approches existantes

La vérification formelle est peu exploitée dans le contexte des systèmes de CAVs, sans doute du fait de la taille inhérentes à ce genre de systèmes. Parmi les approches traitant de la vérification formelle dans le contexte des véhicules autonomes, certaines utilisent des modèles temporisés, ce qui semble approprié pour l'étude du non-déterminisme induit par les délais et la latence dans les communications entre véhicules. C'est le cas, par exemple, dans [30], qui vise à vérifier la couche fonctionnelle des robots mobiles, c'est-à-dire la couche bas niveau qui interagit directement avec les capteurs et les actionneurs et transmet des informations à la couche de décision. Pour cette raison, il n'a pas été nécessaire de modéliser plusieurs agents évoluant

dans un environnement donné. [31] est un autre exemple, qui traite de la qualité des *platoons* de véhicules (permettant aux véhicules de circuler en groupe sur les routes). Il se concentre sur les propriétés du groupe de véhicules, par exemple, l'insertion et le départ, et considère la représentation des véhicules par rapport au peloton. Ces deux approches se concentrent sur des propriétés spécifiques et leurs modèles ne sont pas assez complets pour représenter l'environnement physique des véhicules.

Dans [32], les systèmes hybrides sont utilisés pour modéliser des véhicules autonomes. Avec ce formalisme, combinant à la fois des variables d'état continues et des modes de fonctionnement discrets, leur modèle permet d'obtenir une représentation réaliste de la physique des véhicules, similaire à celle que l'on peut trouver dans les simulations (angle de glissement, taux de lacet, etc.). Cependant, un tel réalisme conduit à de faibles performances pendant la phase de vérification, comme le montre l'étude de cas présentée. En effet, pour un système avec seulement deux véhicules et un seul appliquant une politique de décision générant un non déterminisme limité (n chemins pour n pas de temps) et l'absence de communication entre véhicules, le vérificateur de modèle dReach [33] prend déjà plusieurs minutes pour une exploration complète de l'espace d'état.

Un autre travail, concernant les contrôleurs de train modélisés avec des systèmes hybrides [34], propose une représentation similaire des agents, prenant en compte leur vitesse et leur accélération afin de modéliser un mouvement physique réaliste. Ici, les communications entre trains sont possibles, mais le phénomène d'explosion d'état est encore plus présent, puisqu'un simple cas avec deux trains met plus d'une demi-heure à donner des résultats.

La représentation la plus proche de celle utilisée dans ce manuscrit semble être [35], impliquant des robots évoluant dans une grille bi-dimensionnelle et modélisés avec un formalisme basé sur des automates temporisés. Les robots mobiles sont représentés comme des agents évoluant dans un environnement physique mais à un niveau d'abstraction très élevé et ne pouvant effectuer que des actions élémentaires (telles que le déplacement dans une cellule adjacente). En particulier, le modèle n'inclut pas les valeurs de vitesse ou d'accélération, qui sont cruciales pour le réalisme des véhicules sur une route. De plus, les agents ne peuvent pas communiquer entre eux, ce qui rend impossible l'étude du non-déterminisme induit par les communications sur un système de CAVs.

Chapitre 2

Notions préliminaires et état de l'art

Sommaire

2.1 Notions préliminaires sur les graphes 23
2.2 Les techniques de vérification formelle 25
2.3 Les formalismes de modélisation 29
2.3.1 Réseaux d'automates temporisés 29
2.3.2 Réseaux de Petri de haut niveau 32
2.3.3 Automates hybrides 33

Les méthodes de vérification formelle impliquent de représenter l'ensemble des exécutions possibles d'un système. Cette représentation peut être vue comme un graphe qu'il s'agira de parcourir afin de vérifier des propriétés de logique temporelle. Ce procédé nécessite au préalable de décrire le système à l'aide d'un formalisme de modélisation. Dans ce chapitre, nous verrons en premier lieu quelques notions de base de parcours de graphes. Ensuite, nous passerons en revue les principales techniques de vérification formelle. Finalement, les formalismes et les outils utilisés dans le cadre de la thèse seront décrits.

2.1 Notions préliminaires sur les graphes

En informatique, et dans les mathématiques discrètes en général, un **graphe** est une structure mathématique utilisée pour représenter des problèmes de manière simple. Un graphe est composé de **sommets** et d'**arêtes**. Chaque sommet correspond à un état du problème représenté, et chaque arête met deux sommets distincts en relation.

Un graphe **orienté** est un graphe pour lequel chaque arête possède un sommet de départ et un sommet d'arrivée, de sorte que la relation entre les deux sommets ne va que dans un seul

sens. Pour ce type de graphe, les arêtes peuvent également être appelées **arcs**, ce qui indique une orientation. Un exemple de graphe orienté est donné en Figure 2.1.

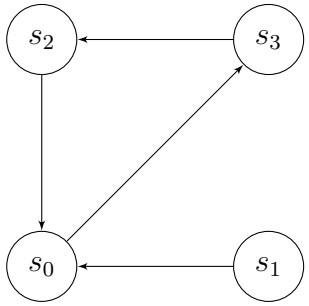

FIGURE 2.1

Un **chemin** allant du sommet s au sommet s' dans un graphe orienté est une suite finie d'arcs consécutifs distincts qui part de s et arrive en s'. Par exemple, en Figure 2.1, (s_1, s_0),(s_0, s_3),(s_3, s_2) est un chemin allant de s_0 à $s2$.

Un **cycle** (ou circuit) dans un graphe orienté est un chemin non vide où le sommet de départ est le même que celui d'arrivée (c'est à dire que $s = s'$). Par exemple, en Figure 2.1, (s_2, s_0),(s_0, s_3),(s_3, s_2) est un cycle.

Un graphe orienté **acyclique** ou DAG (pour *Directed Acyclic Graph*) est un graphe orienté qui ne possède pas de cycle. Cela signifie que les sommets d'un DAG peuvent être ordonnés de manière à ce que tous les arcs partent d'un sommet (ou groupe de sommets) supérieur vers un sommet (ou groupe de sommets) inférieur.

En *model-checking*, le comportement d'un système est représenté sous la forme d'une **structure de Kripke**. Une structure de Kripke est un graphe orienté où les sommets représentent les états accessibles du système et où les arcs représentent les transitions d'états de sorte qu'il est possible de passer d'un état s à un état s' s'il existe un arc de s vers s'. Chaque sommet est étiqueté par un ensemble de valeurs de sorte qu'une propriété portant sur ces valeurs peut être vérifiée comme étant vraie ou fausse sur chacun des états. De plus, un des sommets est défini comme **état initial** du système.

Une **exploration** (ou parcours) de graphe est un algorithme consistant, à partir de l'état initial, à atteindre les états accessibles d'un graphe, en général afin d'y vérifier une propriété donnée. Une structure de Kripke peut être explorée par un algorithme visant à vérifier une propriété de logique temporelle. La **logique temporelle** est une logique mathématique permettant de décrire des propriétés sur l'évolution d'un système dans le temps. Par exemple, «p est toujours vrai» signifie qu'une exploration de la structure de Kripke doit pouvoir vérifier p sur chaque état accessible. Autre exemple, «p finit toujours par être vrai à un moment» signifie qu'une exploration de la structure de Kripke doit pouvoir vérifier p sur au moins un état de

chaque chemin qu'il est possible de suivre depuis l'état initial. C'est pour pouvoir vérifier de telles propriétés qu'ont été proposées des techniques de vérification formelle.

La logique temporelle qui sera utilisée dans le cadre de la thèse est la logique du temps arborescent (CTL) [36]. Les requêtes CTL sont formées de paires de quantificateurs de chemin A ou E (pour Always et Exists, respectivement) et d'opérateurs de chemin G ou F (pour Globally et Finally, respectivement). Par exemple, la formule $EF\ p$ (respectivement $AF\ p$) signifie qu'il existe au moins un état satisfaisant la propriété p sur au moins un chemin (respectivement sur tous les chemins) à partir de l'état initial, où p est une formule propositionnelle (c'est à dire une expression évaluée comme vraie ou comme fausse).

Une vérification de modèle peut être vu comme un calcul qui utilise un modèle (par exemple un ensemble de comportements possibles de véhicules sur une portion de route) et une propriété (par exemple la possibilité d'une collision entre les véhicules) et retourne un résultat binaire (vrai ou faux). Cela nécessite généralement de modéliser le système comme une structure de Kripke, et de formaliser, à l'aide des logiques temporelles, une propriété comportementale. Le résultat de la vérification est ensuite obtenu par l'exploration automatisée de tous les états du modèle, ce qui signifie que tous les futurs possibles d'une situation initiale donnée sont pris en compte dans l'évaluation de la propriété.

2.2 Les techniques de vérification formelle

Dans le domaine des méthodes formelles, on désigne par l'expression **espace d'états** la structure de Kripke représentant l'ensemble des exécutions possibles d'un système, ainsi que les états rencontrés au cours de ces exécutions. Le principal problème associé au *model-checking* est le problème dit d'explosion du nombre d'états. Cela signifie que le nombre d'états dans lesquels peut se trouver un système augmente fortement en fonction du nombre d'éléments qui composent celui-ci (comme par exemple le nombre de variables présentes dans le système). Cela peut alors conduire à des temps de calculs trop élevés pour que la vérification soit utilisable en pratique. Les différentes techniques de *model-checking* se distinguent principalement par leur façon de représenter les ensembles d'exécutions possibles du système. On distingue ainsi les méthodes **explicites** des méthodes **symboliques**. Les méthodes explicites, comme [37], parcourent la structure de Kripke décrivant l'ensemble des exécutions du modèle en calculant pour chaque état quels sont ses successeurs. Les propriétés sont alors généralement vérifiées par des algorithmes de parcours de graphe et vont être exprimées sous forme de logique temporelle. Les méthodes symboliques, comme [38], considèrent de façon ensembliste les configurations que peut prendre le système. Cela consiste à rassembler ensemble des états semblables.

Les méthodes explicites Les réductions utilisées dans le cadre des méthodes explicites qui visent à réduire le nombre d'états ou de chemins à explorer dans l'espace d'états.

Parmi les techniques de réductions les plus utilisées, on trouve les réductions d'ordre partiel. Les réductions d'ordre partiel [39, 40, 41] sont utilisées pour exploiter les symétries présentes dans l'espace d'états des systèmes concurrents. Dans de tels systèmes, l' explosion d'états résulte de la concurrence des actions des différents agents. Cette dernière conduit en effet à plusieurs ordres possibles sur les actions. Si la prise en compte de tels comportements est en général nécessaire, il peut arriver que l'ordre dans lequel a lieu un ensemble d'actions n'impacte pas la propriété que l'on souhaite vérifier. Typiquement, les réductions d'ordre partiel exploitent ce fait et définissent, pour certains états, un ordre de priorité pour les agents concurrents afin d'éviter le non-déterminisme local. Cela permet d'explorer un espace d'états contenant moins d'états et de transitions. Cette méthode est cependant difficilement automatisable, car elle demande une connaissance au préalable du système, et des erreurs peuvent conduire à supprimer des chemins qui impactent la validité de la propriété vérifiée.

Une autre technique très présente dans la littérature est la réduction par symétrie [42, 43] qui consiste à éviter des zones de l'espace d'états qui sont symétriquement équivalentes à celles précédemment explorées. Cette approche dépend là encore fortement du type de système étudié, étant donné que les symétries structurelles du système d'origine seront reflétées dans l'espace d'états. Elle convient donc bien aux systèmes concurrents, qui contiennent souvent de nombreux composants similaires les uns aux autres et où la vérification du modèle peut impliquer un parcours redondant sur des zones équivalentes de l'espace d'états. Les techniques de réduction par symétrie exploitent ces similitudes en restreignant l'espace d'états à l'exploration d'un seul représentant de chaque classe d'équivalence. Pour que cette technique réduise le temps de calcul d'une vérification, il est cependant nécessaire d'identifier les symétries sans construire l'espace d'état. Ainsi, lors de la vérification, lorsqu'un état s est atteint, il faut vérifier si un élément t a déjà été atteint tel que s et t appartiennent à la même classe d'équivalence. Ce problème est connu sous le nom de problème d'orbite. Il s'agit du point crucial de la réduction par symétrie, étant donné qu'elle ne pourra être efficace que si ce problème peut être résolu efficacement pour le système étudié.

Enfin, l'approche *on-the-fly* [44, 45] (à la volée), permet de ne pas avoir à construire l'entièreté de l'espace d'états pour déterminer si un système satisfait ou non une propriété donnée. S'il existe un état ou un cycle d'états permettant de conclure sur la propriété, comme par exemple une propriété de sûreté qui serait invalidée par un état ne la respectant pas, seulement la partie de l'espace d'états conduisant à cet état sera construite. Cette approche convient particulièrement à des explorations en profondeur d'abord visant à trouver les chemins pouvant mener à une situation que l'on sait problématique. En revanche, si la propriété est valide, l'ensemble

de l'espace d'états devra être construit et le parcours à la volée ne sera pas avantageux. Il est cependant possible de combiner l'approche à la volée avec les réductions d'ordre partiel [46], ou l'utilisation des réductions par symétrie [47].

Les méthodes symboliques Il existe deux approches principales pour le *model-checking* symbolique, l'utilisation de diagrammes de décision [48, 49] d'une part, et de SAT-solvers [50, 51] de l'autre.

L'approche basée sur les diagrammes de décision consiste à réduire l'espace d'états de sa forme d'arbre de décision en le transformant en graphe acyclique. Un encodage fréquemment utilisé pour cela est le diagramme de décision binaire (BDD), qui permet de supprimer les isomorphismes et le non-déterminisme local n'ayant pas d'impact sur le système. Dans un encodage BDD, un état du système est considéré comme un chemin linéaire traversant toutes les variables représentant le système. Cela peut représenter un handicap lorsque l'on souhaite vérifier des spécifications complexes, car les informations de structure sont perdues lors de l'encodage. De plus, la taille de la formule booléenne est exponentielle en fonction du nombre de variables, pouvant conduire à une consommation trop importante d'espace mémoire.

L'approche fondée sur les SAT-solvers consiste à encoder le problème de vérification en une instance de SAT. La formule booléenne qui en résulte décrit une exécution de longueur k dans la structure de Kripke (on dit qu'on déroule la relation de transition) et lui concatène la propriété à vérifier. Un SAT-solver est alors utilisé pour trouver un ensemble de variables pour lesquelles la formule est fausse. De ce fait, cette méthode ne peut vérifier une propriété que jusqu'à une certaine profondeur dans l'espace d'état. Pour cette raison, on fait référence à cette approche sous le nom de *Bounded Model Checking* (BMC) [52, 53, 54]. L'avantage par rapport au BDD est que le SAT-solver ne souffre pas de l'explosion de l'espace d'état. Cependant, la taille de la formule booléenne étant exponentielle en fonction du nombre de variables, cette méthode n'est efficace en temps de calcul que pour un k restreint, d'autres problèmes lié au temps de calcul survenant quand le chemin décrit est grand. De plus, le BMC ne permet en général pas de prouver des propriétés de sûreté, c'est à dire de conclure sur la non-atteignabilité d'un état.

De nombreuses variantes du BMC ont été proposées. On compte notamment le *model-checking* par interpolation [55], ayant pour objectif de permettre de prouver une propriété de sûreté. Soit A et B deux formules booléennes, P est un interpolant de (A, B) si $A \rightarrow P$ et $P \wedge B$ est faux, avec P ne contenant que des variables communes à A et B. Si le BMC ne trouve pas de trace atteignant l'état recherché, le problème est décomposé en la recherche d'un état ayant à la fois, un préfixe commençant à l'état initial, et un suffixe se terminant par l'état recherché par le BMC. Il s'agit alors de trouver un interpolant P_i du couple (préfixe,suffixe) tel qu'il existe $P_i + 1 = P_i$. Cela signifie qu'un point fixe est trouvé sans qu'il y ait d'in-

clusion entre la sur-approximation du préfixe et le suffixe (calculé par *backtracking*). Si un tel interpolant existe, on peut en déduire que tout les chemins de l'état initial conduit à cet interpolant, lequel ne conduit jamais à l'état recherché. La propriété est donc invalide. L'utilisation d'une sur-approximation peut cependant conduire à des faux négatifs dans le sens où on déduit qu'une propriété est invalide alors que l'interpolant n'existerait pas sans la sur-approximation. Si cette méthode permet parfois de prouver des propriétés de sûreté avec le BMC, elle peut se révéler coûteuse et implique de trouver des interpolants, rendant son efficacité particulièrement dépendante du type de système modélisé.

Une autre variante du BMC est basée sur les inductions [56]. Elle permet de générer la formule en entrée du SAT-solver sans dérouler la relation de transition, mais en générant étape par étape des clauses inductives (K-induction) sur les états successeurs. La K-induction d'une propriété P implique que soient vérifiés un cas de base et un cas inductif. Le cas de base implique que tous les états accessibles sur les chemins de taille k vérifient P. Le cas inductif implique que si P est vraie sur un chemin de taille k, P est vraie sur les états accessibles à $k + 1$. Cela permet à la fois, grâce aux inductions, de simplifier le problème, mais peut également améliorer le temps nécessaire à la création de la formule booléenne.

Application aux véhicules autonomes communicants Si le *model-checking* symbolique peut permettre de représenter des espaces d'état significativement plus grands, le *model-checking* explicite peut se montrer plus efficace dans certaines conditions. En effet, de nombreux algorithmes de parcours de graphes, ne s'appliquant pas aux méthodes symboliques, peuvent être utilisés pour réduire le temps de calcul. Cela est dû au fait que les approches symboliques utilisent une recherche en largeur d'abord, là où les méthodes explicites peuvent utiliser des algorithmes de recherche en profondeur d'abord. D'autre part, la méthode symbolique basée sur les diagrammes de décision perd son intérêt lorsqu'un système présente peu d'isomorphismes ou de non-déterminisme local. C'est le cas pour des systèmes complexes composés de plusieurs agents associés à des variables pouvant prendre de nombreuses valeurs, par exemple, un modèle représentant des objets en mouvement avec une précision importante sur leur position et vélocité. Le nombre d'états différents est tel que les isomorphismes sont extrêmement peu probables, une différence de comportement à un moment donné conduisant à des différences, même légères, sur le reste des exécutions. Les systèmes de CAVs rentrent dans cette catégorie. En outre, le non-déterminisme local y est limité. Pour ces raisons, l'utilisation de méthodes symboliques serait peu efficace. Ainsi les travaux présentés dans ce manuscrit utilisent des outils et techniques liées au *model-checking* explicite.

On notera également que parmi les méthodes formelles, les algèbres de processus [57, 58] sont également utilisés dans le cadre de la modélisation formelle de systèmes concurrents. Elles

permettent de décrire les interactions entre un ensemble d'agents indépendants. Elles fournissent également des lois algébriques permettant de manipuler et d'analyser les descriptions de processus, ainsi que de raisonner formellement sur les équivalences entre processus, par exemple, en utilisant la bisimulation (deux systèmes sont dit bisimilaires s'il est possible de représenter le comportement de l'un avec celui de l'autre et réciproquement). Les algèbres de processus se caractérisent principalement par la représentation qui est faite de l'évolution du système. En effet, plutôt que de représenter les conséquences des interactions entre processus sur l'état du système (typiquement en modifiant des variables, partagées ou locales), les algèbres de processus vont se focaliser sur les séquences de communications qui en résultent. Cela permet de vérifier de manière formelle que l'ensemble des exécutions possibles respectent certaines règles, notamment dans le cadre d'applications implémentant des algorithmes répartis. Il est possible de modéliser des systèmes temporisés dans le cadre des algèbres de processus [59]. Cependant, si les aspects de communications sont au coeur de notre approche pour les CAVs, c'est l'impact de ces communications sur l'état des véhicules qui nous intéresse, davantage que les séquences de communication elles-mêmes. En conséquence, la vérification de modèle a été préférée à l'utilisation d'algèbres de processus.

2.3 Les formalismes de modélisation

Un modèle formel définit de manière non ambiguë le comportement d'un système. Cette définition est posée par un formalisme mathématique. Nous nous intéresserons ici aux deux formalismes sont utilisés dans les travaux présentés dans ce manuscrit, à savoir les réseaux d'automates temporisés et les réseaux de Petri de haut niveau, utilisés dans le cadre du *model-checking* explicite, et en particulier ceux permettant une modélisation du temps. On mentionnera également le formalisme des automates hybrides, qui présente des caractéristiques intéressantes vis-à-vis de notre approche.

2.3.1 Réseaux d'automates temporisés

Les automates temporisés sont souvent utilisés pour modéliser (à un certain niveau d'abstraction) des systèmes temps réels et tester leurs propriétés temporelles. Le formalisme des réseaux d'automates temporisé [14], semble être le mieux établi pour la spécification et la vérification de systèmes temps réel répartis. Entre autres avantages, il permet une représentation continue du temps et est compatible avec l'application d'algorithmes de vérification de propriétés temporisées exprimées dans la logique temporelle TCTL [60].

Au niveau syntaxique, un automate temporisé est un graphe orienté et connexe, possédant

un sommet initial, et muni d'**horloges**, c'est-à-dire des variables à valeurs réelles, positives ou nulles. Les sommets sont appelés des **localités** et sont associés à des **invariants**. Les invariants sont des contraintes sur les horloges, autorisant l'entrée sur une localité ou forçant sa sortie, ils sont soit vides (interprétés comme vrais), soit notés comme un prédicat de la forme $x < e$ où x est une horloge et e est une constante entière strictement positive. Les arcs sont annotés avec des **gardes**, auxquelles peuvent s'ajouter des réinitialisations d'horloges et des actions de *broadcast*. Les gardes sont des prédicats de la forme $x \geq e$, autorisant à emprunter les arcs auxquels ils sont associés. Un *broadcast* est un canal de communication permettant aux différents composants du réseau de se synchroniser. Pour un canal k, une action est soit une émission de la forme $k!$, soit une réception de la forme $k?$. Lorsqu'une transition associée à une émission $k!$ est franchie, toutes les transitions franchissables dans le réseau associées à une réception $k?$ doivent être franchies simultanément. Cela signifie qu'aucune autre action ne peut se produire et que la valeur des horloges ne peut pas changer tant que toutes les transitions concernées n'ont pas été franchies.

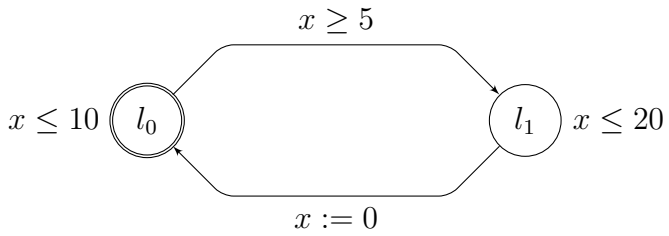

FIGURE 2.2 – Représentation d'un automate temporisé associé à une horloge x initialisée à 0.

Formellement, un automate temporisé est un tuple

$$A = (L, l^0, X, E, I, G, K)$$

où

— L est l'ensemble des localités,

— l^0 est le sommet initial,

— X est l'ensemble des horloges,

— E est l'ensemble des transitions,

— I est l'ensemble des invariants,

— G est l'ensemble des gardes et

— K est l'ensemble des actions de *broadcast*.

Un automate temporisé est représenté en Fig. 2.2. Depuis la localité initiale l_0, il est possible d'attendre jusqu'à 10 unités de temps. A partir de 5 unités de temps, la transition vers l_1 est franchissable. Depuis l_1, il est possible d'attendre jusqu'à 20 unités de temps. La transition vers l_0 est franchissable à tout moment. Son franchissement réinitialise x, ramenant l'automate sur son état initial $(l_0, 0)$.

Plusieurs automates temporisées peuvent être utilisés pour décrire un réseau d'automates temporisés. Le formalisme des réseaux d'automates que nous allons utiliser est celui décrit dans [61] et utilisé par l'outil de vérification de modèle UPPAAL. Un réseau d'automates temporisés est un ensemble $\mathcal{A} = \{A_1, \ldots, A_n\}$ de n automates temporisés A_i pour i entre 1 et n. Le temps augmente de manière uniforme sur toutes les horloges. L'ensemble des localités actives des automates de $\mathcal{A}$ est un vecteur noté $\bar{l} = (l_1, \ldots, l_n)$, initialement sur les localités initiales des automates de $\mathcal{A}$. L'état d'un réseau d'automates est un couple $(\bar{l}, r)$ où $r = X_1 \cup \cdots \cup X_n \rightarrow R^+$ où R^+ est l'ensemble des réels positifs ou nuls (r donne la valeur de toutes les horloges de tous les automates). Un réseau d'automates peut progresser de trois façons :

— Laisser passer un certain laps de temps qu'on notera δ. L'état d'arrivée étant alors $(\bar{l}, r + \delta)$, indiquant que la valeur de chaque horloge a été augmentée d'un laps de temps δ. La nouvelle valeur d'horloge doit respecter les invariants des localités de $\bar{l}$.

— Effectuer une action interne à un automate $A_i \in \mathcal{A}$. L'état d'arrivée étant alors $(\bar{l'}, r)$, où $\bar{l'} = (l_1, \ldots, l'_i, \ldots, l_n)$, indiquant que la localité active de A_i est maintenant l'_i. Pour qu'une action interne à un automate soit possible, ses horloges doivent respecter les contraintes temporelles associées à la transition empruntée.

— Effectuer une synchronisation entre m automates (au moins deux) $A_{i^1}, \ldots, A_{i^m} \in \mathcal{A}$. L'état d'arrivée étant alors $(\bar{l'}, r)$, où $\bar{l'} = (l_1, \ldots, l'_{i^1}, \ldots, l'_{i^m}, \ldots, l_n)$, indiquant que la localité active de A_{i^j} est maintenant l'_{i^j} pour tout j de 1 à m. Pour qu'une synchronisation soit possible, il doit exister un canal de *broadcast* $k \in K$ tel qu'il existe une localité dans $l_{i^1}, \ldots, l_{i^m}$ avec un arc sortant possédant une émission de la forme $k!$, et toutes les autres localités ont un arc sortant possédant une réception de la forme $k?$.

Quelle que soit la progression, les valeurs d'horloges doivent respecter les invariants des localités atteintes. On notera que si des réinitialisations sont présentes sur la transition utilisée, les horloges correspondantes prennent la valeur 0 et r change en conséquence. Si le réseau ne peut progresser d'aucune manière (impliquant que le temps est bloqué par un invariant et qu'aucune transition n'est franchissable), on se trouve dans une situation dite de blocage.

UPPAAL Développé conjointement par les Universités d'Uppsala et d'Aalborg, UPPAAL [9, 10] est un outil de vérification pour les systèmes temps réel ayant prouvé son efficacité dans

des études de cas allant des protocoles de communication aux applications multimedia. Cet outil est conçu pour des systèmes préalablement modélisés sous forme de réseaux d'automates temporisés associés à des variables entières, des structures de données, des fonctions ou encore des canaux de synchronisation. Il permet de générer automatiquement une trace de diagnostic qui explique pourquoi une propriété est (ou n'est pas) satisfaite sur un modèle. Un simulateur intégré à l'outil permet la visualisation des traces. L'outil utilise diverses méthodes d'abstraction et de réduction de l'espace d'états et permet de vérifier un sous ensemble de CTL.

2.3.2 Réseaux de Petri de haut niveau

Les réseaux de Petri [62] sont un formalisme très utilisé pour la spécification et l'analyse de systèmes concurrents. Ils ont bénéficié de nombreuses extensions permettant de répondre à des besoins d'applications spécifiques. En particulier, les réseaux de Petri de haut niveau (parfois appelés réseaux de Petri colorés) peuvent être utilisés pour représenter facilement des situations qui seraient trop complexes pour une représentation avec les réseaux de Petri standard. On notera cependant que l'expressivité de ces deux formalismes reste équivalente. En effet, un réseau de Petri de haut niveau [63] peut être considéré comme une abréviation (une représentation condensée) d'un réseau complexe de bas niveau où les jetons sont des éléments d'un ensemble de valeurs pouvant être vérifiées et mises à jour lors des franchissements de transition. Ils permettent entre autre de représenter des systèmes temps réel [64] et des structures de données complexes.

Formellement, un réseau de Petri de haut niveau est un tuple (S, T, λ, M_0) où :

— S est un ensemble fini de places ;

— T est un ensemble fini de transitions ;

— λ est une fonction d'étiquetage sur les places, les transitions et les arcs telle que

  — pour chaque place $s \in S$, $\lambda(s)$ est un ensemble de valeurs définissant le type de s,

  — pour chaque transition $t \in T$, $\lambda(t)$ est une expression booléenne avec des variables et des constantes définissant la garde de t et

  — pour chaque arc $(x, y) \in (S \times T) \cup (T \times S)$, $\lambda(x, y)$ est l'annotation de l'arc de x à y, conduisant à la production ou la consommation de jetons.

— M_0 est un marquage initial associant des jetons à des places, en fonction de leur type.

La sémantique d'un réseau de Petri de haut niveau peut se définir de différentes façons (entrelacements, pas concurrents, ordres partiels). Dans cette thèse, la sémantique utilisée est la sémantique d'entrelacements, qui est capturée par un système de transition contenant en tant qu'états tous les marquages accessibles depuis le marquage initial M_0. Un marquage M' est

directement accessible à partir d'un marquage M s'il existe une transition t franchissable depuis M et qui conduit à M' ; il est accessible à partir de M si une séquence de ces franchissements y conduit. Une transition t est franchissable à un certain marquage M si les jetons dans toutes les places d'entrée de t permettent de satisfaire le flux exprimé par les annotations des arcs d'entrée et la garde de t, via une évaluation des variables impliquées dans ces derniers. Le franchissement de t consomme les jetons concernés dans les entrées de t et produit des jetons aux places de sortie de t, en fonction des annotations des arcs de sortie et de la même évaluation.

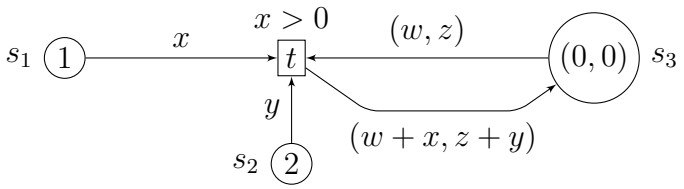

FIGURE 2.3 – Un réseau de Petri de haut niveau.

La figure 2.3 montre un exemple de réseau de Petri de haut niveau où les types des places sont $\mathbb{N}$ pour s_1 et s_2, et $\mathbb{N} \times \mathbb{N}$ pour s_3. La transition t est franchissable dans le marquage initial car il existe une valuation des variables dans les annotations des arcs et dans la garde de t, avec les valeurs des jetons, $x \mapsto 1$, $y \mapsto 2$, $w \mapsto 0$, $z \mapsto 0$, qui satisfait la garde. Le franchissement de t consomme les jetons dans les trois places et produit un nouveau jeton $(1, 2)$ dans la place s_3.

Zinc Le compilateur pour réseaux de Petri de haut niveau (ou réseaux de Petri colorés) ZINC [7], permet de créer des réseaux de Petri, manipuler leurs marquages et déclencher des séquences de transitions. Plusieurs extensions des réseaux de Petri sont implémentées. Les jetons colorés sont des classes d'objets possédants leur propres méthodes et variables, ce qui permet de représenter des systèmes complexes à l'aide de réseaux simples. ZINC ne permet pas une analyse de l'espace d'états et n'est donc pas un outil de *model checking*, mais il peut être utilisé pour explorer des traces ou des espaces d'états à partir d'une bibliothèque de fonctions.

2.3.3 Automates hybrides

Un autre type de modèles utilisé dans le cadre du *model-checking* explicite de systèmes temps réel sont les automates hybrides. Un automate hybride est une machine à état finis avec un ensemble fini de variables continues, chacune décrite par une équation différentielle donnée. Cette spécification combinée de comportements discrets et continus convient bien à la modélisation de systèmes dynamiques intégrant à la fois des composants numériques et non numériques. Un état d'un système avec m variables booléennes et n variables réelles est donc

un point dans $\mathbb{B}^m \times \mathbb{R}^n$. Le système peux évoluer de deux manière distinctes : le changement des seules variables réelles en fonction de l'équation décrivant leur évolution, ou un saut où l'ensemble des variables (réelles et booléennes) changent de valeurs. Il est intéressant de noter que la classe de modèles exprimables avec les automates hybrides inclut celle des automates temporisés. En effet, un automate temporisé est un automate hybride ou seules les horloges sont exprimées avec les variables réelles. Les automates hybrides sont donc plus expressifs, et sont utilisés pour décrire des modèles physiques nécessitant un haut niveau de réalisme, comme par exemple des modèles de variations de température. Néanmoins, cette expressivité a conduit à une augmentation du phénomène d'explosion de l'espace d'états. En effet, les techniques d'abstractions dédiées aux automates temporisés ne peuvent plus s'appliquer dans le cas général des automates hybrides. Le niveau de réalisme recherché par notre analyse ne nécessitant pas de représenter l'évolution des véhicules de manière continue, ce formalisme n'a pas été utilisé dans le cadre de cette thèse.

Chapitre 3

Le cadre logiciel VerifCar

Sommaire

3.1 Le modèle 36
3.1.1 Les composants 36
3.1.2 Choix de modélisation, réglages et précision 39
3.1.3 Structure de données et détail des fonctions 42
3.2 Méthodologie de vérification 45
3.2.1 Indicateurs utilisés pour l'analyse comportementale 46
3.2.2 Méthodologie de l'analyse 50
3.3 Utilisation du cadre logiciel 52
3.3.1 L'algorithme de décision 53
3.3.2 Impact des communications sur le comportement des CAVs 54
3.3.3 Impact de la coopération sur le comportement des véhicules 59
3.3.4 Limitations et gestion des paramètres 62
3.4 Résumé du chapitre 65

Ce chapitre décrit un environnement de travail pour la modélisation formelle et la vérification (*model checking*) d'un système de véhicules autonomes communicants (CAV). Cette approche met l'accent sur l'impact que les communications peuvent avoir sur la sécurité des véhicules et la fluidité du trafic. Le cadre logiciel, appelé VERIFCAR, est composé d'un modèle paramétrique d'un système de CAVs, ainsi que d'une méthode de calcul d'indicateurs permettant d'évaluer la qualité d'une politique de prise de décision de véhicule autonome. Le modèle est conçu pour représenter exhaustivement le non-déterminisme induit par la latence, les délais de communication et la concurrence entre les agents. En outre, il est optimisé pour les pro-

blématiques de vérification formelle, tant au regard du niveau d'abstraction utilisé que de la réduction de l'espace d'états.

Le formalisme de modélisation utilisé dans VERIFCAR pour spécifier le comportement des CAVs est un modèle d'automates temporisés [14]. Il s'agit d'un formalisme de référence dans le domaine de la vérification de systèmes temporels, et est utilisé par plusieurs outils de vérification, comme le model checker UPPAAL [9]. À notre connaissance, ce type d'approche ne semble pas avoir été exploité jusqu'à présent dans le cadre des véhicules autonomes.

Dans ce chapitre, nous présenterons d'abord une modélisation du système de CAVs basé sur le formalisme des réseaux d'automates temporisés. Outre le réseau d'automates et la structure de données utilisée, les choix de paramètres et de variables décrivant le système seront discutés, ainsi que la possibilité de réduire la taille de l'espace d'états en fonction d'une précision souhaitée. Par la suite, la logique temporelle utilisée dans le cadre logiciel ainsi que la méthodologie de vérification seront présentées. Cela inclura le choix des indicateurs utilisés et la façon dont ils seront calculés au cours de la vérification. Finalement, nous présentons plusieurs exemples illustrant la manière dont VERIFCAR peut être utilisé dans l'étude du comportement de véhicules autonomes en présence de communications entre véhicules, avec ou sans certaines formes de négociation ou de communication véhicule-infrastructure. Ce chapitre se conclut par une discussion sur les limites de VERIFCAR et la manière dont il devrait être pris en main par un utilisateur.

3.1 Le modèle

Chaque agent du système est représenté dans son comportement par un composant (un automate temporisé). L'état des agents (position, vitesse...)prend la forme d'une structure de données. On décrit ici l'architecture du modèle, puis on montre comment sont choisies les constantes les plus importantes du système, avant d'expliciter la forme de la structure de donnée et le détail des fonctions utilisées.

3.1.1 Les composants

Soit n le nombre d'agents dans le système. Le modèle est alors composé de $n+1$ automates : l'automate d'environnement A_0 et les automates d'agents A_i pour $1 \leq i \leq n$. Les agents peuvent être des véhicules communicants ou une infrastructure communicante (généralement un ensemble d'unités en bordure de route). Outre un ensemble de paramètres constants, la structure de données du modèle est encodée sous la forme d'un ensemble de sous-structures, chacune d'entre elles modélisant l'état d'un seul véhicule. Toutes les variables qui doivent être connues

concernant un véhicule donné (position, vitesse, connaissance de l'environnement, etc.) font partie d'une telle sous-structure. On notera que ces variables seront entières, étant donné que le formalisme des automates temporisés implique que les variables soient des nombres entiers. Parallèlement à cette structure de données, sont définies $n+1$ horloges C_i pour $0 \leq i \leq n$, et un canal de *broadcast* k. Nous considérons également trois fonctions :

— $update()$, qui met à jour l'état de tous les agents en fonction du temps, c'est-à-dire leur position (longitudinale et latérale) ainsi que leur vitesse longitudinale ;

— $decision(i)$, qui calcule la prochaine accélération et direction à appliquer pour l'agent i ;

— $communicate(i)$, qui envoie les informations communiquées par l'agent i sur ses intentions aux autres agents.

Les automates du modèle utilisé dans VERIFCAR sont représentés en Fig. 3.1.

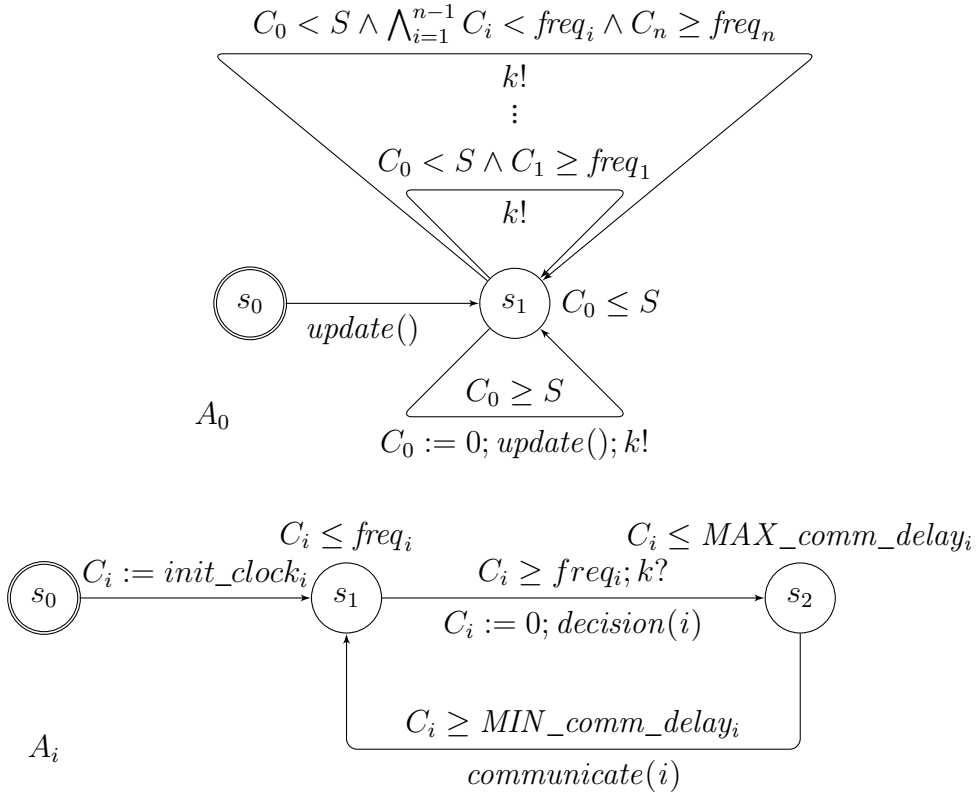

FIGURE 3.1 – VERIFCAR : patrons des automates temporisés A_0 (pour l'environnement) et A_i (pour l'agent i). Les localités initiales sont urgentes et sont représentées par un double trait.

L'automate de l'agent A_i (pour $1 \leq i \leq n$) est associé à l'horloge C_i. Le rôle de cet automate est de déclencher à la fois la décision de l'agent i et la communication de cet agent avec d'autres agents. Il est composé de trois localités s_0, s_1 et s_2. s_0 est la localité initiale : il

s'agit d'une localité urgente (ce qui signifie qu'elle doit être quittée dès que possible) et possède une transition sortante vers s_1, qui définit la valeur initiale de C_i (défini comme paramètre de l'agent) . La localité s_1 est associée à un invariant $C_i \leq \mathit{freq}_i$ où freq_i est un paramètre définissant l'intervalle de temps séparant deux prises de décision pour l'agent i. Il y a une transition de s_1 vers s_2 avec une garde $C_i \geq \mathit{freq}_i$, ce qui déclenche la fonction $\mathit{decision}(i)$ et réinitialise C_i. Cela permet que la prise de décision s'effectue exactement toute les freq_i unités de temps. La transition est également associée au récepteur de *broadcast* $k?$, dont le rôle sera expliqué ultérieurement. Enfin, la localité s_2 est associée à un invariant $C_i \leq \mathit{MIN_comm_delay}_i$ et a une transition sortante vers s_1 avec une garde $C_i \geq \mathit{MAX_comm_delay}_i$, qui déclenche la fonction $\mathit{communicate}(i)$, où $[\mathit{MIN_comm_delay}_i, \mathit{MAX_comm_delay}_i]$ est l'intervalle de temps entre l'émission et la réception de données par d'autres agents. Cet intervalle est la cause du non déterminisme du modèle. Cette modélisation permet de réutiliser une seule horloge pour deux opérations temporisées (décision et communication). Cependant, cela implique que freq_i soit supérieure ou égale à $\mathit{MAX_comm_delay}_i$. Cela ne pose pas de problème, étant donné que les retards de latence dans les protocoles véhicules à véhicules (V2V) ne dépassent pas 100ms [65, 66, 67], tandis que les modules de prise de décision cités dans la littérature utilisent souvent une fréquence de 10Hz [68, 69]. On notera que l'automate est le même pour les véhicules et les infrastructures, mais la structure des données et les actions effectuées par les fonctions de décision et de communication sont différentes.

L'automate A_0 est associé à l'horloge C_0. Son rôle est de mettre à jour à intervalles de temps réguliers (donnés par le paramètre S) l'état de tous les agents. Il est composé de deux localités s_0 (initiale) et s_1. s_0 est une localité urgente et a une transition sortante vers s_1, qui déclenche la fonction $\mathit{update}()$. Cette transition a pour rôle d'effectuer la première mise à jour qui initialise l'état de tous les agents. La localité s_1 est associée à un invariant $C_0 \leq S$ et possède une transition qui boucle sur s_1 avec une garde $C_0 \geq S$, qui déclenche la fonction $\mathit{update}()$ et ré-initialise l'horloge C_0. Cela permet la mise à jour régulière de l'état du système. Cette transition est également associée à l'émetteur de *broadcast* $k!$. Comme la mise à jour émule le mouvement continu des objets, elle devrait toujours avoir priorité sur toutes les transitions de décision disponibles en même temps [1]. En effet, il n'est pas pertinent de garder un chemin qui diffère uniquement parce que la prise de décision a été produite en considérant l'environnement avant la mise à jour. De tels chemins ne seraient présents qu'à cause des abstractions utilisées et ne représenterait donc pas une situation réelle, où l'environnement est "mis à jour" de manière continue. Le rôle du canal de *broadcast* k est d'éviter cette forme de non-déterminisme non pertinent : toutes les transitions déclenchant le processus de décision sont associées à un récepteur

1. Cela ne se produit que pour les transitions de décision, pas pour celles de communication, pour lesquelles les actions sont indépendantes de l'état des véhicules, et donc de la fonction de mise à jour.

du *broadcast* $k?$, de sorte que, en cas de simultanéité, la mise à jour est toujours déclenchée en premier.

De plus, comme un canal de *broadcast* est utilisé, tous les processus de décision disponibles dans la même unité de temps seront déclenchés simultanément. Cela permet d'éviter les états intermédiaires inutiles dans le système avec un non-déterminisme local n'ayant pas d'impact sur le comportement du système. En effet, la décision prise par un agent i est connue des autres agents uniquement après que i communique, ou en observant les changements de comportement qui survient après la mise à jour qui a suivi la décision. Par conséquent, l'ordre dans lequel les transitions concurrentes déclenchant le processus de décision sont exécutées est sans importance.

Ces transitions doivent maintenant attendre que $k!$ soit déclenché. Cependant, elles ne doivent pas nécessairement être synchronisées avec l'échantillonnage de mise à jour défini par S. Une solution à ce problème consisterait à ajouter à A_0 une transition bouclant sur s_1 avec une garde $C_0 < S \wedge \bigvee_{i=1}^{n} C_i \geq \mathit{freq}_i$ qui déclenche $k!$. Cependant, l'opérateur de disjonction $\bigvee$ n'est pas supporté par l'outil que nous utilisons. Pour émuler ce comportement sans introduire de non-déterminisme inutile, nous avons ajouté pour chaque agent une boucle sur A_0 (donc n boucles), de sorte que pour chaque $i \in [1, n]$ il y ait une transition de s_1 à s_1 avec une garde $C_0 < S \wedge \bigwedge_{j=1}^{i-1} C_j < \mathit{freq}_j \wedge C_i \geq \mathit{freq}_i$ qui déclenche $k!$. Cette modélisation peut être assimilée à un diagramme de décision binaire [70]. En effet, si au moins un agent peut effectuer son action, il existe $i \in [1, n]$ tel que $\mathit{decision}(i)$ est disponible et $\forall j \in [1, i-1]$ $\mathit{decision}(j)$ ne l'est pas. Par conséquent, une seule transition dans A_0 est disponible à tout moment, ce qui induit une progression déterministe avec cet automate.

3.1.2 Choix de modélisation, réglages et précision

Pour que le cadre logiciel soit utilisable en pratique, il est nécessaire de maintenir un équilibre entre le réalisme de la représentation et l'efficacité de la vérification du modèle. Cela implique d'avoir le plus petit espace d'états possible tout en perdant le moins d'information possible. Étant donné que les outils existants capables de vérifier formellement des modèles d'automates temporisés ne peuvent gérer que des structures de données entières (essentiellement des paramètres, variables et tableaux), nous devons proposer une discrétisation satisfaisante pour les différentes grandeurs physiques décrivant le comportement des véhicules. En effet, la complexité des procédures de vérification augmente rapidement avec le nombre de valeurs possibles de ces variables entières, tandis que la précision des modèles demande une granularité adéquate.

Notre représentation discrète de l'état des véhicules sera donc codée avec les variables dis-

crètes suivantes :

— ses positions (longitudinale et latérale) x et y,

— sa vitesse longitudinale v avec l'accélération correspondante a,

— et la direction du véhicule $D \in \{-1, 0, 1\}$, correspondant respectivement à un mouvement latéral à droite, une absence de changement et un mouvement latéral à gauche.

À l'exception de D, nous noterons Gran_a, Gran_v, Gran_x, Gran_y la granularité de ces quantités, c'est à dire l'écart entre deux valeurs consécutives (cela permet de normaliser les données sous forme d'entiers), et par N (avec les indices correspondants) la taille de la structure de données nécessaire, appelée *range*.

Les mises à jour des valeurs de vitesse et de position longitudinale après une certaine durée sont exprimées par les formules habituelles. Cependant, comme notre objectif est de pouvoir effectuer efficacement la vérification du modèle, nous avons besoin que notre modélisation soit paramétrique, ce qui permet de préserver un équilibre adéquat entre la taille de l'espace d'états à analyser et le niveau de réalisme souhaité. Ce dernier dépend directement de la durée entre deux observation du système, appelée *sample*. Ainsi, les paramètres et constantes suivants seront utilisés pour décrire le système :

— S est le sample, donné en secondes (s) ;

— L est la longueur et R est la largeur du segment de route, en mètres (m) ;

— V_{min} est le minimum et V_{max} est la valeur maximale de la vitesse (longitudinale) exprimée en mètres par seconde (m/s) ;

— A_{min} est le minimum et A_{max} est la valeur maximale d'accélération exprimée en mètres par seconde au carré (m/s^2) ;

— Gran_a est la granularité de l'accélération, exprimée en mètres par seconde au carré ;

— W est la vitesse latérale lors d'un changement de voie, exprimée en mètres par seconde.

En conséquence, l'intervalle d'accélération est alors $\mathsf{N}_a = 1 + (A_{max} - A_{min})/\mathsf{Gran}_a$, en supposant que A_{max}/Gran_a ainsi que A_{min}/Gran_a soient des entiers. L'accélération est alors exprimée sous la forme $a = A \cdot \mathsf{Gran}_a$,
la vitesse longitudinale sous la forme $v = V \cdot \mathsf{Gran}_v$,
la position longitudinale comme $x = X \cdot \mathsf{Gran}_x$,
et la position latérale comme $y = Y \cdot \mathsf{Gran}_y$,
où A, V, X, Y sont des entiers (variables normalisées sans dimensions). L'intérêt d'introduire ces variables sans dimension sera de simplifier les formules de mise à jour d'états, lorsque les granularités seront choisies de manière adéquate.

À présent, les granularités et les intervalles pour les valeurs de positions longitudinales et latérales et de vitesses peuvent être calculées, ainsi que leurs mises à jour normalisées après un sample S :

Pour la vitesse longitudinale, la mise à jour après un échantillon est

$$v' = v + a \cdot \mathsf{S}.$$

Avec les variables normalisées, cela donne $V' = V + (A \cdot \mathsf{S} \cdot \mathsf{Gran}_a)/\mathsf{Gran}_v$ pour une granularité donnée Gran_v, et $V' = V + A$ si nous choisissons la granularité $\mathsf{Gran}_v = \mathsf{Gran}_a \cdot \mathsf{S}$. Le principal avantage de ce choix est qu'il n'introduit pas de nouvelles pertes d'information et simplifie la formule. Le nombre de valeurs en résultant est de taille $\mathsf{N}_v = 1 + (V_{max} - V_{min})/(\mathsf{Gran}_a \cdot \mathsf{S})$, où une fonction de plafond est utilisée si la division ne fournit pas un entier.

Pour la position longitudinale, la mise à jour après un sample est

$$x' = x + v \cdot \mathsf{S} + a \cdot \mathsf{S}^2/2.$$

Pour une granularité donnée G_x, cela conduit, avec les variables normalisées, à $X' = X + ((V \cdot \mathsf{S} \cdot \mathsf{Gran}_v) + (A \cdot \mathsf{S}^2/2 \cdot \mathsf{Gran}_a))/G_x = X + (2 \cdot V + A)(\mathsf{Gran}_a \cdot \mathsf{S}^2/2)/G_x$ Afin d'éviter des pertes supplémentaires d'information, nous devrions choisir la granularité $G_x = \mathsf{Gran}_a \cdot \mathsf{S}^2/2$, qui donne $X' = X + 2 \cdot V + A$. Cependant, cette granularité sera généralement inutilement petite (par exemple, avec $\mathsf{Gran}_a = 1$ et $\mathsf{S} = 10^{-1}$, nous obtenons une granularité de 5 mm), ce qui conduit à untrès grand nombre de valeurs possibles. Afin d'éviter une explosion de l'espace d'états pendant le processus de vérification, nous allons donc approximer x avec une précision de $\mathsf{Gran}_x = p \cdot G_x$, avec un paramètre adéquat p. Par conséquent, la mise à jour normalisée de x devient $X' = X + (2V + A)/p$ (arrondi) et l'ensemble correspondant est de taille $\mathsf{N}_x = \mathsf{L}/\mathsf{Gran}_x$. Afin de choisir un p adapté, nous pouvons remarquer que Gran_x est la perte de précision longitudinale maximale à laquelle nous pouvons être confrontés au cours d'un sample. Par conséquent, nous pouvons introduire la perte de précision maximale normalisée [2] pendant une seconde $\mathsf{Norm}_x = \mathsf{Gran}_x/\mathsf{S} = p \cdot \mathsf{Gran}_v/2$. La valeur de Norm_x peut être fixée indépendamment des constantes du système et nous obtenons les valeurs $\mathsf{Gran}_x = \mathsf{Norm}_x \cdot S$ et $p = 2 \cdot \mathsf{Norm}_x/\mathsf{Gran}_v$.

Pour la mise à jour de la position latérale après un sample, nous avons

$$y' = y + W \cdot \mathsf{S} \cdot D$$

avec une granularité correspondante $\mathsf{Gran}_y = W \cdot \mathsf{S}$ puisque W (la vitesse latérale lorsqu'il

2. En réalité, cette perte de précision est divisée par deux, grâce à l'arrondi.

y a un mouvement latéral) est une constante. La direction du véhicule $D \in \{-1, 0, 1\}$, correspondant respectivement à un mouvement latéral à droite, une absence de changement et un mouvement latéral à gauche, l'intervalle des valeurs de direction est donc de taille $\mathsf{N}_D = 3$, tandis que l'intervalle des valeurs de la position latérale est de taille $\mathsf{N}_y = \mathsf{R}/\mathsf{Gran}_y$. Avec les variables normalisées, cela devient $Y' = Y + D$.

La taille de l'espace d'états dû aux variables est ensuite obtenue en faisant le produit des tailles de tous les ensembles ci-dessus, pour chaque véhicule, et se calcule donc comme $\mathcal{O}(\alpha^n)$, où n est le nombre de véhicules et $\alpha = \mathsf{N}_a * \mathsf{N}_v * \mathsf{N}_x * \mathsf{N}_D * \mathsf{N}_y$. Les horloges participent également à l'espace d'états, de sorte que la taille de l'espace d'états complet s'obtient en multipliant ce qui précède par une exponentielle supplémentaire, qui résulte d'une intrication du nombre d'intersections et de différences des divers intervalles de temps se produisant dans les gardes et les invariants du système, comme détaillé dans [71].

Heureusement, les outils formels n'ont pas nécessité de construire l'intégralité de l'espace d'états afin d'analyser de tels systèmes. Bien entendu, la complexité des procédures de vérification dépend aussi du degré de non-déterminisme présent dans la spécification et la difficulté à résoudre les requêtes.

3.1.3 Structure de données et détail des fonctions

Nous allons présenter le détail de la structure de données utilisée et des fonctions $update()$, $decision(i)$ et $communicate(i)$ Tout d'abord, UPPAAL nous contraignant à n'utiliser que des nombres entiers, il est nécessaire de mettre les unités à l'échelle. On a donc un paramètre $scale$ tel que les unités de temps seront en secondes / $scale$ et les unités de distance en mètres / $scale$. Les valeurs Gran_v, Gran_x, Gran_y et p sont obtenus en fonction de paramètres du système et l'intervalle de valeur de chaque variable du système est calculé sur la base des granularités, selon les calculs présentés ci-dessus. On normalise également les différents paramètres du système pour qu'ils correspondent tous aux même granularités. Ceux qui seront nécessaires pour la mise à jour lors de l'$update()$ sont :

— $LengthX$, la longueur normalisée du segment de route (égal à la longueur divisée par Gran_x) ;

— $LengthY$, la largeur normalisée du segment de route (égal à la largeur divisée par Gran_y) ;

— min_speed, la vitesse minimale normalisée des véhicules ;

— max_speed, la vitesse maximale normalisée des véhicules ;

— J_beg, la position longitudinale normalisée qui signale le début de la jonction entre la voie d'insertion et la route principale ;

— J_end, la position longitudinale normalisée qui signale la fin de la voie d'insertion ;

— J_inf, la position latérale normalisée à partir de laquelle un véhicule sur la voie d'insertion avant le début de la jonction est en collision avec le bord gauche de la route (cette position tient compte de la largeur d'un véhicule) ;

— J_sup, la position latérale normalisée à partir de laquelle un véhicule sur la voie principale avant le début de la jonction est en collision avec le bord droit de la route (cette position tient compte de la largeur d'un véhicule).

La structure de données du modèle est définie de la façon suivante. Soit nb_car le nombre de véhicules, la structure $car[i]$ contient l'ensemble des variables caractérisant l'état physique du véhicule i. Ces variables sont :

— $car[i].posX$, la position longitudinale du véhicule ;

— $car[i].posY$, la position latérale du véhicule ;

— $car[i].speed$, la vitesse longitudinale du véhicule ;

— $car[i].acceleration$, l'accélération longitudinale du véhicule ;

— $car[i].direction$, la vitesse latérale du véhicule ;

— $car[i].on_the_road$, une variable booléenne indiquant si le véhicule est présent sur la portion de route.

Additionnellement, la structure $comm[i]$ contient l'ensemble des variables liées aux données échangées et reçues par le véhicule i. Par exemple, si notre protocole de décision prévoit que les véhicules communiquent sur une variable VAR, le tableau $comm[i].VAR[]$ donne à l'indice j la dernière valeur de VAR envoyée au véhicule i par le véhicule j.

A présent que la structure de donnée est définie, nous pouvons présenter le détail des fonctions.

La fonction $update()$, qui s'applique à tous les agents simultanément, est décrite ci-dessous. La ligne 4 consiste à ajouter une unité (Gran_x) à la nouvelle position si celle ci est plus proche de l'arrondi supérieur que de l'arrondi inférieur. Les lignes 5 à 7 mettent le véhicule hors de la route si celui ci atteint la limite de la portion observée. La ligne 9 applique la vitesse et l'accélération pour déterminer la nouvelle position. La ligne 10 met le véhicule hors de la route si celui ci se trouve sur la voie d'insertion après la fin de celle ci. Les lignes 13 et 14 vérifient que la vitesse du véhicule est comprise entre min_speed et max_speed. La ligne 15 applique l'accélération pour déterminer la nouvelle vitesse. Si le véhicule se dirige vers la droite (-1), la ligne 20 retire une unité Gran_y à la position latérale actuelle. Si le véhicule se dirige vers la gauche (1), la ligne 25 ajoute une unité Gran_y à la position latérale actuelle. La ligne 28 met le

véhicule hors de la route si celui ci est en collision avec la démarcation physique entre la voie principale et la voie d'insertion.

```
1  for(id in range(nb_car)){
2
3  // Mise a jour position longitudinale
4   if((((2*car[id].speed)+car[id].acceleration)/p)*2 < (((2*car[id].speed)+
       car[id].acceleration)*2)/p and car[id].posX < LengthX) car[id].posX++;
5   if(car[id].posX + (((2*car[id].speed)+car[id].acceleration)/p) >= LengthX)
       {
6     car[id].posX := LengthX;
7     car[id].on_the_road := false;
8   }
9   else car[id].posX += ((2*car[id].speed)+car[id].acceleration)/p;
10  if(car[id].posX > J_end and car[id].posY < J_sup) car[id].on_the_road :=
       false;
11
12 // Mise a jour vitesse longitudinale
13  if(car[id].speed + car[id].acceleration > max_speed) car[id].speed :=
       max_speed;
14  else if(car[id].speed + car[id].acceleration < min_speed) car[id].speed :=
        min_speed;
15  else car[id].speed += car[id].acceleration;
16
17 // Mise a jour position laterale
18  if(car[id].direction == -1 and car[id].on_the_road){
19    if((car[id].posX <= J_end and car[id].posX >= J_beg) or car[id].posY >=
        J_sup or car[id].posY <= J_inf){
20      if(car[id].posY > 1) car[id].posY--;
21    }
22  }
23  if(car[id].direction == 1 and car[id].on_the_road){
24    if((car[id].posX <= J_end and car[id].posX >= J_beg) or car[id].posY >=
        J_sup or car[id].posY <= J_inf){
25      if(car[id].posY < LengthY) car[id].posY++;
26    }
27  }
28  if(car[id].posX < J_beg and car[id].posY < J_sup and car[id].posY > J_inf)
       car[id].on_the_road := false;
29 }
```

Le patron de la fonction $decision(id)$ est décrit ci-dessous. Les lignes 1 et 2 définissent les variables temporaires de direction et d'accélération. Leur ensembles de définitions RangeD et RangeA correspondent respectivement à N_D et N_v. Les variables nécessaires à la prise de

décision doivent être définie à la suite (ligne 4). La prise de décision n'a lieu que si le véhicule se trouve sur la portion de route. La prise de décision (ligne 8) va affecter des valeurs aux variables temporaires de direction et d'accélération. Elle ne peut écrire dans aucune variable de la structure de donnée et ne peut lire que les variables de $car[id]$ et $comm[id]$ (celles connues par le véhicule). À la ligne 10, les valeurs des variables échangées (la structure $comm[id]$) sont mises à jour. L'affectation se fait dans le cas général pour chacune des variables, mais nous n'avons qu'une seule variable dans notre exemple. Les lignes 12 et 13 affectent dans la structure de donnée les valeurs choisies pour l'accélération et la direction du véhicule.

```
1  RangeD temp_dir;
2  RangeA temp_acc;
3
4  // Declaration des variables necessaires a la prise de decision
5
6  if(car[id].on_the_road){
7
8     // Prise de decision
9
10    comm[id].VAR[id] := new_value
11
12    car[id].direction := temp_dir;
13    car[id].acceleration := temp_acc;
14 }
```

La fonction $communicate(id)$ consiste en une boucle, qui pour chaque véhicule n, va écrire dans la structure des données reçues $comm[n]$ et à l'indice correspondant pour chaque variable, les valeurs envoyées par le véhicule id.

```
1  for(n in range(nb_car)){
2     comm[n].VAR[id] := car[id].VAR[id];
3  }
```

3.2 Méthodologie de vérification

Ici, nous présenterons dans un premier temps des indicateurs et formules booléennes pouvant être utilisés pour la vérification. L'outil de *model checking* sur lequel est basé VERIFCAR, UPPAAL, utilisant un sous-ensemble de CTL pour son langage de requêtes, nous définirons dans un second temps un ensemble de requêtes CTL que nous utiliserons dans nos expériences (présentées dans la section suivante) et nous tâcherons de les généraliser afin d'obtenir une méthodologie automatisée permettant d'analyser le système en fonction de ces indicateurs.

3.2.1 Indicateurs utilisés pour l'analyse comportementale

Comme cela a déjà été mentionné, le but de VERIFCAR est de permettre d'évaluer et de comparer les algorithmes de prise de décision des CAVs par rapport à des propriétés telles que la sécurité, l'efficacité, le confort ou l'équité. Chacun des aspects mentionnés sera vérifié avec différents indicateurs appropriés. Pour vérifier la sécurité, nous proposons un indicateur de temps avant collision (TTC), qui donne, pour un temps donné t et deux véhicules, le temps avant la collision si les deux véhicules continuent de rouler à vitesse constante. Le TTC est un indicateur couramment utilisé dans la littérature [5, 6] lors de l'évaluation de la sécurité des véhicules sur une seule voie. Ici, il est adapté à un espace à deux dimensions.

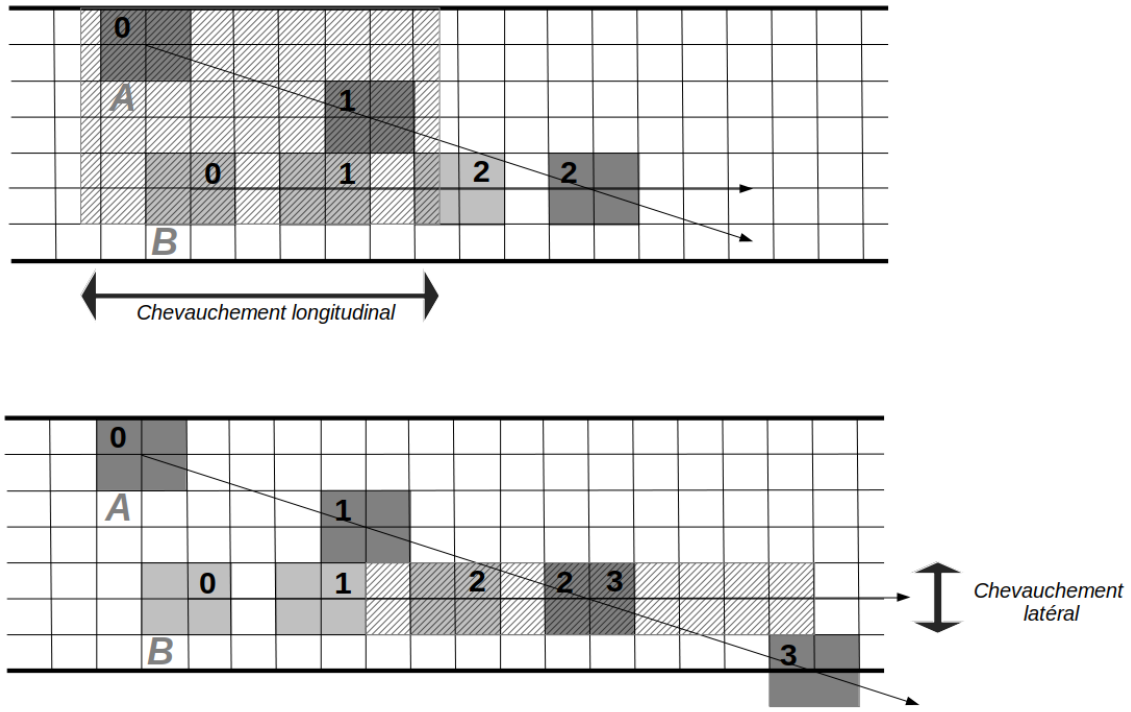

FIGURE 3.2 – Illustration des chevauchements pour l'axe x (en haut) et l'axe y (en bas). Le véhicule A, plus rapide, est en gris foncé tandis que le véhicule B est en gris clair. Les positions des véhicules sont indiquées à des dates choisies, indiquées en haut à gauche pour A et en haut à droite pour B. Les zones hachurées correspondent aux zones de chevauchement.

Pour deux points mobiles sur un axe, l'instant t_{meet} où ils se rejoignent est donné par leur distance actuelle divisée par leur différence de vitesse. L'algorithme que nous avons développé pour VERIFCAR généralise cette idée pour deux véhicules de taille variable se déplaçant sans rotation sur une grille à deux dimensions.

Lorsqu'un véhicule est suivi à une certaine distance par un véhicule plus rapide, on peut définir pour les directions longitudinales et latérales (c'est à dire, sur les axes x et y) un in-

tervalle de temps $[t_{meet}, t^{meet}]$ correspondant au chevauchement des deux véhicules : t_{meet} est l'instant où les extrémités les plus proches (dans la direction considérée) coïncident, et t^{meet} est la même chose pour les extrémités les plus éloignées. Le TTC est ensuite obtenu en comparant les intervalles de temps obtenus pour les deux directions :

— Si les deux intervalles de temps résultants ont une intersection non vide, le bord gauche de cette intersection donne la valeur TTC. Si cette valeur TTC est égale à zéro, cela signifie qu'une collision se produit actuellement entre les véhicules.

— S'il n'y a pas d'intersection entre ces intervalles, il n'y a pas de collision possible dans le futur vis-à-vis du comportement actuel et, par convention, la valeur TTC est considérée comme infinie.

À titre d'exemple, considérons la situation illustrée en Figure 3.2. Initialement, le véhicule A est en position (3,6) avec une valeur de vitesse longitudinale de 5 et une valeur de vitesse latérale de -2, tandis que le véhicule B est en position (4,2) avec une valeur de vitesse longitudinale de 3 et une valeur de vitesse latérale de 0. Les longueurs et largeurs des véhicules ont une valeur de 2 unités. Le chevauchement temporel sur l'axe x est de $[-0.5, 1.5]$, tandis que le chevauchement temporel sur l'axe y est de $[1, 3]$. L'intersection de ces intervalles ($[1, 1.5]$) est la fenêtre temporelle durant laquelle les véhicules se chevauchent dans l'environnement à deux dimensions, voir la figure 3.3. La valeur de la bordure gauche donne le TTC (ici, TTC $= 1$).

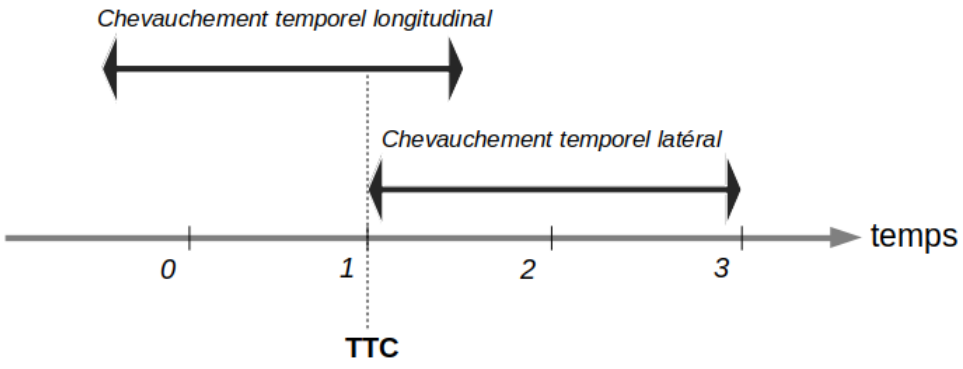

FIGURE 3.3 – Illustration du calcul du TTC basé sur des intervalles de temps longitudinaux et latéraux.

Quand on considère un chemin complet de l'espace d'états, on peut calculer le plus petit TTC sur ce chemin. Si on considère tous les chemins, on obtient une valeur minimale et maximale pour ce plus petit TTC. La complexité du calcul du TTC est en temps constant, ce qui est intéressant au regard de notre nécessité de réduction du temps de vérification.

Le détail du calcul du TTC est donné en Algorithme 1, où px_i (respectivement py_i) est la position longitudinale (respectivement latérale) du véhicule i, et $Long$ (respectivement Lat) est l'écart longitudinal (respectivement latéral) en dessous duquel se produit une collision. De même, vx_i (respectivement vy_i) est la valeur de la vitesse longitudinale (respectivement la vitesse latérale) du véhicule i. $Long$ et Lat dépendent de la longueur et de la largeur de chaque véhicule : généralement, la position étant centrée sur le véhicule, $Long$ sera la moyenne de la longueur des deux véhicules, tandis que Lat sera la moyenne de la largeur des deux véhicules. L'algorithme consiste à calculer les moments de début X_{in} et de fin X_{out} du chevauchement longitudinal (respectivement Y_{in} et Y_{out} pour le chevauchement latéral) . Si aucun chevauchement (longitudinal ou latéral) ne se produira jamais, le moment de début se verra attribuer la valeur ∞ et le moment de fin sera fixé à 0. Après quoi, le calcul du TTC consiste a retourner la plus petite valeur de l'intersection entre les chevauchements longitudinal et latéral.

Algorithm 1 Calcul du TTC entre deux véhicules A et B.

{Calcul de X_{in}}
if $(px_b \geq px_a$ **and** $vx_b < vx_a)$ **or** $(px_a \geq px_b$ **and** $vx_a < vx_b)$ **then**
  $X_{in} = \frac{|px_b - px_a| - Long}{|vx_a - vx_b|}$ {A et B se rapprochent}
else if $|px_b - px_a| < Long$ **then**
  $X_{in} = 0$ {Chevauchement déjà présent}
else
  $X_{in} = \infty$ {Chevauchement impossible}
end if
{Calcul de X_{out}}
if $(px_b \geq px_a$ **and** $vx_b < vx_a)$ **or** $(px_a \geq px_b$ **and** $vx_a < vx_b)$ **then**
  $X_{out} = \frac{|px_b - px_a| + Long}{|vx_a - vx_b|}$ {A et B se rapprochent}
else if $vx_a \neq vx_b$ **then**
  $X_{out} = \frac{||px_b - px_a| - Long|}{|vx_a - vx_b|}$ {A et B s'éloignent}
else if $|px_b - px_a| < Long$ **then**
  $X_{out} = \infty$ {Chevauchement déjà présent}
else
  $X_{out} = 0$ {Chevauchement impossible}
end if

{Calcul de Y_{in}}
if $py_b \geq py_a$ **and** $vy_b < vy_a$ **then**
  $Y_{in} = \frac{py_b - py_a - Lat}{vy_a - vy_b}$ {B à gauche de A, A et B se rapprochent}
else if $py_a \geq py_b$ **and** $vy_a < vy_b$ **then**
  $Y_{in} = \frac{py_a - py_b - Lat}{vy_b - vy_a}$ {A à gauche de B, A et B se rapprochent}
else if $|py_b - py_a| < Lat$ **then**
  $Y_{in} = 0$ {Chevauchement déjà présent}
else
  $Y_{in} = \infty$ {Chevauchement impossible}
end if
{Calcul de Y_{out}}
if $py_b \geq py_a$ **and** $vy_b < vy_a$ **then**
  $Y_{out} = \frac{py_b - py_a + Lat}{vy_a - vy_b}$ {B à gauche de A, A et B se rapprochent}
else if $py_b \geq py_a$ **and** $vy_a \neq vy_b$ **then**
  $Y_{out} = \frac{py_b - py_a - Lat}{vy_a - vy_b}$ {B à gauche de A, A et B s'éloignent}
else if $py_a \geq py_b$ **and** $vy_a < vy_b$ **then**
  $Y_{out} = \frac{py_a - py_b + Lat}{vy_b - vy_a}$ {A à gauche de B, A et B se rapprochent}
else if $vy_a \neq vy_b$ **then**
  $Y_{out} = \frac{py_a - py_b - Lat}{vy_b - vy_a}$ {A à gauche de B, A et B s'éloignent}
else if $|py_b - py_a| < Lat$ **then**
  $Y_{out} = \infty$ {Chevauchement déjà présent}
else
  $Y_{out} = 0$ {Chevauchement impossible}
end if
{Calcul du TTC}
if $X_{in} \leq Y_{in}$ **and** $Y_{in} \leq X_{out}$ **then**
  return Y_{in} {Intersection des chevauchements}
else if $Y_{in} \leq X_{in}$ **and** $X_{in} \leq Y_{out}$ **then**
  return X_{in} {Intersection des chevauchements}
else
  return ∞ {Pas d'intersection entre les chevauchements temporels}
end if

L'efficacité, le confort ou l'équité peuvent être vérifiés à l'aide d'indicateurs tels que le temps de trajet (*Travel Time* [28, 29] en anglais), la décélération maximale ou le temps d'attente. On peut choisir de vérifier les extremums, la valeur moyenne ou la covariance entre les agents pour de tels indicateurs. De manière générale, les indicateurs habituels utilisés dans la littérature, tels que ceux décrits dans [72] (par exemple, le temps d'attente, la consommation de carburant, le temps perdu, etc.) peuvent être implémentés dans VERIFCAR.

3.2.2 Méthodologie de l'analyse

Il nous est possible de vérifier les valeurs prises par n'importe quelle variable présente dans la structure de données du modèle. Par exemple, pour vérifier s'il est possible que le véhicule i atteigne la voie numéro 2 on peut utiliser une requête de la forme $EF\ vehicule[i].lane = 2$. En outre, la formule propositionnelle de la propriété peux prendre la forme d'une fonction (même complexe) tant que celle ci est booléenne (évaluée comme vraie ou fausse). Etant donné que nous étudions des scénarios complexes impliquant du non-déterminisme, différentes exécutions du système mènent souvent à des états différents. Par conséquent, nous ne devrions pas considérer une seule valeur pour un indicateur donné mais un ensemble de valeurs possibles.

Pour chaque exécution, il existe une plus petite valeur pour un indicateur $indic$. Nous appelons $indic_{min}$ l'ensemble de ces plus petites valeurs (chacune pour une exécution).

Soit $k \stackrel{\text{df}}{=} \inf(indic_{min})$ et $k' \stackrel{\text{df}}{=} \sup(indic_{min})$, alors k est la plus petite valeur possible qui satisfasse

$$Q^{EF}_{min}(k) \stackrel{\text{df}}{=} EF\ indic \leq k$$

et k' est la plus petite valeur possible qui satisfasse

$$Q^{AF}_{min}(k') \stackrel{\text{df}}{=} AF\ indic \leq k'.$$

Cela implique que $EF\ indic \leq k - 1$ et $AF\ indic \leq k' - 1$ doivent être faux.

De manière similaire, nous appelons $indic_{max}$ l'ensemble des plus grandes valeurs de $indic$ pour les différentes exécutions possibles.

Soit $k \stackrel{\text{df}}{=} \inf(indic_{max})$ et $k' \stackrel{\text{df}}{=} \sup(indic_{max})$, alors k est la plus grande valeur possible qui satisfasse

$$Q^{AF}_{max}(k) \stackrel{\text{df}}{=} AF\ indic \geq k$$

et k' est la plus grande valeur possible qui satisfasse

$$Q^{EF}_{max}(k') \stackrel{\text{df}}{=} EF\ indic \geq k'.$$

Cela implique que $AF\ indic \geq k+1$ and $EF\ indic \geq k'+1$ doivent être faux.

Pour trouver ces extrema, nous utilisons un algorithme de dichotomie classique en conjonction avec des requêtes de vérification de modèle. Définissons $Q(n)$ comme étant soit $Q^{EF}_{min}(n)$, soit $Q^{AF}_{min}(n)$ et $[i,j]$ comme l'intervalle des valeurs possibles pour n. L'algorithme 2 décrit la procédure permettant de déterminer la plus petite valeur possible qui satisfait $Q(n)$. Le nombre maximal de requêtes nécessaires pour trouver cette valeur est $\log_2(j-i)$.

Si $Q(n)$ est soit $Q^{EF}_{max}(n)$, soit $Q^{AF}_{max}(n)$, nous utilisons le même algorithme en ajoutant une négation à la condition : $if(\neg\, Q(\frac{u+v}{2}))$ et en retournant u au lieu de v.

Algorithm 2 Calcul de la valeur minimale de l'indicateur $indic$ pour laquelle $Q(n)$ est vrai. $[i,j]$ est l'intervalle des valeurs possibles pour n, avec $i < j$. Il est entendu que Q est monotone, que $Q(j)$ est vrai et $Q(i)$ ne l'est pas.

$u \leftarrow i$
$v \leftarrow j$
while $u \neq v-1$ **do**
  if $Q(\frac{u+v}{2})$ **then**
    $v \leftarrow \frac{u+v}{2}$
  else
    $u \leftarrow \frac{u+v}{2}$
  end if
end while
return v

Outre ces indicateurs numériques, l'ordre d'arrivée des véhicules peut donner des informations supplémentaires sur leurs comportements. Nous définissons une fonction booléenne notée $before(v_1, v_2)$, prenant deux véhicules v_1 et v_2 comme arguments et étant évaluée comme vraie dans un état si x a atteint la fin de la route et que y ne l'a pas encore fait. Pour trouver tout les ordres d'arrivée possibles, nous vérifions pour chaque paire de véhicules A et B les requêtes suivantes : $EF\ before(A,B)$ et $EF\ before(B,A)$. Pour chaque paire de requêtes, nous avons trois résultats possibles :

— Les deux sont vraies, A arrive en premier dans certaines exécutions et B en premier dans d'autres.

— Une seule requête est vraie, l'ordre ne change jamais pour la paire dans toutes les exécutions possibles.

— Aucune n'est vraie, les deux véhicules arrivent toujours au bout de la route exactement au même moment.

Grâce à ces simples requêtes d'atteignabilité, nous parvenons à visualiser les comportements possibles pouvant survenir dans le système. Dans certains cas de comportements hautement non

déterministes, une requête plus approfondie pourrait être utile pour vérifier les ordres d'arrivée entre plus de deux véhicules. Ce cas sera illustré en 3.3.3.

3.3 Utilisation du cadre logiciel

Dans cette section, nous allons illustrer comment VERIFCAR peut être utilisé pour l'analyse de politiques de prise de décision des CAVs. Nous nous concentrons sur deux scénarios non déterministes, qui permettent de mettre avant la manière dont les paramètres de communication (et en particulier les paramètres temporels) sont impliqués dans le comportement des véhicules. Tout d'abord, nous soulignons l'impact des délais de communication sur le non-déterminisme du système, c'est à dire les délais entre la diffusion et la réception. Ensuite, en injectant des défauts dans le fonctionnement des émetteurs et des récepteurs, nous étudions la robustesse d'une politique de décision face à de tels disfonctionnements. Enfin, nous comparons trois politiques de décision :

— utiliser uniquement les communications simples entre véhicules ;

— utiliser la négociation entre véhicules ;

— utiliser des infrastructures routières intelligentes.

Ces différentes prises de décision sont comparés en fonction de critères de sécurité et d'efficacité (le temps dont les véhicules ont besoin pour atteindre leur objectif).

Les résultats de nos expériences, obtenus à l'aide des requêtes présentées dans la section 3.2.2, sont rapportés dans les tables ci-dessous avec la signification suivante :

— L'ordre d'arrivée entre deux véhicules indique lequel arrive le premier. En cas de non déterminisme, cela peut donner une meilleure idée du comportement des véhicules que les temps de trajet seuls, étant donné que cela indique si un véhicule est capable d'en dépasser un autre.

— Les temps de parcours d'un véhicule (il s'agit du moment où le véhicule quitte la portion de route) indique les valeurs minimales et maximales possibles pour tous les scénarios possibles.

— La plus petit TTC pour deux véhicules est la valeur minimale du TTC pour une trace. Pour toutes les traces possibles à partir du même état initial, une valeur minimale et une valeur maximale du plus petit TTC sont calculées. Le minimum correspond au scénario le plus dangereux et le maximum au scénario le moins dangereux.

Les scénarios que nous utilisons dans les expériences impliquent trois véhicules évoluant sur un tronçon de route à grande vitesse à trois voies de $\mathsf{L} = 500$ m de long et $\mathsf{R} = 10,5$ m de large

avec deux voies (gauche et droite) et une voie d'insertion qui commence au début de la section, rejoint la voie de droite après 200 m et se termine 200 m plus tard. Les contraintes définissant la vitesse des véhicules sont $V_{min} = 0$ m / s et $V_{max} = 40$ m / s (= 144 km/h) ; $A_{min} = -5$ m/s^2 et $A_{max} = 3$ m/s^2, et $W = 1$ m/s. Nous choisissons $\mathsf{Gran}_a = 1$ m/s^2 et la période de mise à jour de l'environnement $\mathsf{S} = 0.1$ s (10 Hz). Un tel S permet de surveiller le comportement du système de manière satisfaisante et une telle Gran_a donne un nombre suffisant de choix d'accélération pour les algorithmes de décision des véhicules. On peut remarquer que c'est un choix judicieux car il conduit également à de bonnes divisions entières dans les formules que nous avons exprimées dans la section 3.1.2. Avec de tels paramètres et la granularité ne garantissant aucune perte d'information supplémentaire, c'est-à-dire $G_x = 0,005$ m, la complexité de la structure de données par véhicule est de l'ordre de 10^{11} et elle devient $(10^{11})^3$ pour notre scénario à 3 voitures, ce qui pourrait poser problème pour les outils de vérification. Nous choisissons donc une précision normalisée raisonnable $\mathsf{Norm}_x = 1$, ce qui signifie que la perte de précision sur la position longitudinale du véhicule en une seconde est toujours inférieure à 1 mètre (0,5 mètre en arrondissant) et nous obtenons $p = 2 \cdot q/\mathsf{Gran}_v = 20$. Par conséquent, la granularité sur x devient $\mathsf{Gran}_x = p \cdot G_x = 0.1$ m et cette approximation permet de diviser l'espace d'états par $p^3 = 8000$ sans impacter significativement le comportement du modèle.

3.3.1 L'algorithme de décision

Notre algorithme de décision a pour objectif qu'un véhicule suive son itinéraire le plus rapidement possible tout en respectant les règles du code de la route et en évitant les collisions avec d'autres véhicules. L'algorithme de décision est local (chaque véhicule prend sa décision de son coté) et prend comme entrée les connaissances globales que le véhicule possède sur lui-même et sur les autres. Il calcule les trajectoires temporelles de ses voisins, permettant ainsi de calculer en sortie sa nouvelle accélération et direction. L'algorithme doit éviter de futures collisions avec les véhicules de devant, mais pas derrière tant qu'il n'y a pas de danger immédiat (les véhicules devant sont ainsi prioritaires par rapport à ceux qui se trouvent derrière). Le comportement attendu est que les véhicules s'adaptent aux actions de ceux qui les précèdent.

L'algorithme 3 donne une version de haut niveau de la fonction de décision principale où $[MinAcc, MaxAcc]$ représente l'ensemble de toutes les valeurs d'accélération possibles, $[1, NbLanes]$ représente l'ensemble de toutes les voies possibles et $[0, MaxDelay]$ représente l'ensemble de toutes les valeurs de délai possibles. La fonction de décision cherche une trajectoire appropriée avec les objectifs suivants (par ordre de priorité) :

— aller aussi vite que possible,

— être aussi proche que possible de la voie définie par son itinéraire,

— retarder le moins possible les changements de direction

Intuitivement, un délai non nul signifie que le véhicule doit changer de voie, mais que sa meilleure option est de le faire plus tard : la décision est donc de ne pas changer de voie en attendant la prochaine prise de décision.

L'implémentation complète, ainsi que tout le matériel source, sont disponibles sur [73]. Cet algorithme a été implémenté pour les études de cas et est donné pour une meilleure compréhension de la présente section, mais ne doit pas être considéré comme l'une de nos contributions principales : notre objectif est d'illustrer la méthodologie de vérification et non de promouvoir un algorithme décisionnel.

Algorithm 3 Pseudo-code de haut niveau pour la fonction de prise de décision

```
Calcul de l'ensemble des trajectoires temporelles des véhicules de devant
Choix de la voie L à atteindre en fonction de la navigation
for Acceleration ← MaxAcc to MinAcc do
  for Lane ∈ [1, NbLanes], en commençant par L et en explorant le voisinage de L do
    for Delay ← 0 to MaxDelay do
      if Le comportement choisi ne génère pas de conflit avec les véhicules se trouvant
      devant then
        if Delay = 0 (Attente non nécessaire) then
          Choix de la Direction en fonction de la Lane choisie
        else
          Direction ← 0 (Le véhicule va tout droit car il est en attente)
        end if
        Renvoyer les nouvelles valeurs d'Acceleration et de Direction
      end if
    end for
  end for
end for
Comportement d'urgence
```

3.3.2 Impact des communications sur le comportement des CAVs

Dans cette sous-section, nous nous concentrons sur le processus de décision n'impliquant pas de coopération (sans négociation ni infrastructure). Ainsi, chaque véhicule diffuse sa trajectoire prévue sans essayer d'avoir un impact direct sur le comportement des autres véhicules. Bien sûr, il reçoit également les informations de ses voisins et en tiens compte dans son processus de décision.

Nous analysons le Scénario 1, dans lequel le véhicule A est initialement sur la voie de droite à la position 50 m avec une vitesse de 20 m/s, le véhicule B sur la voie de droite à la position 0

m avec une vitesse de 35 m/s et le véhicule C sur la voie d'insertion en position 20 m avec une vitesse de 28,2 m/s. Tous les véhicules ont pour objectif d'atteindre la voie de droite au bout de la portion de route, cf. Figure 3.4.

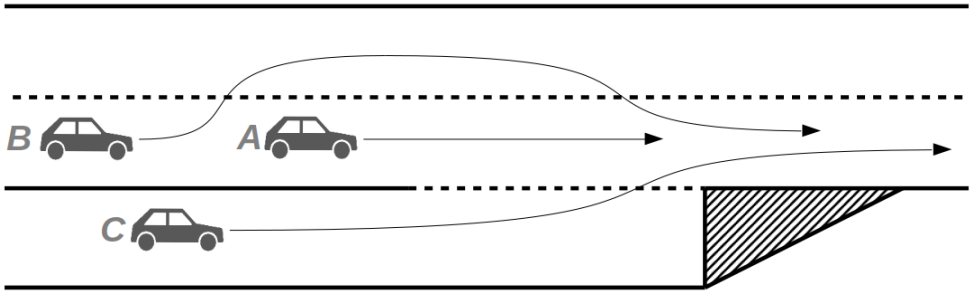

FIGURE 3.4 – Position initiale et trajectoires possibles des CAVs dans le Scénario 1.

Nous supposons que les communications véhicule à véhicule prennent entre 30 ms et 40 ms dans le cas de référence. Toutes les décisions des véhicules ont une fréquence d'activation de 10 Hz. Afin d'éviter des comportements synchrones irréalistes, les horloges des agents A, B et C sont initialisées respectivement à 0 ms, 30 ms et 70 ms. Ce type de décalage entre les horloges (qui est une situation de la vie réelle dans laquelle les véhicules ne synchronisent pas leurs horloges) induit un non-déterminisme dans la mesure où un véhicule peut prendre une décision après ou avant la réception d'une information critique. On notera que, comme tous les véhicules ont la même fréquence de prise de décision, un scénario dans lequel les horloges seraient initialisées avec la même valeur aurait un comportement déterministe, car la réception des données se produirait toujours entre deux décisions, quel que soit le moment où elle se produit dans l'intervalle de temps possible.

Délais de communication

Dans la Table 3.1, on peut observer les différences sur le scénario 1 lorsque différents intervalles de temps pour les délais de communication sont utilisés. L'exploration complète de l'espace d'états indique le temps nécessaire à construire et explorer pleinement l'espace d'états une fois, c'est-à-dire pour une seule exécution avec UPPAAL. Certains indicateurs, tels que le temps de parcours ou le plus petit TTC, nécessitent des recherches dichotomiques, et donc plusieurs exécutions avec UPPAAL, mais le nombre d'exécutions ne dépend pas de la taille de l'espace d'états et est généralement petit. De plus, l'espace d'états est mémorisé et toutes les exécutions n'ont pas besoin de l'explorer intégralement. Par exemple, le calcul du temps de parcours (le plus coûteux des indicateurs que nous avons pris en compte, en termes de temps de calcul), prend environ 45 s pour obtenir les valeurs $\inf$ et $\sup$ de la durée du trajet lorsque le temps total d'exploration est égal à 11 s ; de même, il prend environ 180 s lorsque le temps total

Indicateur		Variation du délai de communication		
		$[30, 40]$ms	$[40, 40]$ms	$[0, 90]$ms
Ordre d'arrivée	A vs B	B	B	B
	A vs C	C	C	C
	B vs C	$B\|C$	B	$B\|C$
Temps de parcours [s]	A	$(13.1, 13.2)$	$(13.1, 13.1)$	$(13.0, 13.2)$
	B	$(12.7, 13.0)$	$(12.7, 12.7)$	$(12.7, 13.0)$
	C	$(12.6, 12.8)$	$(12.8, 12.8)$	$(12.6, 12.8)$
Plus petit TTC [s]	A and B	$(1.03, 2.90)$	$(2.90, 2.90)$	$(1.03, 2.90)$
	A and C	(∞, ∞)	(∞, ∞)	$(2.80, \infty)$
	B and C	$(1.50, \infty)$	(∞, ∞)	$(1.50, \infty)$
Exploration complète de l'espace d'états [s]		11	4	53

TABLE 3.1 – Comparaison de l'ordre d'arrivée, du temps de parcours et du TTC dans le scénario 1 pour trois variantes d'intervalle du délai de communication. Pour l'ordre d'arrivée, X vs Y dans la colonne 2 indique que nous vérifions qui est le plus rapide entre X et Y, tandis que le gagnant est indiqué dans les colonnes 3-5. Pour le temps de parcours et le TTC, les paires des colonnes 3 à 5 correspondent aux valeurs $\inf$ et $\sup$ résultantes.

d'exploration est égal à 53 s, ce qui correspond à peu près au même ordre de grandeur (facteur 4).

Le cas de référence, où le délai de communication est de [30,40] ms, illustre un comportement non déterministe car plusieurs indicateurs indiquent différentes valeurs possibles, ce qui signifie qu'il existe différents chemins menant à des résultats différents. En comparant les valeurs de temps de parcours, nous pouvons constater que sa valeur pour A est toujours plus grande que pour B ou C, ce qui signifie que A sera toujours derrière les deux autres véhicules à la fin de la route. Ceci est confirmé par l'indicateur d'ordre d'arrivée. La valeur de temps de parcours $\inf$ pour B est comprise entre les valeurs de temps de trajet $\inf$ et $\sup$ de C. Cependant, les valeurs $\inf$ et $\sup$ de B sont respectivement supérieures aux valeurs $\inf$ et $\sup$ de C. Par conséquent, il se pourrait que B soit toujours derrière C. Ceci peut être vérifié grâce à l'ordre d'arrivée entre B et C, qui indique qu'il existe des chemins où B atteint la fin de la route avant C et des chemins où c'est l'inverse. Enfin, le plus petit TTC nous donne des indications sur la sécurité (une valeur ∞ signifie qu'il n'y a jamais eu de risque de collision, sinon une valeur faible signifie que nous étions proches d'une collision). Cela est utile pour analyser l'impact des paramètres de décision ou pour comparer les politiques de décision.

Ici, en plus du cas de référence, nous vérifions deux variations des délais de communication : une où on réduit l'intervalle à [40,40] ms (le délai est constant, ce qui réduit le non déterminisme) et une où nous l'étendons à [0,90] ms. Comme prévu, toutes les valeurs $\inf$ et

$\sup$ obtenues lorsque nous réduisons l'intervalle sont bornés par les valeurs du cas de référence, alors que lorsque nous étendons l'intervalle, toutes valeurs du cas de référence sont bornés par les nouvelles valeurs de $\inf$ et $\sup$. Cela est cohérent vis-à-vis du temps nécessaire à l'exploration complète de l'espace d'états, qui augmente proportionnellement à l'intervalle, comme indiqué par la dernière ligne de la table. Plus précisément, l'intervalle de [40,40] ms semble donner un scénario déterministe avec un seul chemin. Il convient de mentionner que l'utilisation d'une seule valeur de délai possible réduit considérablement le non-déterminisme, mais ne le supprime pas nécessairement. En fait, il peut encore y avoir des états où il existe une concurrence entre les actions. Par exemple, si une communication et une décision sont disponibles en même temps, l'ordre dans lequel les actions sont effectuées peut conduire à des états différents.

Évaluation de la robustesse de la prise décision par injection de fautes

Indicateur		Capteur de A désactivé		Capteur de B désactivé	
		$DLS = 0.5$m	$DLS = 1$m	$DLS = 0.5$m	$DLS = 1$m
Ordre d'arrivée	A vs B	B	B	B	B
	A vs C	C	C	C	C
	B vs C	C	C	$B\|C$	$B\|C$
Temps de parcours [s]	A	$(13.3, 13.3)$	$(13.4, 13.4)$	$(13.3, 13.3)$	$(13.3, 13.3)$
	B	$(13.0, 13.0)$	$(13.0, 13.0)$	$(12.7, 12.9)$	$(12.7, 12.9)$
	C	$(12.6, 12.6)$	$(12.6, 12.6)$	$(12.6, 13.0)$	$(12.6, 13.0)$
Plus petit TTC [s]	A and B	$(3.00, 3.00)$	$(3.00, 3.00)$	$(\mathbf{0.00}, \mathbf{0.00})$	$(2.80, 2.80)$
	A and C	(∞, ∞)	(∞, ∞)	$(2.86, \infty)$	$(5.04, \infty)$
	B and C	$(1.50, 1.50)$	$(1.50, 1.50)$	$(0.15, \infty)$	$(0.40, 1.35)$
Exploration complète de l'espace d'états [s]		5	5	10	11

TABLE 3.2 – Comparaison de l'ordre d'arrivée, du temps de parcours et du TTC sur le Scénario 1 avec des récepteurs de communication désactivés et des variations de la distance de sécurité latérale (DLS). Les autres notations sont comme dans la Table 3.1.

Les Tables 3.2 et 3.3 montrent les résultats de la vérification avec injection de fautes, en désactivant respectivement le récepteur ou l'émetteur de l'un des véhicules. Les capteurs du véhicule fonctionnent toujours, ce qui signifie que le véhicule reçoit toutes les autres informations perceptibles (telles que la position, la vitesse, l'accélération, etc.) des autres véhicules. L'information non perceptible ou faussée concerne l'intention des véhicules (leurs trajectoires prévues), c'est à dire les données envoyées par message de véhicules à véhicules.

Lorsque nous désactivons le récepteur du véhicule A, le comportement semble déterministe, C étant toujours le premier à atteindre la fin de la route. Lorsque nous désactivons le récepteur du véhicule B, le comportement est proche du cas de référence. Cependant, on peut observer que le plus petit TTC entre A et B devient 0.0 s, ce qui signifie qu'il y a une possibilité de

Indicateur		Emmeteur de A désactivé		Emmeteur de B désactivé		
		DLS=0.5 m	DLS=1 m	DLS=0.5 m	DLS=1 m	DLS=1.6 m
Ordre d'arrivée	A vs B	B	B	B	B	A
	A vs C	A	A	C	C	A
	B vs C	B	B	$B\|C$	$B\|C$	B
Temps de parcours [s]	A	$(13.3, 13.3)$	$(13.3, 13.3)$	$(13.4, 13.4)$	$(13.4, 13.4)$	$(13.0, 13.0)$
	B	$(12.7, 12.7)$	$(12.7, 12.7)$	$(12.7, 13.1)$	$(12.7, 13.1)$	$(13.1, 13.1)$
	C	$(13.5, 13.5)$	$(13.5, 13.5)$	$(12.6, 12.8)$	$(12.6, 12.8)$	$(13.4, 13.4)$
Plus petit TTC [s]	A and B	$(\mathbf{0.00}, \mathbf{0.00})$	$(2.80, 2.80)$	$(0.30, 0.30)$	$(2.09, 3.00)$	$(1.50, 1.50)$
	A and C	$(2.06, 2.06)$	$(2.06, 2.06)$	(∞, ∞)	(∞, ∞)	$(2.22, 2.22)$
	B and C	(∞, ∞)	(∞, ∞)	$(\mathbf{0.00}, \infty)$	$(\mathbf{0.00}, \infty)$	$(4.30, 4.30)$
Exploration complète de l'espace d'états [s]		6	6	12	12	10

TABLE 3.3 – Comparaison de l'ordre d'arrivée, du temps de parcours et du TTC sur le Scénario 1 avec des émetteurs de communication désactivés et des variations de la distance de sécurité latérale (DLS). Les autres notations sont comme dans la Table 3.1.

collision entre ces deux véhicules. En outre, un état où le plus petit TTC entre B et C est de 0.15 s peut être atteint, indiquant un danger sérieux. Cela peut être dû au manque d'informations que le véhicule B a sur l'intention des autres véhicules lors de son dépassement. Pour obliger B à davantage de précautions, nous augmentons la distance de sécurité latérale (DLS) dans le processus de décision de 0.5 m à 1 m. Ceci contribue à augmenter la sécurité au détriment de l'efficacité. Dans le cas du récepteur désactivé sur le véhicule A, on peut effectivement constater une légère perte d'efficacité (le véhicule A prend maintenant un peu plus longtemps pour atteindre la fin de la route). Dans le cas du récepteur désactivé sur le véhicule B, nous n'observons pas une perte d'efficacité, mais une augmentation globale de la valeur minimale du plus petit TTC, y compris des valeurs nulles.

Les mêmes propriétés sont vérifiées lors de la désactivation cette fois de l'émetteur de données et non du récepteur (Table 3.3). Lors de la désactivation de l'émetteur du véhicule A, nous observons une collision entre les véhicules A et B, qui ne se produit plus avec la DLS de 1 m. Lors de la désactivation de l'émetteur du véhicule B, nous obtenons le plus petit TTC de $(0.00, \infty)$, ce qui signifie qu'il existe au moins un chemin menant à une collision entre les véhicules B et C et au moins un autre chemin sans danger. L'augmentation de la DLS à 1 m n'empêche pas la collision. Cela nous motive à rechercher l'augmentation minimale de la DLS nécessaire pour faire face à cette collision. Pour ce faire, nous avons procédé par dichotomie, comme pour les autres indicateurs. Le résultat obtenu est qu'il est nécessaire d'avoir une DLS d'au moins 1.6 m, ce qui évite les collisions mais au prix d'une grande perte d'efficacité. En effet, dans ce scénario, on peut constater qu'aucun des véhicules ne peut désormais dépasser le véhicule A (le plus lent), ce qui entraîne la moins bonne efficacité globale de nos résultats.

3.3.3 Impact de la coopération sur le comportement des véhicules

Dans cette section, nous étudions la politique de décision mettant en œuvre les types de coopération définies dans la section 1.2, à savoir la négociation et la médiation par infrastructures. Nous décrivons d'abord comment elles ont été implémentées.

Implémentation des négociations

Dans ce type de coopération, chaque véhicule essaie de négocier à chaque fois que sa trajectoire souhaitée entrera en conflit avec la trajectoire prévue d'un autre véhicule en train de changer de voie. Cela se produit lorsqu'un véhicule plus lent change de voie devant un autre plus rapide. Le véhicule va alors prendre sa décision en considérant le conflit, mais va également communiquer au véhicule concerné qu'il est une source de conflit. Pour que l'espace des états soit aussi petit que possible, la demande de négociation (ou son absence) est simplement indiquée dans la diffusion par une variable booléenne. Lorsque le véhicule ciblé reçoit les informations, il stoppera sa manœuvre pendant un certain temps si le délai nécessaire à la résolution du conflit n'a pas d'incidence sur la finalité de sa propre manœuvre et permet au véhicule plus rapide de le dépasser. Le comportement émergent est qu'aucun des véhicules ne ralentira, à moins que d'autres conflits existent et ne permettent la modification de trajectoire conflictuelle. En effet, si on considère l'algorithme 3, le véhicule initie la négociation après avoir calculé ses nouvelles valeurs d'accélération et de direction, tandis que le véhicule ciblé calcule en premier lieu une trajectoire sûre pour lui-même avant de prendre en compte la négociation. Le pseudo code de l'agorithme se trouve en Annexe A.

Implémentation des infrastructures

Le terminal routier dans notre décision basée sur l'infrastructure effectue les actions suivantes à une fréquence donnée : Premièrement, il calcule les trajectoires prévues et souhaitées pour tous les véhicules, en utilisant les informations reçues à partir des diffusions. Ensuite, il identifie les conflits entre véhicules, c'est-à-dire les situations dans lesquelles la trajectoire prévue d'un véhicule est en conflit avec la trajectoire souhaitée d'un autre. Puis, il décide des actions à appliquer à chaque véhicule afin de maximiser la somme des accélérations des véhicules. Enfin, le terminal envoie à chaque véhicule choisi les ordres correspondants. L'impact sur l'algorithme 3 est qu'au lieu d'essayer d'être le plus près possible de l'accélération maximale et de la direction correspondant à la navigation du véhicule, il essaie maintenant d'être aussi proche que possible des valeurs d'accélération et de direction envoyées par le terminal. Le pseudo code de l'algorithme se trouve en Annexe B.

Scénario de test

Notre objectif est de comparer les politiques de décision mentionnées ci-dessus, à savoir :

— la variante de référence de la décision sans négociation ni infrastructure (et sans injection faute),

— la variante utilisant la négociation ;

— la variante utilisant une infrastructure routière intelligente.

Nous allons utiliser pour cela le scénario 2 (cf. Figure 3.5). Dans ce scénario, le véhicule A est initialement sur la voie de droite à la position 0 m avec une vitesse de 30 m/s, le véhicule B est sur la voie de gauche à la position 30 m avec une vitesse de 15 m/s et le véhicule C est sur la voie d'insertion à la position 40 m avec une vitesse de 20 m/s. Ils ont tous pour objectif d'être sur la voie de droite à la fin de la portion de route.

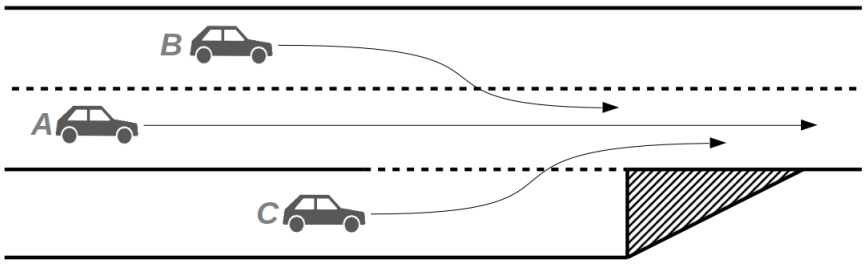

FIGURE 3.5 – Position initiale et trajectoires possibles des CAVs dans le scénario 2.

On considère que les communications du terminal vers les autres agents prennent entre 50 ms et 100 ms et les communications des véhicules vers les autres agents (y compris le terminal) prennent entre 30 ms et 40 ms. La décision du terminal a une fréquence d'activation de 2 Hz et son horloge est initialisée à 0 ms. La décision de chaque véhicule a une fréquence d'activation de 10 Hz. Nous considérons également deux variantes d'initialisation des valeurs d'horloges. Dans la variante d'initialisation 1, les horloges des agents A, B et C sont initialisées respectivement à 70 ms, 30 ms et 0 ms, tandis que dans la variante d'initialisation 2, les horloges des agents A, B et C sont initialisées respectivement à 0 ms, 30 ms et 70 ms.

La Table 3.4 montre les résultats obtenus pour le Scénario 2. Considérons d'abord la variante d'initialisation 1. La prise de décision de base semble conduire à un comportement déterministe, dans lequel C réussit à atteindre la voie de droite devant le véhicule B tandis que A doit freiner pour éviter une collision avec le véhicule B. Cela peut être confirmé en regardant le plus petit TTC entre A et B, qui est inférieur à 1 seconde.

En ajoutant l'aspect de négociation, on peut observer un changement de comportement, lequel n'est à présent plus déterministe. Bien que les temps de trajet pour B et C n'aient pas

Scénario		Init. variant 1			Init. variant 2		
		Cas de référence	Negociation	Infrastructure	Cas de référence	Negociation	Infrastructure
Ordre d'arrivée	A vs B	B	$A\|B$	A	B	A	A
	A vs C	C	$A\|C$	A	C	$A\|C$	A
	B vs C	C	C	C	C	C	C
Temps de parcours	A	$(14.6, 14.6)$	$(13.0, 15.9)$	$(13.2, 13.3)$	$(14.6, 14.6)$	$(13.0, 13.6)$	$(13.2, 13.4)$
	B	$(14.4, 14.4)$	$(14.4, 14.4)$	$(14.4, 14.4)$	$(14.4, 14.4)$	$(14.4, 14.4)$	$(14.4, 14.4)$
	C	$(13.2, 13.2)$	$(13.2, 13.2)$	$(13.8, 13.8)$	$(13.2, 13.2)$	$(13.2, 13.2)$	$(13.8, 13.8)$
Plus petit TTC	A and B	$(0.90, 0.90)$	$(\mathbf{0.00}, 1.56)$	$(1.40, 1.44)$	$(0.95, 0.95)$	$(0.60, \infty)$	$(1.36, 1.41)$
	A and C	(∞, ∞)	$(1.14, \infty)$	(∞, ∞)	(∞, ∞)	$(1.14, \infty)$	(∞, ∞)
	B and C	(∞, ∞)	(∞, ∞)	(∞, ∞)	(∞, ∞)	(∞, ∞)	(∞, ∞)
Exploration complète de l'espace d'états [s]		5	379	27	3	299	36

TABLE 3.4 – Comparaison de l'ordre d'arrivée, du temps de trajet et du TTC pour la décision de base, la décision avec négociation et la décision avec infrastructure pour les deux variantes d'initialisation du Scénario 2.

changé et soient toujours identiques (une valeur unique) pour toutes les exécutions possibles, ce n'est plus le cas pour le véhicule A, où on peut observer un temps de trajet minimal plus court que pour les autres véhicules tandis que son temps de trajet maximal a augmenté. Les ordres d'arrivée nous indiquent que le véhicule A peut parfois arriver avant B, parfois non. Il en va de même pour la comparaison entre A et C. Ces informations ne sont pas suffisantes pour obtenir une image parfaite des scénarios possibles. Comme C arrive toujours avant B, on peut être sûr que les ordres d'arrivée, où A arrive en premier ou en dernier, sont possibles. Cependant, pour savoir si A peut arriver entre B et C, il faut vérifier la propriété suivante : $\mathit{before}(C, A) \rightarrow \mathit{before}(B, A)$, ce qui signifie [3] que lorsque A n'est pas le premier, il doit être dernier, et donc un cas où il arrive entre les deux les véhicules n'est pas possible. Ici, la propriété ci-dessus est fausse et l'exploration de l'espace d'états permet de trouver un chemin menant à un état où le véhicule A arrive entre C et B.

En ce qui concerne la sécurité, on constate que le plus petit TTC entre A et B peut être nul, ce qui signifie qu'il y a des scénarios où une collision est possibles entre ces deux véhicules. Le gain potentiel en temps de déplacement global se paye donc par un manque de sécurité.

Enfin, lorsque la prise de décision basée sur l'infrastructure est utilisée, on constate que le véhicule A arrive toujours en premier et que le véhicule C parvient à atteindre la voie de droite avant le véhicule B, comme auparavant. La comparaison des temps de trajet nous montre qu'avec l'infrastructure, l'efficacité n'est pas aussi bonne qu'avec la négociation dans le meilleur des cas (les véhicules A et C étant plus lents), mais le non-déterminisme n'est pas significatif (contrairement à la négociation lorsqu'il existe risque de collision) et le plus petit TTC est encore meilleur que pour la prise de décision de base.

3. L'opérateur $\rightarrow$, appelé *leads to*, pris en charge par le model checker UPPAAL est équivalent à $AG\ (\neg\ \mathit{before}(C, A) \vee AF\ \mathit{before}(B, A))$.

En examinant les résultats obtenus avec la variante d'initialisation 2, on constate qu'il n'existe pratiquement aucune différence entre la prise de décision de référence et celle basées sur l'infrastructure. Cependant, dans ce scénario, la négociation ne conduit plus à une collision. Le véhicule B arrive toujours en dernier, les véhicules A et C en premier ou en second. Le plus petit TTC est plus petit que dans le cas par de base mais nous gagnons en efficacité globale.

Compte tenu du temps d'exploration de l'espace d'états, on peut voir que le non-déterminisme induit par la négociation dans cet exemple peut entraîner une augmentation considérable du temps nécessaire (environ cent fois plus longtemps) par rapport à notre cas de référence. Notez que cela n'est pas dû aux calculs liés au processus de négociation lui-même mais au non-déterminisme. Enfin, la prise de décision basée sur l'infrastructure, en dépit d'un comportement déterministe, prend plus de temps que le cas de référence pour l'exploration de l'espace d'états (environ 10 fois plus longtemps). Cela est dû au fait que l'infrastructure routière est mise en œuvre comme un agent supplémentaire du système, et que cela impacte la taille de l'espace d'états. Comme on peut le constater, le temps nécessaire pour effectuer la vérification est toutefois maintenu de l'ordre de quelques minutes. grâce à nos choix de modélisation et à la possibilité de régler la précision pour contrôler l'explosion du nombre d'états.

3.3.4 Limitations et gestion des paramètres

À travers plusieurs autres expériences, nous avons également évalué l'impact des principaux paramètres sur le temps de calcul avec UPPAAL. Pour étudier cet impact, nous avons mesuré le temps d'exploration complet en faisant varier plusieurs scénarios avec des comportements non déterministes. Les résultats globaux sont donnés dans la Table 3.5. Les paramètres environnementaux, tels que la longueur de la portion de route, le nombre de valeurs possibles des variables utilisées pour les trajectoires temporelles ou pour représenter les comportements de véhicules, montrent un impact linéaire sur le temps de calcul, même avec des scénarios complexes. Ceci est également vrai pour la perte de précision maximale normalisée Norm_x.

Par exemple, pour un scénario avec une durée d'exploration complète de plus de 20 minutes sans approximation [4], l'utilisation d'une valeur Norm_x = 1 m/s permet de réduire ce temps à environ 6 minutes. Ces résultats indiquent que nous pouvons diminuer le temps de calcul de manière significative avec un coût limité sur la précision de l'environnement modélisé.

En revanche, nous sommes limités vis-à-vis du degré de réalisme, en particulier pour des scénarios impliquant beaucoup de non-déterminisme, où les variations de la période de mise à jour S et de la fréquence de décision montrent un impact exponentiel sur le temps de calcul. En particulier, comme la fréquence induite par la période de mise à jour doit être inférieure

4. c'est à dire, quand $\mathsf{Norm}_x = \mathsf{Gran}_v/2$: dans cet exemple $\mathsf{Norm}_x = 0.05$ m/s.

ou égale à toute autre fréquence du système, l'utilisation d'une fréquence de décision donnée implique une mise à l'échelle en conséquence de la période de mise à jour. Par conséquent, l'augmentation de la fréquence de prise de décision des véhicules dans un système peut entraîner une explosion de l'espace d'états. Cependant, il est toujours possible de monter jusqu'à 20 Hz avec un temps de calcul raisonnable (quelques heures) pour des scénarios complexes comportant beaucoup de non déterminisme, ce qui, comme indiqué dans la section 3.1.1, est suffisant pour les modules de décision mentionné dans la littérature. Pour des scénarios plus simples, cependant, une fréquence de décision plus élevée peut être utilisée.

	Norm_x[m/s]			Gran_a[m/s^2]			L[m]		Plage des délais		S[s]			$freq_i\ i \in [1,n]$[Hz]	
Value	0.5	0.1	0.05	0.5	0.2	0.1	750	1000	100	500	0.05	0.02	0.01	20	33.3
Time [s]	11	11	11	16	17	52	15	18	14	31	16	58	90	45	166

	Norm_x[m/s]			Gran_a[m/s^2]		S[s]		$freq_i\ i \in [1,n]$[Hz]
Value	0.5	0.1	0.05	0.5	0.2	0.05	0.03	20
Time [m]	11.1	15.6	20.6	9.1	18.2	19.0	113.6	500.3

TABLE 3.5 – En haut : Durée d'exploration complète (en secondes) du Scénario 1 avec des paramètres modifiés. À titre de comparaison, l'exploration complète du cas de référence prend 11 secondes. En bas : Durée d'exploration complète (en minutes) de la variante d'initialisation 1 du Scénario 2 à l'aide de la négociation, avec des paramètres modifiés. À des fins de comparaison, l'exploration complète du cas de référence prend 6,3 minutes.

Enfin, il faut noter que le nombre de véhicules n'impacte pas forcément le temps de calcul. C'est bien davantage le non déterminisme résultant des évolutions possibles du système, que le nombre de variables, qui peut ralentir ce dernier. Ceci est illustré dans la Table 3.6, où des véhicules ont été ajoutés aux cas précédemment étudiés, avec diverses valeurs de position et de vitesse choisies aléatoirement. Les temps d'exploration les plus faibles indiquent que la taille croissante de la structure de données semble avoir un impact linéaire sur le système. D'autre part, on peut voir que l'espace d'états généré peut être radicalement différent entre les systèmes avec le même nombre de véhicules mais avec des paramètres initiaux différents.

On notera ainsi que le nombre raisonnable de véhicules pouvant être présent dans le système dépend donc fortement du scénario étudié. En effet, l'ajout de véhicules peut parfois réduire le non-déterminisme du système et parfois l'augmenter. Ce résultat en particulier laisse à penser que VERIFCAR pourrait difficilement supporter des comportement erratiques comme un algorithme de décision avec choix aléatoires qu'il faudrait explorer exhaustivement. En revanche, le cadre logiciel convient bien à la vérification des comportements de véhicules autonomes communicants caractérisés par des protocoles de décision bien définis.

	Scenario 1					Scenario 2 variante 1 (négo.)	
Nombre de véhicules ajoutés	1	2	3	4	5	1	2
Temps le plus court [s]	13	24	30	40	52	347	505
Temps le plus long [s]	17	50	151	142	167	817	1798

TABLE 3.6 – Temps d'exploration complet pour le scénario 1 et la variante 1 du scénario 2 où la négociation est utilisée, tandis que des véhicules sont ajoutés avec des valeurs de position et de vitesse initiales aléatoires. Plusieurs initialisations aléatoires ont effectuées pour chaque cas et les temps d'exploration observés les plus petits et les plus longs sont indiqués.

Pour étudier le comportement résultant de l'utilisation d'un algorithme de décision, il est nécessaire de l'implémenter dans le langage d'UPPAAL, qui ne prend en charge que les variables discrètes. Par conséquent, il est recommandé de mettre les variables à l'échelle et d'arrondir les divisions pour éviter ou au moins réduire les accumulations d'erreur. De plus, s'il existe des variables nécessaires à la prise de décision ou au protocole de communication, elles doivent être ajoutés à la structure de données. Le nombre de valeurs possibles doit être défini et il est recommandé de les garder aussi petites que possible. La principale limitation vient des paramètres liés au temps, principalement la fréquence de la prise de décision, qui peut difficilement dépasser 20 Hz. Cependant, comme mentionné précédemment en section 3.1.1, cette limitation ne doit pas poser de problème pour modéliser la plupart des protocoles de décision décrits dans la littérature. En ce qui concerne le calibrage des paramètres, il convient de trouver un équilibre entre la précision sur la position longitudinale (définie par le paramètre de perte de précision maximale normalisé, Norm_x) et la fréquence de mises à jour du système. Le premier permet de réduire les temps de calcul avec un coût en précision et le second de gagner en précision sur la surveillance du système au détriment de l'augmentation des temps de calcul. Afin d'estimer une perte de précision acceptable, on peut d'abord procéder par simulation, ce qui peut également être fait avec UPPAAL. On peut étudier un indicateur donné avec des exécutions utilisant différentes valeurs de Norm_x, puis les comparer au résultat obtenu avec le Norm_x qui garantit qu'il n'y ait pas de perte d'information (avec $p = 1$ si nous nous référons à la section 3.1.2). L'utilisateur doit alors choisir le plus grand Norm_x pour lequel l'erreur observée est toujours acceptable (cela peut nécessiter une expertise sur le domaine du cas d'étude).

Enfin, nous résumons ici toutes les hypothèses et abstractions présentes dans VERIFCAR :

— Les capteurs sont parfaits (précis et immédiats) ;

— La décision est prise à une fréquence constante fixée ;

— La latence de communication se produit dans un intervalle de temps défini et fini ;

— La fréquence de décision pour un véhicule donné est supérieure ou égale à la latence de

communication maximale pour le même véhicule ;

— Les lois physiques sont simplifiées (la rotation, le frottement, l'inertie sont ignorés) ;

— L'environnement est plat (deux dimensions seulement sont considérées).

3.4 Résumé du chapitre

Le chapitre présente le cadre logiciel VERIFCAR, un environnement de travail comprenant une modélisation du système et une méthodologie de vérification de modèle de CAVs. Il est fondé sur les automates temporisés de l'outil UPPAAL, qui permet la vérification de propriétés exprimées dans un sous-ensemble de CTL.

L'architecture du modèle représente les actions possibles de chaque agent du système (des véhicules communicants ou des infrastructures routières communicantes) par un automate temporisé. Chaque action peut avoir lieu dans un intervalle de temps modélisant le non-déterminisme induit par la latence et les délais de communication. L'état des agents est conservé dans une structure de données faisant partie du modèle. Un automate additionnel met à jour les valeurs de cette structure de données selon une fréquence définie. Une discrétisation des différentes grandeurs physiques étant nécessaire dans UPPAAL, la granularité de chaque variable représentant l'état du véhicule sur la route est mise à l'échelle automatiquement en fonction des paramètres du modèle. Cela permet d'éviter les approximations sur les calculs et le cumuls d'erreurs qui en découlent mais peut conduire à une explosion du nombre d'états lorsque la granularité est inutilement petite. La précision souhaitée peut alors être fixée comme paramètre et permet de réduire l'espace d'états.

La méthodologie de vérification se base sur des indicateurs permettant d'évaluer le comportement des véhicules en termes de sécurité, d'efficacité ou de confort. En particulier, l'aspect de sécurité est considéré à travers le temps avant collision (TTC), qui est ici adapté à un espace à deux dimensions et peut être calculé en temps constant. Comme les propriétés sont vérifiées sur un modèle non-déterministe, un ensemble de valeurs possibles doit être considéré pour chaque indicateur. Cela conduit parfois à utiliser l'outil de vérification de modèle itérativement pour identifier par dichotomie les extrema des indicateurs.

Finalement, VERIFCAR est utilisé sur divers exemples d'étude du comportement de véhicules autonomes communicants. On constate que les paramètres environnementaux, tels que la longueur de la portion de route, le nombre de valeurs possibles des variables utilisées pour les trajectoires temporelles ou pour représenter les comportements de véhicules, ont un impact linéaire sur le temps de calcul, même avec des scénarios complexes. À l'inverse, l'augmentation de la fréquence de mise à jour de l'environnement à un impact exponentiel. Cela limite l'utilisa-

tion de VERIFCAR, la fréquence de prise de décision des véhicules ne pouvant être supérieure à celle de mise à jour de l'environnement. Néanmoins, il est possible d'aller jusqu'à 20 Hz avec un temps de calcul de quelques heures pour des scénarios complexes comportant beaucoup de non déterminisme, ce qui suffit pour décrire les modules de décision utilisés habituellement dans le domaine des véhicules autonomes communicants.

Chapitre 4

Exploration de MAPTs

Sommaire

4.1 Le formalisme des MAPTs 69
4.1.1 Syntaxe et sémantique des G-MAPTs 69
4.1.2 Traduction en réseaux de Petri de haut niveau 77
4.1.3 Accélération 79
4.2 Algorithmes d'exploration 86
4.2.1 Espace d'états en couches 87
4.2.2 Exploration basée sur les variables fortes et faibles 96
4.2.3 Exploration «à la volée» d'un MAPT 99
4.3 Comparaison de performances 100
4.3.1 Efficacité de la dynamique accélérée. 101
4.3.2 Efficacité de l'exploration par couches. 102
4.3.3 Heuristiques 103
4.3.4 Comparaison avec VERIFCAR 105
4.4 Résumé du chapitre 107

Le cadre logiciel VERIFCAR, qui a été présenté dans le chapitre précédent, permet une analyse exhaustive de situations critiques en termes de sécurité, d'efficacité et de robustesse, en mettant l'accent en particulier sur l'impact des latences, des délais de communication et des défaillances sur le comportement des CAVs. Cependant, il est limité en termes d'expressivité et ne traite que des variables discrètes, ce qui n'est pas toujours pratique et peut conduire à des calculs imprécis.

Il s'avère que les espaces d'états dans les applications étudiées avec VERIFCAR sont généralement très grands mais prennent la forme sémantique d'un graphe orienté acyclique (DAG).

De plus, entre deux réinitialisations d'horloge, chaque agent a syntaxiquement la forme d'un DAG. Nous exploitons ces particularités afin de construire un environnement de vérification dédié aux problèmes d'accessibilité. Concrètement, le graphe est exploré de manière dynamique (en vérifiant les propriétés de la logique temporelle directement lorsque nous explorons des états) afin d'éviter de construire l'espace d'états complet, et donc de ne pas perdre de temps et d'espace mémoire en stockant et en comparant tous les états précédemment atteints. Cela permet également d'ajuster l'algorithme de vérification avec des méthodes heuristiques permettant de choisir le chemin à explorer en priorité, ce qui accélère considérablement le temps de calcul si l'état recherché existe. Pour ce faire, nos algorithmes explorent le graphe en profondeur d'abord, car les algorithmes de largeur d'abord ne peuvent explorer les chemins librement et sont limités à l'exploration complète de tous les états à un certain niveau de profondeur avant l'exploration du niveau suivant.

Pour les systèmes présentant un haut niveau de concurrence entre les actions, tels que les systèmes de CAVs, la plupart du non-déterminisme résulte de la possibilité que plusieurs actions de différents agents soient disponibles à partir d'un état donné, pouvant se produire dans des ordres différents mais conduisant éventuellement au même état. Cela correspond dans l'espace des états à ce qu'on appellera des **diamants**. L'exploration en largeur d'abord permet de comparer les états à une profondeur donnée et donc d'éliminer les doublons, ce qui est un moyen efficace de détecter de tels diamants. D'un autre côté, l'exploration en profondeur d'un tel espace d'états avec des diamants conduit à examiner éventuellement plusieurs fois les mêmes états ou chemins, ce qui n'est pas efficace. Dans ce contexte, notre objectif est de détecter et de fusionner des états identiques provenant de diamants tout en continuant d'explorer l'espace d'états principalement en profondeur. Cette détection de diamants consiste en une exploration préalable en largeur dans une certaine couche de l'espace d'états, chaque couche recouvrant les états de plusieurs niveaux de profondeur ayant des caractéristiques communes. Il se trouve que de telles couches peuvent être observées dans l'espace d'états de systèmes de CAVs. Cela permet une exploration en profondeur d'une couche à l'autre, tout en réduisant considérablement les chances d'explorer plusieurs fois les mêmes états. La classe de modèles sur laquelle ce type d'algorithmes peut être appliquée sera nommée *Multi-Agent with timed Periodic Tasks* (MAPTs), systèmes multi-agent à tâches périodiques, en français, et constitue un sous-ensemble des réseaux d'automates temporisés.

Pour implémenter de tels algorithmes, nous utilisons ZINC [7], un compilateur qui génère, à partir d'un modèle formel, une bibliothèque de fonctions permettant d'explorer facilement l'espace d'états.Le formalisme utilisé par ZINC est celui des réseaux de Petri de haut niveau, nous avons donc défini une transformation des G-MAPT vers ce formalisme. Nous utilisons ces fonctions pour explorer «à la volée» l'espace d'états avec des algorithmes conçus pour nos

besoins. Cela permet notamment d'appliquer des heuristiques conduisant à des temps de calcul réduits, ainsi qu'une meilleure expressivité sur les requêtes de logique temporelle que UPPAAL, notamment en incluant des requêtes imbriquées. Un autre avantage comparatif à UPPAAL est que nous ne sommes pas limités aux calculs de nombres entiers et que nous pouvons utiliser des variables réelles (ou rationnelles), limitant ainsi la perte d'informations. Pour utiliser ZINC, on émule le temps avec des variables discrètes. Nous le faisons d'une manière qui préserve le comportement du système : en utilisant le modèle avec les mêmes variables discrètes qu'avec UPPAAL, nous obtenons des résultats identiques.

Dans ce chapitre, après avoir défini dans le détail le formalisme des MAPTs, nous présentons la structure en couches et les algorithmes en tirant parti. Enfin, nous proposons des heuristiques et les utilisons dans des expériences mettant en évidence les avantages de notre approche en termes d'expressivité, de niveau d'abstraction et de temps de calcul.

4.1 Le formalisme des MAPTs

Nous commençons par une définition formelle des G-MAPTs, une classe générale des MAPTs, étudions ses propriétés et donnons une traduction pour les réseaux de Petri de haut niveau. Ensuite, nous introduisons nos modèles de MAPT, en limitant légèrement ceux de G-MAPT (on contraint l'acyclicité et la *liveness* du modèle) afin de permettre une procédure d'accélération et l'utilisation d'algorithmes d'exploration «à la volée».

4.1.1 Syntaxe et sémantique des G-MAPTs

Un G-MAPT est un modèle composé de plusieurs agents pouvant interagir via des variables partagées. Dans le formalisme, on ne considérera qu'une seule variable, qui représente l'ensemble des combinaisons de valeurs possibles de toute les variables.Chaque agent est associé à une horloge et effectue des actions se produisant dans des intervalles de temps donnés. Il n'y a pas de compétition entre les agents dans le sens qu'aucun agent n'aura à attendre l'action d'un autre pour accomplir sa propre action. Cependant, il peut y avoir du non-déterminisme lorsque plusieurs actions sont disponibles en même temps, ou qu'une action est possible en même temps qu'une augmentation du temps.

Un G-MAPT est un tuple $(\mathcal{V}, F, A, Init)$ où :

— $\mathcal{V}$ est un ensemble fini de valeurs ;

— F est un ensemble fini de transformation de variable, c'est-à-dire des fonctions calculables de $\mathcal{V}$ dans $\mathcal{V}$;

— A est un ensemble de n agents tels que $\forall i \in [1, n]$, l'agent $A_i \stackrel{\text{df}}{=} (L_i, C_i, T_i, E_i)$ avec :

  — L_i est un ensemble de localités notées sous forme d'une liste $L_i \stackrel{\text{df}}{=} (l_i^1, \ldots, l_i^{m_i})$ avec $m_i > 0$, telle que $\forall i \neq j$, $L_i \cap L_j = \emptyset$;

  — C_i est l'horloge unique de l'agent A_i, avec des valeurs dans $\mathbb{N}$;

  — T_i est un ensemble fini de transitions, formant un graphe acyclique orienté entre les localités, avec une localité initiale unique l_i^1 et une localité finale unique $l_i^{m_i}$, chaque transition étant de la forme (l, f, I, l') où $l, l' \in L_i$ sont les localités d'origine et de destination, $f \in F$ est une fonction et $I \stackrel{\text{df}}{=} [a, b]$ est un intervalle avec $a, b \in \mathbb{N}$ et $a \leq b$;

  — $E_i \in \mathbb{N} \setminus \{0\}$ est la période de réinitialisation de l'agent A_i.

— Init est un triplet $((l_1, \cdots, l_n), (\mathsf{init}_1, \cdots, \mathsf{init}_n), \mathsf{init}_V)$ où $\forall i \in [1, n]$, $l_i \in L_i$, $\mathsf{init}_i \in \mathbb{N}$ et $\mathsf{init}_V \in \mathcal{V}$.

Pour chaque agent A_i et chaque localité $l \in L_i$, nous notons $l^\bullet = \{(l, f, I, l') \in T_i\}$ l'ensemble des transitions sortantes de l et par ${}^\bullet l = \{(l', f, I, l) \in T_i\}$ l'ensemble des transitions menant à l. Notez que, selon les hypothèses, ${}^\bullet l_i^1 = \emptyset$ et $(l_i^{m_i})^\bullet = \emptyset$. De plus, lorsque $i \neq j$, puisque $L_i \cap L_j = \emptyset$, $T_i \cap T_j = \emptyset$ aussi, de sorte que chaque transition appartient à un seul agent, évitant ainsi toute confusion dans le modèle.

Un exemple simple de G-MAPT M avec deux agents non déterministes est représenté en Exemple 1.

Dans la sémantique, pour chaque agent A_i, on émule une transition de $l_i^{m_i}$ vers l_i^1 qui réinitialise l'horloge C_i toutes les E_i unités de temps. Ainsi, chaque agent effectue un cycle sur une période déterminée, lequel se termine nécessairement par une réinitialisation de C_i. Il peut cependant y avoir plusieurs chemins possibles pour ce cycle, puisqu'une localité peut avoir plusieurs transitions sortantes, et donc il peut exister plusieurs chemins allant de l_i^1 à $l_i^{m_i}$.

Le comportement du système est défini comme un système de transitions où Init est l'état initial. L'état d'un G-MAPT composé de n agents tels que décrit ci-dessus est un triplet noté $s = (\vec{l}, \vec{c}, v)$ avec $\vec{l} = (l_1, \cdots, l_n)$ où $l_i \in L_i$ est la localité actuelle de l'agent A_i, $\vec{c} = (c_1, \cdots, c_n)$ où $c_i \in \mathbb{N}$ est la valeur de l'horloge C_i, et $v \in \mathcal{V}$ est la valeur de la variable v. Il existe trois différents types de changement d'état : le franchissement de transition, la réinitialisation d'horloge et l'augmentation du temps.

— Une transition $(l, f, [a, b], l') \in T_i$ peut être franchie si $l_i = l$ et $a \leq c_i \leq b$. Dans le nouvel état, $l_i \leftarrow l'$ et $v \leftarrow f(v)$.

— Une horloge C_i peut être réinitialisée si $l_i = l_i^{m_i}$ et $c_i = E_i$. Dans le nouvel état, $c_i \leftarrow 0$ et $l_i \leftarrow l_i^1$.

— Le temps peut augmenter si $\forall i \in [1, n]$, soit il existe au moins une transition $(l, f, [a, b], l') \in T_i$ avec $l_i = l$ et $c_i < b$, soit $l_i = l_i^{m_i}$ et $c_i < E_i$. Une augmentation du temps signifie que $\forall i \in [1, n]$, $c_i \leftarrow c_i + 1$.

On peut remarquer que seule la variable v est un élément global du système, tous les autres éléments sont locaux à un agent. Il s'agit d'une variable unique mais qui peut être considérée sous la forme d'un vecteur, tel qu'un agent puisse modifier plusieurs composants de ce vecteur au moyen des fonctions de F utilisées dans ses transitions. Cela permet donc la représentation de plusieurs variables globales. Les valeurs de v ne sont pas limitées au domaine des nombres entiers, mais seul un ensemble comptable de valeurs peut être atteintes : celles qui peuvent être obtenues à partir de init_v par une application récursive de fonctions à partir de F, la variable n'étant pas modifiée par les réinitialisations ni par les augmentations de temps.

Un chemin (ou exécution) dans un G-MAPT est une séquence ordonnée et non vide d'états contenant comme premier élément l'état initial du système. Le $i + 1^{\text{ème}}$ élément du chemin est obtenu par application d'une des règles de changement d'état sur le $i^{\text{ème}}$ élément (son prédécesseur).

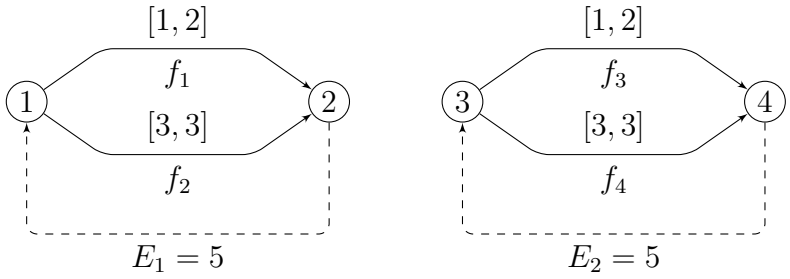

FIGURE 4.1 – Représentation visuelle du G-MAPT de l'Exemple 1. Les arcs en pointillés représentent les réinitialisations d'horloge.

Example 1. *Soit* $M = (\mathcal{V}, F, A, \mathsf{Init})$ *où :*

— $\mathcal{V} = \mathbb{R} \times \mathbb{N}$*;*

— $F = \{f_1, f_2, f_3, f_4\}$ *avec*
$f_1(x, y) \to (2x, y + 1)$ $f_2(x, y) \to (x + 1.3, y + 1)$
$f_3(x, y) \to (\frac{x}{2}, y)$ $f_4(x, y) \to (2x, y)$

— $A = \{(L_1, C_1, T_1, E_1), (L_2, C_2, T_2, E_2)\}$ *avec*
$L_1 = \{1, 2\}$ $T_1 = \{(1, f_1, [1, 2], 2), (l, f_2, [3, 3], 2)\}$ $E_1 = 5$
$L_2 = \{3, 4\}$ $T_2 = \{(3, f_3, [1, 2], 4), (3, f_4, [3, 3], 4)\}$ $E_2 = 5$

— $\mathsf{Init} = ((1, 3), (0, 0), (0.5, 0))$.

Une représentation visuelle de M est donnée en Fig. 4.1 tandis que le fragment initial de sa dynamique est donné en haut à gauche dans la Fig. 4.4. On notera que seuls les franchissements de transition et les augmentations de temps sont représentés dans la Fig. 4.4 et que les valeurs de la variable v dans les états y sont omis. ◇

Dans un système dynamique, la **persistance** est une propriété qui indique que, si deux changements d'état sont possibles à partir d'un certain état, alors aucun de ces changements n'empêche l'autre à l'état suivant, et leur exécution dans n'importe quel ordre conduit au même état final, formant un ainsi un diamant. Dans les systèmes G-MAPT, nous avons une sorte de persistance limitée à différents agents.

Proposition 1. *Dans un* G-MAPT*, si $i \neq j$ et $s = (\vec{l}, \vec{c}, v)$ est un état quelconque, on a :*

1. *si s permet deux transitions $t_1 = (l_1, f_1, I_1, l'_1) \in T_i$ et $t_2 = (l_2, f_2, I_2, l'_2) \in T_j$, menant respectivement aux états s_1 et s_2, alors s_1 permet t_2 conduisant à un état s_3 et s_2 permet t_1 conduisant à un état s_4 ; de plus, $s_3 = s_4$ si est seulement si $f_1 \circ f_2(v) = f_2 \circ f_1(v)$, c'est-à-dire si f_1 et f_2 sont commutatives sur v ;*
2. *si s permet une transition $t = (l, f, I, l') \in T_i$ et la réinitialisation de A_j, menant respectivement aux états s_1 et s_2, alors s_1 permet la réinitialisation de A_j conduisant à un état s_3 et s_2 permet t conduisant au même état s_3 ;*
3. *si s permet la réinitialisation de A_i et la réinitialisation de A_j, menant respectivement aux états s_1 et s_2, alors s_1 permet la réinitialisation de A_j conduisant à un état s_3 et s_2 permet la réinitialisation de A_i conduisant au même état s_3.*

Démonstration.

1. La propriété résulte du fait que les transitions ne modifient pas les horloges et qu'une transition d'un agent quelconque modifie uniquement la localité de cette agent, ainsi que la variable commune v ; dans s_3, la variable devient $f_2 \circ f_1(v)$, tandis que dans s_4, la variable devient $f_1 \circ f_2(v)$. On notera que si $i = j$, s_1 ne permet pas la transition t_2 puisque, par l'hypothèse de l'acyclicité, la localité de A_i est modifiée et ne permet plus t_2 (et symétriquement) ;
2. la propriété résulte du fait qu'une transition ne modifie aucune valeur d'horloge, et la réinitialisation de A_j ne modifie que la localité et l'horloge de ce dernier ; la variable commune v aura la valeur $f(v)$ à la fois après l'exécution de la transition suivie de la réinitialisation et après la réinitialisation suivie de la transition.
3. la propriété résulte du fait qu'une réinitialisation d'un agent modifie uniquement la localité et l'horloge de ce dernier.

□

Contraintes Nous définissons dans cette sous-section deux types de contraintes, qui sont motivées par notre domaine d'application cible et seront utilisées pour obtenir des propriétés intéressantes.

Un G-MAPT est appelé **fortement vivant** s'il satisfait la contrainte suivante :

Contrainte 1.

1. *Si la localité initiale d'un agent A_i est terminale ($l_i = l_i^{m_i}$), alors la valeur initiale de l'horloge C_i satisfait* $\mathsf{init}_i \leq E_i$ *;*
2. *sinon (quand $l_i \neq l_i^{m_i}$), on a* $\mathsf{init}_i \leq \max\{b \mid (l_i, f, [a, b], l') \in l_i^\bullet\}$ *;*
3. *de plus, pour chaque agent A_i, si $l \in L_i \setminus \{l_i^1, l_i^{m_i}\}$, alors* $\max\{b \mid (l', f, [a, b], l) \in {}^\bullet l\} \leq \min\{b' \mid (l, f', [a', b'], l'') \in l^\bullet\}$ *;*
4. *et on a* $\max\{b \mid (l', f, [a, b], l_i^{m_i}) \in {}^\bullet l_i^{m_i}\} \leq E_i$.

Les deux premières contraintes garantissent que, lorsque le système est au début de son exécution, que ce soit dans la localité terminale ou non d'un autre agent quelconque, le système n'est pas bloqué et nous aurons la possibilité d'exécuter une action. La contrainte suivante garantit que chaque fois que l'on entre dans une localité non terminale, toute transition sortante de cette localité aura la possibilité de se produire, éventuellement après avoir effectuer une augmentation de temps (le cas où nous entrons dans une localité initiale est sans importance, car les réinitialisations mettent l'horloge correspondante à 0). La dernière contrainte capture des caractéristiques similaires dans le cas où nous entrons dans une localité terminale.

En d'autres termes, chaque transition ou réinitialisation dans $l^\bullet$ est permise lors de l'entrée dans l ou le sera ultérieurement (éventuellement après un certain laps de temps pour atteindre la borne inférieure a).

Proposition 2. *Dans un* G-MAPT *satisfaisant la contrainte 1, après toute évolution ω, chaque transition et chaque réinitialisation (ainsi que les augmentations de temps) peuvent être franchies dans le futur.*

Démonstration. On peut d'abord montrer par induction sur la taille de ω que, si $s = (\vec{l}, \vec{c}, v)$ est l'état atteint après l'évolution ω, pour tout agent A_i nous avons $(l_i = l_i^{m_i}) \implies c_i \leq E_i$ et $(l_i \neq l_i^{m_i}) \implies c_i \leq \max\{b \mid (l_i, f, [a, b], l') \in l_i^\bullet\})$, c'est à dire que le même G-MAPT avec un état initial s satisfait également la contrainte 1. La propriété est triviale pour $\omega = \varepsilon$. Si la propriété est satisfaite pour un ω quelconque , elle le restera pour toute extension, par la définition de la sémantique des G-MAPTs et des deux derniers points de la contrainte 1. Il en résulte également que toute évolution ω satisfaisant les propriétés mentionnées peut être étendue.

Il reste à montrer que toute extension peut être effectuée dans un futur proche.

Concernant l'augmentation du temps, on peut observer que, si une augmentation ne pouvait plus se produire, étant donné que l'ensemble des localités de chaque agent a la forme d'un DAG, étendre ω va nécessairement conduire à une réinitialisation. De plus, comme chaque E_i est strictement positif, l'ensemble des réinitialisations finiraient par être effectuées et le système se bloquerait à un moment donné, ce qui est impossible. Par conséquent, nous sommes certains que le temps augmentatera. Etant donné que les augmentation de temps peuvent toujours être exécutés dans le futur, toutes les réinitialisations finiront par être franchies.

Enfin, passons à $t = (l'_i, f, [a, b], l''_i) \in T_i$. De la même manière que pour l'augmentation du temps, il sera finalement possible d'effectuer une réinitialisation r_i, puis de suivre un chemin allant de l_i^1 à l'_i dans le DAG de A_i en effectuant chaque transition tour à tour lorsque le a correspondant est atteint, du fait de la contrainte 1.3. □

La contrainte suivante garantit l'acyclicité de la dynamique du G-MAPT(son espace d'états) de manière syntaxique :

Contrainte 2.

1. *Si $\mathcal{V} \stackrel{\text{df}}{=} W \times X$ et il existe un agent A_i tel que dans tous les chemins de l_i^1 vers $l_i^{m_i}$, il existe une transition $t \stackrel{\text{df}}{=} (l, f, [a, b], l')$ telle que pour tout $(w, x) \in \mathcal{V}$, $f(w, x) = (w', x')$ avec $x < x'$;*
2. *et il n'y a pas de $f \in F$ tel qu'il existe $(w, x) \in \mathcal{V}$, où $f(w, x) = (w', x')$ avec $x > x'$.*

La première contrainte garantit qu'au moins un agent incrémente la partie X de la variable v au moins une fois entre deux réinitialisations consécutives. La seconde assure qu'aucune fonction ne peut décrémenter la partie X de la variable v.

En d'autres termes, la partie X de v avance à chaque cycle de l'agent A_i, ce qui entraîne une absence de cycles dans l'espace d'états du G-MAPT.

Proposition 3. *Un* G-MAPT *satisfaisant la contrainte 2 est acyclique.*

Démonstration. S'il y a un cycle, cela signifie qu'il existe un chemin depuis l'état initial avec au moins deux états différents dans le chemin $s_1 = (\vec{l}, \vec{c}, v)$ et $s_2 = (\vec{l'}, \vec{c'}, v')$, qui sont en fait identiques. Puisque les localités de chaque agent forment un DAG statique déterminé par ses transitions et que chaque E_i est strictement positif, avoir $\vec{l} = \vec{l'}$ et $\vec{c} = \vec{c'}$ ne peut se produire que si tous les agents ont effectué au moins une réinitialisation entre s_1 et s_2. En effet, puisque $\vec{c} = \vec{c'}$, cela implique qu'il n'y a que des transitions dans le cycle, ce qui est en contradiction avec la syntaxe des agents. De plus, étant donné qu'entre deux réinitialisations d'un agent, le temps augmente strictement, $\vec{c} = \vec{c'}$ ne peut se produire que si tous les agents ont effectué une

ou plusieurs réinitialisations. Il est donc suffisant d'observer par la contrainte 2.2 que la variable v ne peut pas diminuer, et par la contrainte 2.1 qu'une transition t garantissant que $v \neq v'$ est franchie nécessairement. □

Definition 1. *Un* MAPT *est un* G-MAPT *satisfaisant les contraintes 1 et 2.*

Un MAPT peut être non déterministe, mais il est fortement vivant et dispose d'un espace d'états DAG. Par exemple, le G-MAPT M de l'Exemple 1 satisfait les deux contraintes (l'acyclicité est satisfaite car y est incrémenté dans tous les cycles de A_1) et est donc en fait un MAPT.

Correspondance avec les automates temporisés Il n'est pas difficile de transformer un G-MAPT en un réseau d'automates temporisés (étendus avec des fonctions) ayant essentiellement le même comportement. Cela nous est utile entre autres pour effectuer des comparaisons avec modèles utilisés sous UPPAAL avec VERIFCAR.

La transformation s'opère de la manière décrite ci-dessous. Les ensembles F et $\mathcal{V}$ peuvent être définis pour les automates temporisés de la même manière que pour les G-MAPTs. Pour chaque agent $A_i = (L_i, C_i, T_i, E_i)$, il existe un automate A_i ayant l'ensemble de localités L_i et une horloge locale C_i, obtenu de la manière suivante.

— $\forall j \in [1, m_i - 1]$, la localité l_i^j est associée à un invariant $c_i \leq B$ où B est la plus grande borne supérieure des intervalles de toutes les transitions sortantes de l_i^j ;

— la localité $l_i^{m_i}$ est associée à un invariant $c_i \leq E_i$;

— $\forall (l, f, [a, b], l') \in T_i$, on a une transition de l vers l' qui appelle la fonction f et est associée à une garde $a \leq c_i \leq b$;

— on a une transition de $l_i^{m_i}$ vers l_i^1 associée à une garde $c_i = E_i$ et qui réinitialise C_i.

En somme, chaque transition dans le réseau d'automates est associée à une garde correspondante à celle de la transition dans le G-MAPT, et on ajoute également à chaque localité un invariant qui contraint le système à effectuer une des transitions sortantes de la localité. Autrement dit, il est impossible de rester infiniment longtemps dans une localité.

Afin de comparer le comportement du G-MAPT à celui de sa transformation en automates temporisés, on utilise une projection discrète de la sémantique d'un réseau d'automates temporisés. Cette projection est le plus petit système de transitions tel que :

— pour tout état $(\vec{l}, \vec{c}, v)$ du réseau d'automates, il existe dans la projection discrète un état $(\vec{l}, \lceil \vec{c} \rceil, v)$, l'état initial étant identique ;

— pour tout arc dans la sémantique du réseau d'automates, qui part d'un état $(\vec{l}, \vec{c}, v)$ vers un état $(\vec{l'}, \vec{c'}, v')$, il existe dans la projection discrète un arc qui part d'un état $(\vec{l}, \lceil \vec{c} \rceil, v)$ vers un état $(\vec{l'}, \lceil \vec{c'} \rceil, v')$.

Proposition 4. *La sémantique d'un* G-MAPT *qui vérifie la contrainte 1 est équivalente à la projection discrète de la sémantique de sa transformation en réseau d'automates temporisés étendus avec des fonctions.*

Démonstration. On peux vérifier que la sémantique du G-MAPT est inclue dans celle du réseau d'automates (et donc dans sa projection) puisque pour chaque intervalle de temps I où la sémantique du G-MAPT admet une action pour toute valeur d'horloge entière dans I, la sémantique du réseau d'automates admet une action pour toute valeur d'horloge réelle dans I.

Pour l'autre sens de l'équivalence, on remarque que dans le cas des augmentations de temps de plusieurs unités de temps, la projection produit à la fois des arcs unitaires et des arcs combinant plusieurs unités de temps. Ces dernières peuvent être supprimés sans perte de généralité vu qu'ils peuvent être obtenus par transitivité à partir des arcs unitaires. Que la projection privé des arcs non unitaires soit inclue dans la sémantique du G-MAPT demande cependant que le comportement avec le temps continu (donc en nombres réels) conduise seulement à des états caractérisés par un vecteur de localités et une valeur de variable qui sont atteignables par la sémantique du G-MAPT. Cela est garanti par le fait que :

— les réinitialisations se produisent toujours à des dates entières ;

— pour toute transition dans un G-MAPT qui vérifie la contrainte 1, le moment où est franchie une transition n'impacte pas l'intervalle de temps pendant lequel les actions futures pourront avoir lieu ;

— toute fonction $f \in F$ ne dépend pas du temps.

□

On notera que cette propriété n'est pas satisfaite si le G-MAPT n'est pas fortement vivant (c'est à dire si la contrainte 1 n'est pas satisfaite). En effet, si l'une des deux premières conditions de la contrainte n'est pas satisfaite, l'état initial du réseau d'automate ne satisfait pas l'invariant d'au moins une des localités initiales du réseau et n'est donc pas valide dans ce formalisme. Et si l'une des deux dernières conditions de la contrainte n'est pas satisfaite, la transition correspondante est bloquée par la sémantique des automates puisque l'invariant d'au moins une localité n'est pas respecté tandis que pour le modèle G-MAPT, le problème de blocage se produit après le déclenchement de la transition.

4.1.2 Traduction en réseaux de Petri de haut niveau

Les algorithmes d'explorations des MAPT ont été implémentés avec ZINC, un outil académique permettant de construire et de parcourir l'espace d'états d'un réseau de Petri de haut niveau avec une sémantique d'entrelacements. En effet, il est possible d'exprimer un G-MAPT en tant que réseau de Petri de haut niveau.

Definition 2. *Étant donné un* G-MAPT $Q = (\mathcal{V}, F, A, Init)$ *avec* $|A| = n$*, sa traduction en réseau de Petri de haut niveau* $N = translate(P) = (S, T, \lambda, M_0)$ *est définie comme suit :*

- $S = \{s_A, s_C, s_V\}$ *les places du réseau avec les types* $\lambda(s_A) \stackrel{\text{df}}{=} L_1 \times \cdots \times L_n$ *où* L_i *est l'ensemble des localités de l'agent* A_i *(son* $j^{ème}$ *élément sera noté* l_i^j*),* $\lambda(s_C) \stackrel{\text{df}}{=} \mathbb{N}^n$ *et* $\lambda(s_V) \stackrel{\text{df}}{=} \mathcal{V}$*; Pour tout jeton* x *de type* $\lambda(s_A)$ *ou* $\lambda(s_C)$*, on note* $x[i]$ *le* $i^{ème}$ *élément de la liste.*
- $T \stackrel{\text{df}}{=} T_{trans} \cup T_{reset} \cup \{t_{time}\}$ *où*
  - T_{trans} *est le plus petit ensemble de transitions tel que, pour chaque agent* $A_i = (L_i, C_i, T_i, E_i)$ *dans* A *et pour chaque transition* $(l, f, [a, b], l') \in T_i$*, il existe une transition* $t \in T_{trans}$ *telle que* $\lambda(s_A, t) \stackrel{\text{df}}{=} x$*,* $\lambda(s_C, t) \stackrel{\text{df}}{=} y$*,* $\lambda(s_V, t) \stackrel{\text{df}}{=} z$*,* $\lambda(t, s_A) \stackrel{\text{df}}{=} x'$ *où* $x'[i] \leftarrow l'$ *et* $\forall j \neq i$*,* $x'[j] \leftarrow x[j]$*,* $\lambda(t, s_C) \stackrel{\text{df}}{=} y$*,* $\lambda(t, s_V) \stackrel{\text{df}}{=} f(z)$ *et* $\lambda(t) \stackrel{\text{df}}{=} (x[i] = l) \wedge (a \leq y[i] \leq b)$*. Ceci est équivalent à l'ensemble des transitions du* G-MAPT*.*
  - T_{reset} *est le plus petit ensemble de transitions tel que, pour chaque agent* $A_i = (L_i, C_i, T_i, E_i)$ *dans* A*, il existe une transition* $t \in T_{reset}$ *telle que* $\lambda(s_A, t) \stackrel{\text{df}}{=} x$*,* $\lambda(s_C, t) \stackrel{\text{df}}{=} y$*,* $\lambda(t, s_A) \stackrel{\text{df}}{=} x'$ *où* $x'[i] \leftarrow l_i^1$ *et* $\forall j \neq i$*,* $x'[j] \leftarrow x[j]$*,* $\lambda(t, s_C) \stackrel{\text{df}}{=} y'$ *où* $y'[i] \leftarrow 0$ *et* $\forall j \neq i$*,* $y'[j] \leftarrow y[j]$*, et* $\lambda(t) \stackrel{\text{df}}{=} (x[i] = l_i^{m_i}) \wedge (y[i] = E_i)$ *où* $m_i \stackrel{\text{df}}{=} |L_i|$*. Ceci est équivalent à l'ensemble des réinitialisations d'horloge du* G-MAPT*.*
  - $\lambda(s_A, t_{time}) \stackrel{\text{df}}{=} x$*,* $\lambda(s_C, t_{time}) \stackrel{\text{df}}{=} y$*,* $\lambda(t_{time}, s_A) \stackrel{\text{df}}{=} x$*,* $\lambda(t_{time}, s_C) \stackrel{\text{df}}{=} y'$*, où* $\forall i \in [1, n]$*,* $y'[i] \leftarrow y[i] + 1$*, et* $\lambda(t_{time}) \stackrel{\text{df}}{=} G_1 \wedge \cdots \wedge G_n$ *où* G_i *agit en tant que garde de la "borne supérieure" pour toutes les transitions dans l'agent* A_i*, c'est à dire,* $G_i \stackrel{\text{df}}{=} (g_1 \vee \cdots \vee g_{m_i})$ *avec* $m_i = |L_i|$ *et* $\forall j \in [1, m_i - 1]$*,* $g_j \stackrel{\text{df}}{=} (x[i] = l_i^j) \wedge (y[i] < B)$*, où* $B = \max\{b | (l_i^j, f, [a, b], l') \in l_i^{j\bullet}\}$ *est la plus grande des bornes supérieures des intervalles de toutes les transitions sortantes de* l_i^j *et* $g_{m_i} \stackrel{\text{df}}{=} (x[i] = l_i^{m_i}) \wedge (y[i] < E_i)$*. Ceci équivaut à une augmentation de temps.*
- $(M_0(s_A), M_0(s_C), M_0(s_V)) = \mathsf{Init}$ *est le marquage initial.*

La traduction associe des singletons en tant qu'annotations d'arc pour tous les arcs. En conséquence, lors de l'exécution, à partir du marquage initial qui associe un jeton à chaque

place, il y aura toujours exactement un jeton dans chacune des places. Chaque marquage accessible, où la place s_A contient $\vec{l}$, la place s_C contient $\vec{c}$ et la place s_V contient v, encode un état $(\vec{l}, \vec{c}, v)$ du G-MAPT considéré.

La figure 4.2 présente la traduction du réseau de Petri du G-MAPT de l'Exemple. 1. Au marquage initial, t_1, t_2, t'_1 et t'_2 ne sont pas franchissables car le jeton lu dans s_C (le vecteur des valeurs d'horloge) ne satisfait pas les gardes, alors que r_1 et r_2 ne sont pas franchissables puisque le jeton lu dans s_A (le vecteur de localités) ne satisfait pas les gardes. En revanche, la transition $time$ est franchissable. Son franchissement lit [1] les jetons dans le places s_V et s_A, consomme $(0, 0)$ et produit $(1, 1)$ dans s_C. Dans ce nouveau marquage, $time$, $t1$ et $t2$ sont franchissables et le processus continue exactement comme dans le G-MAPT.

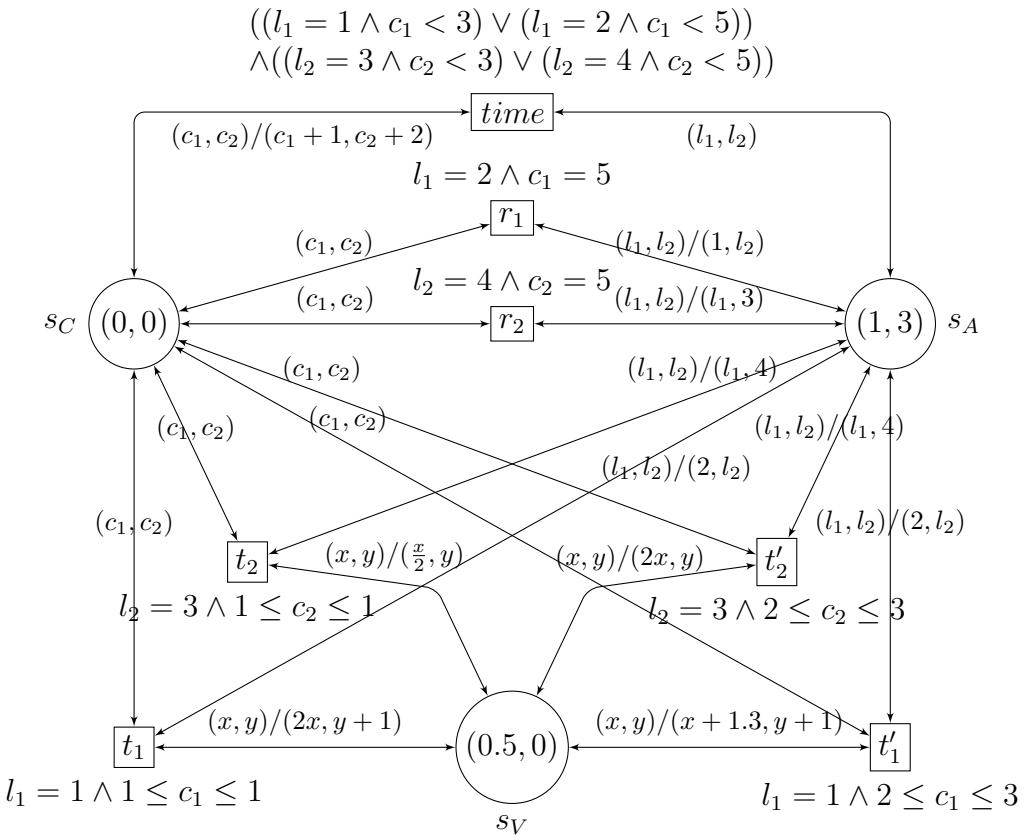

FIGURE 4.2 – Traduction du réseau de Petri du G-MAPT de l'Exemple 1 avec le marquage initial. Les arcs sont tous bidirectionnels et annotés par les paires w/z (ou juste w au lieu de w/w), ce qui signifie que w est l'étiquette de l'arc allant d'une place vers la transition et z est l'étiquette de l'arc allant dans le sens opposé. Pour des questions de lisibilité, $x[i]$ et $y[i]$ sont respectivement remplacés dans la figure par l_i et c_i.

1. cela signifie que le jeton est consommé et produit à l'identique

Proposition 5. *Un* G-MAPT Q *et sa traduction en réseau de Petri* $N = translate(Q)$ *ont des espaces d'états équivalents.*

Démonstration. Immédiate par les définitions. □

4.1.3 Accélération

Dans certains cas il est suffisant de ne s'intéresser qu'à la relation de causalité entre les actions de transitions. Les dates exactes auxquelles ces dernières ont lieu est alors sans importance. Il est alors souvent possible de réduire la taille de l'espace d'états d'origine. Pour ce faire, nous supposons que le G-MAPT satisfait la contrainte 1.

Nous pouvons d'abord considérer des **zones d'actions**, définies comme intervalles de temps maximal dans lequel les mêmes transitions et réinitialisations sont permises à partir de l'état actuel (notez cependant que, lorsqu'une réinitialisation est franchissable, la zone a une longueur de 1, car avant E_i, la réinitialisation correspondante n'est pas franchissable et nous ne pouvons pas dépasser E_i). Au lieu d'augmenter le temps par pas unitaires, on peut alors progresser en un seul pas d'une zone d'action à une autre, jusqu'à ce que nous décidions de franchir une transition ou une réinitialisation. Cela correspond généralement à une augmentation du temps de plusieurs unités à la fois. On notera que, lorsque l'on saute vers une zone, n'importe quel point de celle-ci peut être choisis car, par définition, tous se comportent de la même manière en ce qui concerne les transitions ou les réinitialisations possibles. Afin de lever les ambiguïtés, nous choisirons d'aller toujours à la fin de la zone.

Cependant, toutes les zones d'action accessibles depuis un état donné n'ont pas besoin d'être explorées : nous pouvons négliger les zones d'action qui sont **dominées** par d'autres, c'est à dire les zones pour lesquelles l'ensemble de transitions et de réinitialisations possibles est inclus dans une autre zone. Par exemple, supposons que les ensembles de transitions franchissables soient successivement (à partir de l'état actuel) : $\{t1, t2\} \rightarrow \{\mathbf{t1}, \mathbf{t2}, \mathbf{t3}\} \rightarrow \{t2, t3\} \rightarrow \{t3\} \rightarrow \{t3, t4\} \rightarrow \{\mathbf{t3}, \mathbf{t4}, \mathbf{t5}\} \rightarrow \{t4, t5\} \ldots$. Les zones d'action maximales (non dominées) sont indiquées en gras. On peut donc d'abord «sauter» (dans le temps) jusqu'à la fin de la zone permettant $\{t1, t2, t3\}$, puis choisir si nous voulons franchir $t1$ ou $t2$ ou $t3$, ou sauter à la fin de la zone permettant $\{t3, t4, t5\}$, où nous pouvons à nouveau décider de franchir ou non une transition. Si il n'y pas d'action non dominée plus loin, alors le temps n'augmentera pas et un franchissement de transition est alors nécessaire.

Pour poursuivre l'analyse, supposons que $s = (\vec{l}, \vec{c}, v)$ soit l'état actuel et que, pour chaque

agent A_i, on ait

$$B_i(s) \stackrel{\text{df}}{=} \begin{cases} E_i - c_i & \text{si } l_i = l_i^{m_i} \\ \max\{b - c_i \mid (l_i, f, [a,b], l') \in l_i^\bullet\} & \text{sinon} \end{cases}$$

$$B(s) \stackrel{\text{df}}{=} \min\{B_i(s) \mid i \in [1,n]\}$$

À partir de nos hypothèses, chaque $B_i(s)$, et donc chaque $B(s)$, est non négatif et nous ne pouvons augmenter le temps de plus de $B(s)$ unités de temps avant de choisir de déclencher une transition ou une réinitialisation. Notamment, si $B(s) = 0$, l'augmentation du temps empêcherait toute transition dans une localité d'être franchissable de nouveau. Pour éviter un tel blocage local, nous devons choisir une transition ou une réinitialisation à franchir.

Lorsque le temps évolue, si nous atteignons a ou E, l'ensemble des transitions et des réinitialisations possibles augmente (notez qu'un E se comporte à la fois comme un a et comme un b), et si nous dépassons un b (nous ne pouvons pas dépasser un E), cet ensemble diminue. Il faut donc trouver le premier b ou E précédé d'au moins un a.

Il est possible de calculer cela comme suit :

$$\mathbf{a}(s) \stackrel{\text{df}}{=} \begin{cases} \min(\alpha) & \text{si } \alpha \stackrel{\text{df}}{=} \{a - c_i \mid \exists\, (l_i, f, [a,b], l_i') \in T_i \text{ pour un agent } A_i \\ & \text{et } c_i < a \leq B(s)\} \cup \{E_i - c_i \mid l_i = l_i^{m_i} \text{ pour un agent } A_i \\ & \text{et } c_i < E_i \leq B(s)\} \neq \emptyset \\ 0 & \text{sinon} \end{cases}$$

$$\delta(s) \stackrel{\text{df}}{=} \begin{cases} \min(\beta) & \text{si } \beta \stackrel{\text{df}}{=} \{b - c_i \mid \exists\, (l_i, f', [a,b], l') \in l_i^\bullet \text{ pour un agent } A_i \\ & \text{avec } 0 < \mathbf{a}(s) \leq b - c_i \leq B\} \cup \{E_i - c_i \mid l_i = l_i^{m_i} \text{ pour un} \\ & \text{agent } A_i \text{ avec } E_i \leq c_i + B\ \} \neq \emptyset \\ 0 & \text{sinon} \end{cases}$$

On peut remarquer que $\mathbf{a}(s) > 0 \iff \alpha \neq \emptyset \iff \beta \neq \emptyset \iff \delta(s) > 0$.

Si $\mathbf{a}(s) = 0$, cela signifie qu'il n'y a aucun moyen de faire grandir l'ensemble des transitions franchissables ou les réinitialisations à l'avenir ; il n'y a donc aucun intérêt à laisser le temps évoluer (d'où $\delta(s) = 0$) et il faut maintenant choisir une transition ou une réinitialisation à franchir (le temps pourra éventuellement augmenter dans la nouvelle localité). Sinon, on peut franchir une transition ou une réinitialisation, ou effectuer un saut temporel de $\delta(s)$ unités de temps.

On peut remarquer que l'état initial ainsi que les états atteints après un franchissement ne sont pas nécessairement dans une zone maximale. Cela peut être vérifié facilement : à partir

de l'état en question, le premier b ou E est précédé par un ou plusieurs a. Nous pouvons alors forcer un saut temporel $\delta(s)$ comme calculé ci-dessus pour atteindre (la fin de) la première zone maximale avant de nous demander si nous allons effectuer un franchissement. Cependant, il est également possible d'effectuer un franchissement depuis l'état actuel.

Enfin, nous pouvons observer que, lorsque nous effectuons un franchissement dans un agent A_i, nous sommes positionnés avant $B(s)$ et donc avant $B_i(s)$, par définition. De part les contraintes 1.3 and 1.4, quel que soit le saut temporel effectué depuis l'état précédent, la (fin de) première zone maximale est identique, ce qui permet de ne manquer aucun franchissement permis dans la nouvelle localité atteinte.

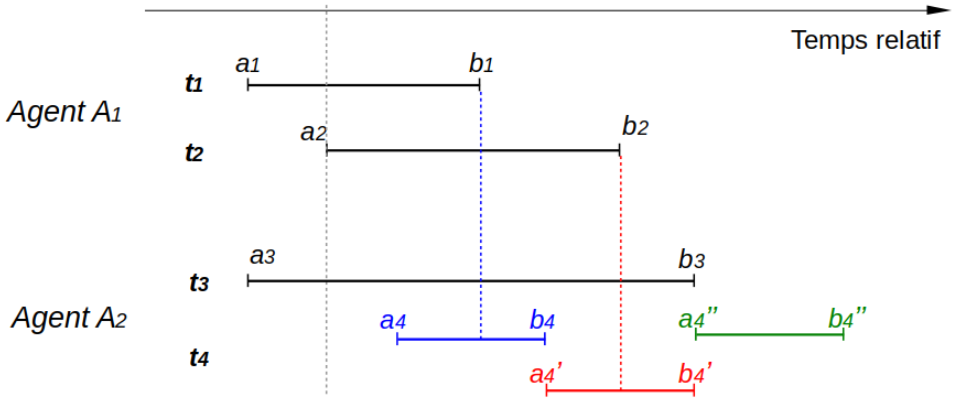

FIGURE 4.3 – Exemple d'augmentation du temps en fonction de l'accélération de la zone d'actions. Le temps actuel est indiqué par des pointillés gris, tandis que l'augmentation maximale possible pour chaque variante est indiquée avec sa couleur respective (bleu pour t_4, rouge pour t'_4 et vert pour t''_4).

Pour une meilleure compréhension, considérons l'exemple d'accélération illustré en Fig. 4.3, avec deux agents A_1 et A_2. Pour simplifier la présentation, nous supposons que les horloges C_1 et C_2 sont alignées, de sorte que les intervalles de temps des transitions peuvent être représentés dans le même espace. L'état actuel du système peut être décrit comme suit : à partir de la localité actuelle de l'agent A_1, des transitions sortantes t_1 et t_2 peuvent être franchies dans les intervalles respectifs $[0, 3]$ et $[1, 5]$, tandis que de la localité actuelle de l'agent A_2, les transitions sortantes t_3 et t_4 (en bleu dans la figure) peuvent être franchies respectivement en $[0, 6]$ et $[2, 4]$. Pour toute transition t_i, ses bornes inférieure et supérieure seront appelées respectivement a_i et b_i. Supposons que l'on soit actuellement à l'instant 1. Dans un tel cas, la zone d'actions actuelle est $[a_2, a_4[$ et elle permet le franchissement de t_1, t_2 et t_3. La zone d'actions suivante $[a_4, b_1]$ permet les quatre transitions (et est donc maximale). Donc, à partir de la zone d'actions actuelle, nous pouvons franchir l'une des t_1, t_2 ou t_3 ou augmenter le

temps. L'augmentation du temps (accéléré) devrait alors mener à la zone d'actions $[a_4, b_1]$, par exemple à b_1. De cette façon, nous inclurions toutes les séquences de transitions possibles, y compris le franchissement de t_4 suivi de t_1.

Considérons maintenant une variante de l'exemple dans laquelle t_4 est remplacé par t'_4 (en rouge dans la figure), avec un intervalle de temps de $[4, 6]$. Dans ce scénario, la zone d'actions actuelle est $[a_2, b_1]$ et permet t_1, t_2 et t_3. La zone suivante est $]b1, a4'[$, qui permet le franchissement de $t2$ et $t3$ uniquement : puisque ces transitions sont incluses dans la zone d'actions actuelle, cette zone n'est pas intéressante du point de vue de la causalité. Enfin, la zone $[a'_4, b_2]$ rend franchissable t_2, t_3 et t_4, ce qui est intéressant car une nouvelle transition est possible et la prochaine augmentation de temps doit donc conduire à (la fin de) cette zone. Il est important de noter que toutes les séquences impliquant t_1 sont conservées, étant donné que l'augmentation du temps n'est que l'une des évolutions possibles du système, les franchissements de t_1, t_2 et t_3 étant également possibles.

Enfin, considérons une seconde variante dans laquelle t_4 est remplacé par t''_4 (en vert sur la figure), avec un intervalle de temps de $[6, 8]$. Dans ce scénario, la zone d'actions actuelle est toujours $[a_2, b_1]$, ce qui rend franchissable t_1, t_2 et t_3. La zone d'actions suivante $]b1, b2]$ ne permet que $t2$ et $t3$. Comme auparavant, les transitions possibles sont incluses dans la zone d'actions actuelle, ce qui signifie que se rendre dans cette zone ne permettrait pas de nouvelles actions. Cependant, il n'est pas possible d'aller plus loin puisque nous avons atteint $B(s)$ (cela correspond au temps nécessaire pour atteindre b_2). Nous devons donc choisir une transition à franchir dans la zone actuelle. La transition t''_4 est actuellement non permise ; elle pourra l'être à l'avenir après qu'au moins t_1 ou t_2 ait été franchie.

Dans le contexte des réseaux de Petri, l'accélération peut être définie syntaxiquement (en modifiant la garde de la transition t_{time} et l'annotation de l'arc de la transition t_{time} vers la place s_C) et correspond au remplacement des éléments suivants dans la Définition 2 :

— $\lambda(t_{time}, s_C) = y'$, où $\forall i \in [1, n]$, $y'[i] \leftarrow y[i] + \delta(s)$;

— $\lambda(t_{time}) \stackrel{\text{df}}{=} \delta(s) > 0$.

Le calcul de $\delta(s)$ est possible grâce aux localités actuelles des agents encodées dans les jetons de la place s_A et les valeurs des horloges encodées dans les jetons de la place s_C.

Dynamique abstraite Afin de prouver que la sémantique accélérée est capable de conserver la causalité des MAPTs, nous allons considérer le graphe dont les nœuds sont les projections des évolutions de l'état initial sur l'ensemble des transitions et des réinitialisations. Autrement dit, si nous avons un mot sur l'alphabet composé de $+\delta$ (les augmentation du temps, avec $\delta = 1$ dans la sémantique d'origine, non accélérée), $t_{i,j}$ (les transitions de l'agent A_i) et de r_i (les

réinitialisation de l'agent A_i) représentant une évolution possible du système jusqu'à un certain point, en supprimant tous les $+\delta$, nous obtiendrons sa projection et un nœud du graphe abstrait. Les arcs (étiquetés) entre ces nœuds seront définis par la règle suivante : si α et αt (ou αr) sont deux nœuds, il existe un arc étiqueté t (ou r) entre eux. Ceci définit une arborescence étiquetée (généralement infinie) des exécutions de la sémantique (originale ou accélérée) du système considéré.

Le noeud initial sera étiqueté par la projection $(\vec{l}; v)$ de l'état initial $(\vec{l}, \vec{c}; v)$. Cela détermine récursivement les autres nœuds : si $(\vec{l}; v)$ est l'étiquette d'un nœud et qu'il existe un arc étiqueté $t = (l_i, f, [a, b], l'_i)$ allant de ce nœud vers un autre, ce dernier sera étiqueté $(\vec{l'}; f(v))$, où $\vec{l'}$ est $\vec{l}$ avec l_i remplacé par l'_i ; et si l'arc est étiqueté r_i, le noeud de destination sera $(\vec{l'}; v)$, où $\vec{l'}$ est $\vec{l}$ avec l_i remplacé par l_i^1.

A titre d'illustration, considérons le MAPT de l'Exemple1, où nous négligeons les valeurs de la variable pour simplifier la présentation. Le fragment initial de la dynamique originale et accélérée ainsi que la dynamique abstraite correspondante sont représentés en Fig. 4.4, en supposant initialement que les horloges sont égales à 0, que A_1 est dans l'état 1 et A_2 dans l'état 3.

Proposition 6. *La sémantique originale et la sémantique accélérée d'un* MAPT *conduisent à la même dynamique abstraite.*

Démonstration. Il suffit de montrer que l'ensemble des projections (non temporisées) des évolutions de la sémantique originale est identique à celui de la sémantique accélérée.

Tout d'abord, nous pouvons observer que chaque évolution de la sémantique accélérée est également une évolution de la première : une augmentation de temps de $\delta(s)$ unités de temps pour atteindre une zone d'action maximale est identique à $\delta(s)$ augmentation de temps de 1 unité de temps ; En effet, par définition, $\delta(s) \leq B(s)$ et dans les deux cas, nous avons au final $B(s) - \delta(s) = B(s) - \delta(s) \cdot 1 \geq 0$.

Il reste donc à montrer que, si $\mathsf{word}(\omega)$ est la projection d'une évolution ω de la sémantique originale, c'est aussi la projection d'une évolution ω' de la sémantique accélérée. Nous allons procéder par induction sur la longueur de ω et montrer plus exactement que pour chaque ω, il existe une évolution accélérée ω' telle que $\mathsf{word}(\omega') = \mathsf{word}(\omega)$ et l'ensemble des transitions et réinitialisations franchissables après ω est inclus dans celui après ω'.

La propriété est satisfaite initialement, lorsque $\omega = \omega' = \varepsilon$ mais également si l'ensemble franchissable n'est pas maximal et que nous choisissons l'accélération allant au premier ensemble franchissable maximal via une augmentation de $\delta(s)$ unités de temps. En effet, nous savons par définition qu'une augmentation du temps de $\delta(s)$ unités de temps mène toujours au premier ensemble franchissable maximal et nulle part ailleurs. Ensuite, dans ce dernier cas, par

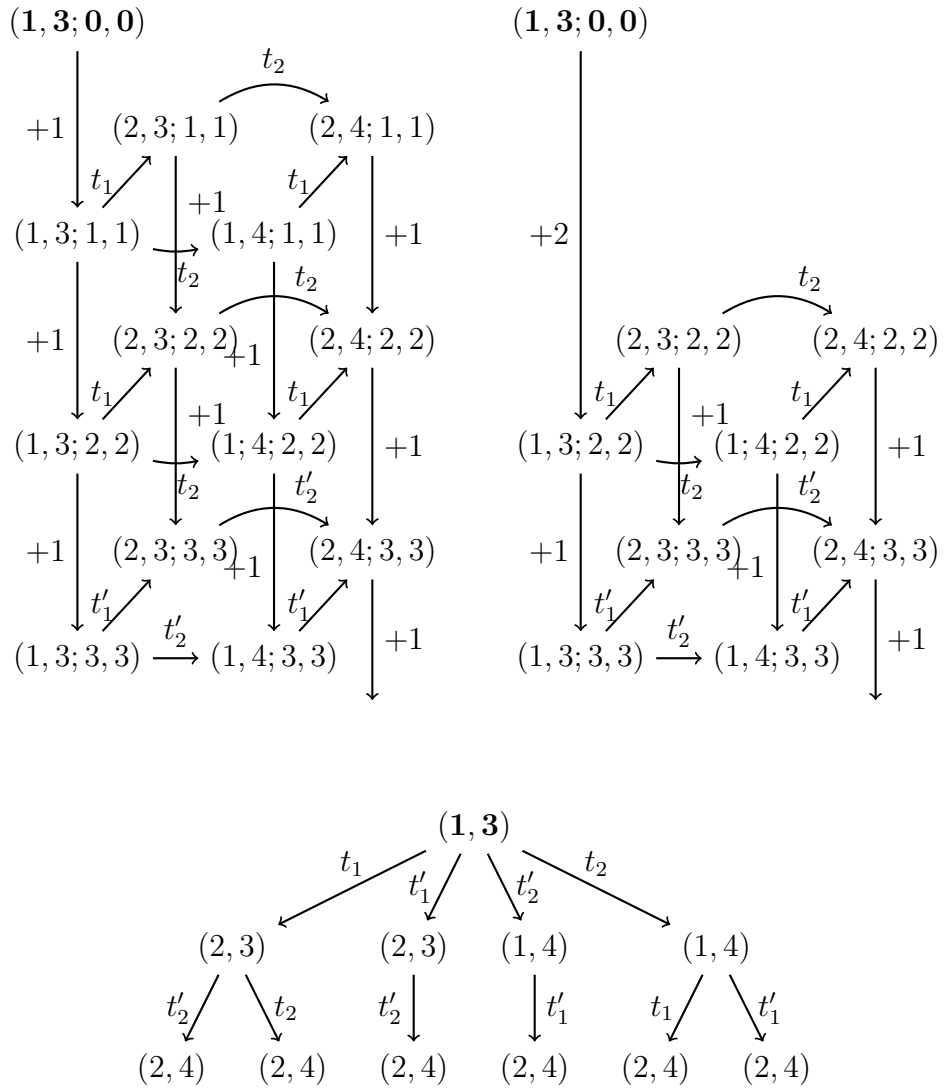

FIGURE 4.4 – Les fragments initiaux (sans variables) des différentes dynamiques pour le MAPT de l'Ex 1. En haut à gauche : la dynamique d'origine. En haut à droite : la dynamique accéléré. En bas : la dynamique abstraite.

définition, $\delta(s) \leq B(s)$ et l'ensemble des transitions et réinitialisations franchissables s'agrandit.

La dynamique abstraite conserve les transitions et les réinitialisations, qui sont seules à affecter les localités et la variable. Ainsi, si $\mathsf{word}(\omega) = \mathsf{word}(\omega')$, les localités et la variable sont les mêmes après ω et ω'. Soit $\Delta(\omega)$ le temps écoulé au cours de l'évolution décrite par ω. Nous pouvons observer que les horloges sont déterminées par $\mathsf{word}(\omega)$ et $\Delta(\omega)$, indépendamment du moment où l'augmentation du temps s'est produite : pour tout agent A_i, $c_i = \mathsf{init}_i + \Delta(\omega) - E_i \cdot \#_{r_i}(\omega)$, où $\#_{r_i}(\omega)$ est le nombre de réinitialisations de A_i dans $\mathsf{word}(\omega)$. Egalement, nous ne pouvons pas effectuer plus de $\min_i\{E_i\}$ augmentations de temps à la suite, car une réinitialisation devrait avoir lieu auparavant.

Nous allons maintenant supposer que ω étend $\widetilde{\omega}$ par un événement, que $\widetilde{\omega}$ et $\widetilde{\omega}'$ forment une paire correspondante et qu'il est alors possible de construire une évolution accélérée correspondante ω'.

Si $\omega = \widetilde{\omega} \cdot (+1)$, c'est-à-dire si ω est obtenu à partir de $\widetilde{\omega}$ en ajoutant une augmentation de temps (de 1 unité de temps), la projection de ω est la même que celle de $\widetilde{\omega}$, donc de $\widetilde{\omega}'$ par l'hypothèse d'induction. Si l'ensemble devenu franchissable après ω est inclus dans celui après $\widetilde{\omega}'$, celui-ci satisfait toujours l'hypothèse d'induction. Si l'ensemble rend franchissable après ω n'est plus inclus dans celui après $\widetilde{\omega}'$, cela signifie que nous avons atteint un ou plusieurs a qui n'étaient pas encore atteints par $\widetilde{\omega}'$. Nous pouvons en déduire que $\Delta(\widetilde{\omega}') < \Delta(\omega)$. Mais ensuite, aller dans la prochaine zone d'actions maximale implique, pour la sémantique accélérée, d'atteindre ces a (sans dépassement de $B(s)$, sans quoi cela se produirait également pour ω, ce qui obligerait à effectuer une transition ou à réinitialisation après $\widetilde{\omega}$) et de récupérer l'hypothèse d'induction.

Si $\omega = \widetilde{\omega} \cdot r_i$, pour un agent A_i, on a $init_i + \Delta(\omega) = \mathsf{init}_i + \Delta(\widetilde{\omega}) = k \cdot E_i$ pour un facteur k, avec r_i appartenant à l'ensemble des transitions ou réinitialisations franchissables après $\widetilde{\omega}$. Cependant, une zone d'actions permettant une réinitialisation est un intervalle comprenant exactement une unité de temps et est maximale. Par conséquent, après $\widetilde{\omega}'$, on a la même zone d'actions et $\mathsf{init}_i + \Delta(\widetilde{\omega}') = k \cdot E_i$ (le même facteur que pour ω puisque l'augmentation du temps entre les réinitialisations est limitée). On peut donc aussi effectuer r_i après $\widetilde{\omega}'$, l'état après $\widetilde{\omega}' \cdot r_i$ est alors identique à après ω et la situation est la même qu'initialement.

Si $\omega = \widetilde{\omega} \cdot t$, pour une transition t d'un agent A_i, par l'hypothèse d'induction t peut également se produire après $\widetilde{\omega}'$ et $\mathsf{word}(\omega) = \mathsf{word}(\widetilde{\omega}' \cdot t)$. Tout t' franchissable après $\widetilde{\omega}$ dans A_j pour $j \neq i$ reste franchissable après ω ainsi que après $\widetilde{\omega}' \cdot t$, par l'hypothèse d'induction. Pour l'agent A_i, à partir du troisième élément de contrainte 1, aucune transition sortante de la nouvelle localité n'a déjà atteint le point à partir duquel elle n'est plus franchissable dans la sémantique originale (après ω) comme dans la sémantique accélérée (après $\widetilde{\omega}'$). Si $\Delta(\omega) \leq \Delta(\widetilde{\omega}' \cdot t) = \Delta(\widetilde{\omega}')$, l'hor-

loge C_i de A_i n'est pas plus grande après ω qu'après $\widetilde{\omega}' \cdot t$ de sorte que toutes les transitions franchissables de A_i après ω soient également franchissables après $\widetilde{\omega}' \cdot t$, et l'hypothèse d'induction reste valide. Au contraire, si $\Delta(\omega) > \Delta(\widetilde{\omega}' \cdot t)$, il peut arriver que des $\widetilde{t}$ dans A_i soient franchissables après ω mais pas après $\widetilde{\omega}' \cdot t$; cependant, à partir de $\widetilde{\omega}'$, il est alors possible d'avoir une augmentation du temps de $\Delta(\omega) - \Delta(\widetilde{\omega}' \cdot t)$, ce qui conduit au même état qu'après ω ; il est alors également possible de considérer une zone d'actions maximale après $\widetilde{\omega}' \cdot t$ qui englobe toutes les transitions franchissables après ω pour l'atteindre dans la sémantique accélérée, et l'hypothèse d'induction reste valide.

□

4.2 Algorithmes d'exploration

Lors de la vérification formelle d'un système, on a généralement le choix entre explorer l'espace d'états en profondeur ou en largeur d'abord. Pour les propriétés d'accessibilité, où l'on cherche si un état satisfaisant une propriété spécifique peut être atteint, la profondeur d'abord (dirigée et limitée par la requête) est généralement considérée comme plus efficace. Cependant, la majorité du non-déterminisme dans les systèmes présentant un niveau élevé de concurrence (tels que les systèmes de CAVs) conduit à des diamants. En effet, si des transitions de différents agents sont disponibles depuis un état donné, elles peuvent se produire dans plusieurs ordres possibles, tous convergeant la plupart du temps vers le même état (voir le paragraphe sur la persistance en Section 4.1.1). Afin d'éviter d'explorer encore et encore les mêmes états, une exploration en profondeur doit stocker tous les états déjà visités jusqu'à présent, ce qui est généralement impossible à faire pour de gros systèmes. Par exemple, si les états $s1$ et $s2$ partagent un successeur commun $s3$, l'algorithme calculera les successeurs de $s1$, puis supprimera $s1$ de la mémoire et poursuivra avec ses successeurs, jusqu'à atteindre $s3$ et explorera tous les chemins de $s3$, en oubliant à chaque fois les nœuds déjà visités. Ainsi, lorsque l'algorithme aura exploré tous les chemins de $s1$ et commencera à explorer à partir de $s2$, il n'y a pas de trace mémoire que $s3$ a déjà été exploré et tous les chemins à partir de celui-ci seront explorés à nouveau.

Au contraire, utiliser des algorithmes de largeur d'abord évite ce problème, car les états doublons obtenus à une profondeur donnée peuvent être supprimés. Cependant, cela impliquerait également d'explorer tous les états atteignables à une profondeur donnée et d'interdire l'utilisation d'heuristiques pour diriger et limiter l'exploration.

Une idée est alors d'essayer de combiner les deux approches.

4.2.1 Espace d'états en couches

L'espace d'états d'un MAPT présente des caractéristiques intéressantes : hormis l'absence de cycles (voir Prop. 3 : l'espace d'états dans notre cas est toujours un DAG), sa structure peut souvent être divisée en couches de sorte que tous les états en bordure d'une couche partagent les mêmes vecteurs de localités et d'horloges (et donc le même ensemble de transitions franchissables). et sont situés à la même distance temporelle de l'état initial. La seule différence concerne la valeur de la variable v, due au non déterminisme et à la concurrence inhérente à ce type de modèle. Le non-déterminisme signifie qu'un agent a le choix entre plusieurs transitions à un moment donné ; La concurrence signifie qu'au moins deux agents peuvent effectuer des transitions à un moment donné. Dans le premier cas, l'agent peut suivre plusieurs chemins pour atteindre un certain point, ce qui conduit à des valeurs différentes de la variable ; dans le second cas, les transitions des deux agents peuvent être commutées, ce qui conduit à nouveau à des valeurs différentes de la variable.

Ceci est schématisé par la Fig. 4.5, où l'on peut voir comment l'espace des états est divisé en sous-espaces (chacun étant un DAG avec un état initial unique) tel que chaque état final d'un sous-espace est l'état initial d'un autre. Ces sous-espaces peuvent se croiser.

Plus formellement, dans un DAG, nous avons un ordre partiel naturel : $s_1 < s_2$ s'il existe un chemin non vide de s_1 à s_2 ; s_1 et s_2 sont incomparables s'il n'y a pas de chemin non vide entre eux.

Une coupe est un sous-ensemble maximal d'états incomparables. Une coupe divise l'espace partiellement ordonné en trois sous-ensembles : les états avant la coupe, la coupe elle-même et les états après la coupe.

Dans ce qui suit, nous noterons ω une évolution (éventuellement vide) menant d'un état s à un état s', c'est-à-dire la séquence de transitions, de réinitialisations et d'augmentations de temps étiquetant un chemin allant de s à s' dans l'espace d'états du modèle considéré. Comme auparavant, on notera par $\Delta(\omega)$ la somme des augmentations de temps le long de ω, également appelée distance temporelle de s à s' (le long de ω).

Definition 3. *Dans un* MAPT *(dont l'espace d'états est toujours un* DAG*), une coupe est dite* ***cohérente*** *si tous ses états partagent les mêmes vecteurs de localités et d'horloges, et que deux évolutions quelconques ω_1, ω_2 reliant l'état initial aux états de la coupe ont la même distance temporelle : $\Delta(\omega_1) = \Delta(\omega_2)$. Des coupes cohérentes peuvent être utilisées pour définir les limites des couches.*

Soit s un état arbitraire dans un MAPT*; les états accessibles depuis s forment un sous-espace acyclique, dans lequel on peut également définir des coupes cohérentes : un cône cohérent avec un apex s est l'ensemble des états jusqu'à une coupe cohérente dans ce sous-espace*

(y compris s et la coupe).

Proposition 7.

— *Les coupes cohérentes ne se croisent pas, dans le sens suivant. Soit $\mathcal{C}_1$ et $\mathcal{C}_2$ deux coupes cohérentes dans un* MAPT*, $s_1, s'_1 \in \mathcal{C}_1$, $s_2, s'_2 \in \mathcal{C}_2$. S'il existe une évolution ω de s_1 à s_2 et ω' de s'_2 à s'_1, alors $\mathcal{C}_1 = \mathcal{C}_2$ et $\omega = \varepsilon = \omega'$. En particulier, deux coupe cohérentes distincte ne peuvent avoir un état commun.*

— *La distance temporelle entre les coupes cohérentes est constante, dans le sens suivant. Soit $\mathcal{C}_1$ et $\mathcal{C}_2$ deux coupes cohérentes différentes dans un* MAPT*, s_1, $s'_1 \in \mathcal{C}_1$, s_2, $s'_2 \in \mathcal{C}_2$, avec une évolution ω de s_1 à s_2 et ω' de s'_1 à s'_2, alors $\Delta(\omega) = \Delta(\omega')$.*

— *Si $s_1 \in \mathcal{C}_1$ et qu'il existe un cône cohérent avec une distance temporelle Δ (pour toute évolution ω de s_1 à la base du cône, $\Delta(\omega) = \Delta$), alors il y a une coupe cohérente $\mathcal{C}_3$ séparée de $\mathcal{C}_1$ par une distance temporelle Δ.*

Démonstration.

— Dans le premier cas, s'il existe un chemin $\widetilde{\omega}$ de s_0 à s_1 et un chemin $\widetilde{\omega}'$ de s_0 à s'_2, nous devons avoir $\Delta(\widetilde{\omega}) + \Delta(\omega) = \Delta(\widetilde{\omega}\omega) = \Delta(\widetilde{\omega}')$ et $\Delta(\widetilde{\omega}') + \Delta(\omega') = \Delta(\widetilde{\omega}'\omega') = \Delta(\widetilde{\omega})$, donc $\Delta(\omega) = -\Delta(\omega')$, ce qui est possible uniquement si $\Delta(\omega) = 0 = \Delta(\omega')$.
De plus, puisque s_2 et s'_2 ont les mêmes localités et horloges, il existe une évolution ω' de s'_2 vers un état s''_1 avec les mêmes localités et horloges que s_1, d'où une évolution $\omega\omega'$ de distance temporelle 0 de s_1 à s''_1, qui donne les mêmes localités et les mêmes horloges. Puisque les localités de chaque agent A_i forment un DAG et que $E_i > 0$, cela n'est possible que si $\omega = \varepsilon = \omega'$.
En particulier, si $\omega = \varepsilon$, c'est-à-dire si $s_1 = s_2$, on a également $s'_1 = s'_2$ et $\mathcal{C}_1 = \mathcal{C}_2$, et de même si $\omega' = \varepsilon$.

— Dans le cas suivant, si s_0 est l'état initial et qu'il existe un chemin $\widetilde{\omega}$ de s_0 à s_1 et un chemin $\widetilde{\omega}'$ à partir de s_0 à s'_1, on doit avoir $\Delta(\widetilde{\omega}) + \Delta(\omega) = \Delta(\widetilde{\omega}\omega) = \Delta(\widetilde{\omega}'\omega') = \Delta(\widetilde{\omega}') + \Delta(\omega')$, d'où la propriété.

— La dernière propriété résulte de l'observation selon laquelle, si $s'_1 \in \mathcal{C}_1$, étant donné que s_1 et s_2 ont les mêmes localités et les mêmes horloges, toute évolution de s_1 à la base du cône est également présente à partir de s'_1 et conduit à un état avec les mêmes localités et les mêmes horloges que les états situés à la base du cône (mais les variables peuvent différer). Et inversement, si un chemin mène de s'_1 à un état de $\mathcal{C}_3$, il a une distance temporelle Δ et il existe un chemin identique de s_1 vers un état de la base du cône.

□

Par exemple, si l'agent A_i dans un MAPT commence à l_i^1 avec une horloge nulle, après E_i unités de temps et avant $(E_i + 1)$ unités de temps, il passera nécessairement par sa réinitialisation (il est possible qu'il effectue d'autres transitions avant et / ou après cette réinitialisation sans modifier son horloge, mais il est certain que l'agent effectuera cette réinitialisation avant la prochaine augmentation de temps). Par conséquent, si chaque agent part de sa localité initiale avec une horloge nulle, après $\mathsf{ppcm}\{E_1, \ldots, E_n\}$ (le plus petit commun multiple des différentes périodes de réinitialisation, noté $\mathsf{ppcm}(E)$), il est certain que nous pourrons revenir à l'état initial, mais éventuellement avec différentes valeurs de la variable, donnant la bordure d'une couche. À partir de cette bordure, les mêmes séquences de transitions / réinitialisations / augmentations de temps qu'au départ se produiront périodiquement (avec une période de $\mathsf{ppcm}(E)$, conduisant à de nouvelles bordures, avec les localités initiales et les horloges nulles.

La situation sera similaire si $\forall A_i : \mathsf{init}_i = \mathsf{init} \mod E_i$ pour une valeur $\mathsf{init} < \mathsf{ppcm}(E)$. En effet, pour chaque agent A_i et chaque k, après $E_i - \mathsf{init}_i + k \cdot E_i$ augmentations de temps, nous visiterons l_i^1 avec une horloge nulle, donc nous visiterons les nouvelles bordures après $(k+1) \cdot \mathsf{ppcm}(E) - \mathsf{init}$ augmentations de temps.

Pour d'autres valeurs initiales des horloges, il n'est pas certain que nous pourrons structurer l'espace d'états en couches, mais dans les deux cas, il se peut également qu'il existe d'autres types de bordures.

Considérons par exemple le système illustré en haut de la Fig. 4.6, où chaque agent commence par une horloge nulle dans sa localité initiale. L'agent A_1 est déterministe puisqu'il existe une seule transition provenant de l_1^1 ainsi que de l_1^2, et l'agent A_2 ne l'est pas, car deux transitions peuvent se produire quand il se trouve dans l_2^2. Si on ne tiens pas compte de la valeur de la variable v, le graphe pour A_1 est périodique avec une période de $E_1 = 10$ et le graphe pour A_2 est périodique avec une période de $E_2 = 15$. L'ensemble du système est donc périodique avec une période de $\mathsf{ppcm}(E) = 30$. La séquence d'intervalles illustrée au bas de la figure pour A_1 représente les intervalles ou les transitions doivent avoir lieu, avec leur distance temporelle par rapport à l'état initial (ici, ces intervalles sont disjoints, mais ils peuvent aussi se chevaucher). Notez que si une horloge initiale init_i était strictement positive, la séquence d'intervalles pour l'agent i serait décalée vers la gauche de init_i unités de temps. Pour A_2, les intervalles indiqués sur la figure ont une interprétation différente, qui sera expliquée plus loin.

Une bordure ne peut pas se produire à un moment t (mesuré à partir du début du système) si, pour un agent déterministe A_i, il y a un intervalle $[a, b]$, tel que

$$a + k \cdot E_i - \mathsf{init}_i < t < b + k \cdot E_i - \mathsf{init}_i$$

En effet, dans ce cas, au moment t, A_i peut être soit à la source, soit à la destination de la

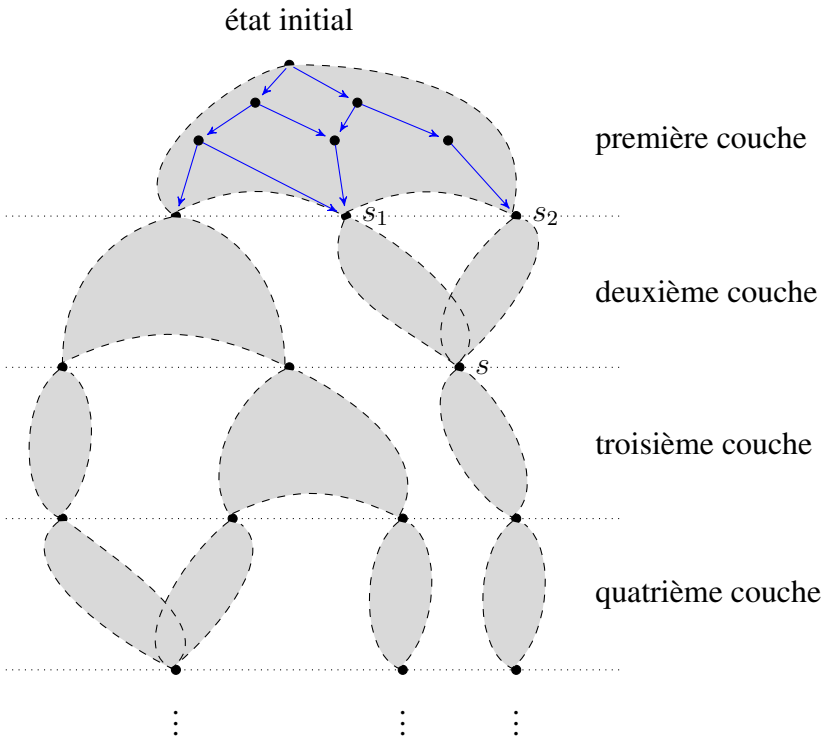

FIGURE 4.5 – Forme générale d'un espace d'états en couches avec un zoom sur la partie initiale. Les états identiques sont fusionnés. Tous les états situés en bordure d'une couche partagent les mêmes vecteurs de localités et d'horloges mais ont des valeurs différentes pour v. Chaque sous-espace grisé est un DAG ayant un état initial unique et un ou plusieurs états finaux. Le sous espace de la partie initiale (entouré de lignes pointillées) détaille en bleu les états et transitions présents.

transition correspondante, sans pouvoir être certain de la localité.

Par conséquent, un agent déterministe peut permettre qu'une bordure se situe à un instant t si l'un des cas suivants se produit :

— Si t est strictement compris entre les divers intervalles de A_i, nous savons immédiatement que lorsque nous arrivons à ce moment, nous nous trouvons sur une localité spécifique de A_i. Par exemple, au moment $t = 19$ en Fig. 4.6, nous sommes sûrs que A_1 se trouve sur la localité l_1^3.

— Si t est situé à la fin d'un intervalle, l'agent peut se trouver dans la localité source ou dans la localité de destination. La première situation n'existe pas dans tous les chemins, car il est possible de quitter la source avant t, tandis que la deuxième situation existe dans tous les chemins. Par conséquent, la deuxième situation est appropriée pour une coupe cohérente. Ce cas se produit en Fig. 4.6, par exemple lorsque le système atteint le moment $t = 5$, A_1 peut être dans l_1^1 ou dans l_1^2.

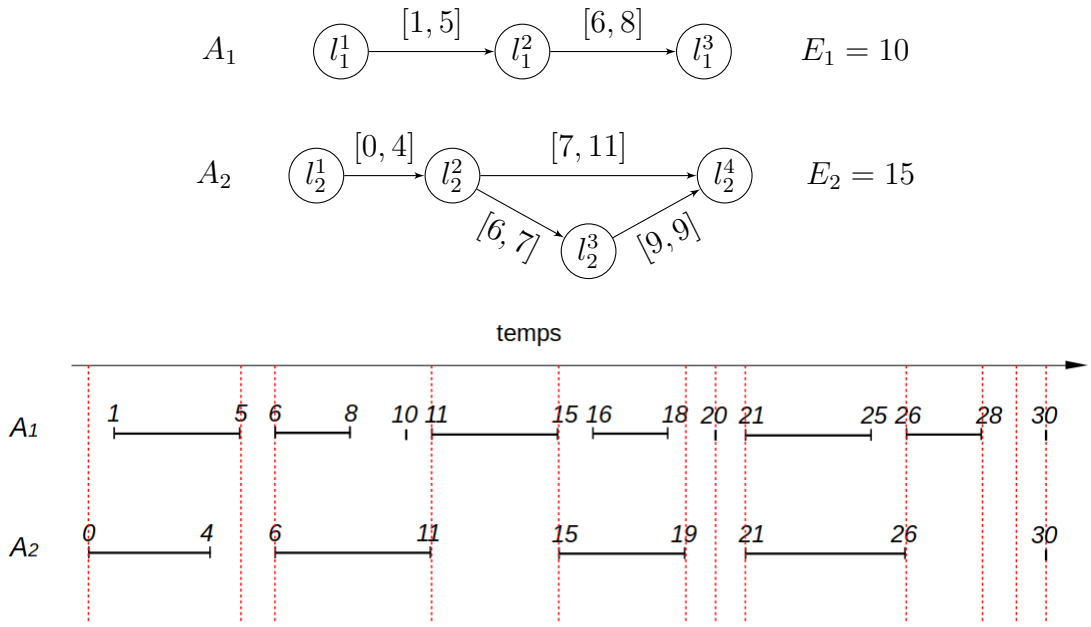

FIGURE 4.6 – En haut : Exemple de MAPT composé des agents A_1 et A_2 avec les horloges C_1 et C_2 initialisées à 0. En bas : intervalles de temps où une transition ou un ensemble de transitions peut être effectué. Les lignes en pointillés rouges indiquent les unités de temps où une coupe cohérente peut exister.

— Si t est situé au début d'un intervalle, l'agent peut se trouver dans la localité source ou dans la localité de destination. Ceci est symétrique au cas précédent, et c'est la première situation qui existe dans tous les chemins et qui convient pour une coupe cohérente. Ce cas se produit en Fig. 4.6, par exemple lorsque le système atteint l'instant, $t = 6$, A_1 peut être dans l_1^2 ou dans l_1^3.

— Si t est situé sur un intervalle de longueur 0 (comme une réinitialisation ou une transition avec un intervalle où $a = b$). Cela correspond à une union des deux cas précédents, où deux localités sont possibles. Ici, les deux conviennent à une coupe cohérente. Ce cas se produit en Fig. 4.6, par exemple, lorsque le système atteint l'instant $t = 20$, A_1 peut être soit dans l_1^3, soit dans l_1^1.

— Si t est à la fois à la fin d'un intervalle et au début d'un autre (ce qui signifie qu'ils se croisent sur t), cela revient à une combinaison des cas précédents. Ainsi, une situation appropriée pour une coupe cohérente consiste à considérer le système après avoir effectué la transition correspondant à l'intervalle qui précède et avant de réaliser la transition correspondant à l'intervalle qui suit. Une occurrence particulière de ce cas est visible en Fig. 4.6 à l'instant 15, où A_2 est à la droite d'un intervalle de longueur 0 correspondant à sa réinitialisation et à la gauche de l'intervalle de la transition de l_2^1. Dans cette situation,

A_2 peut être soit dans l_2^4, l_2^1 ou l_2^2. Le fait que l'un des intervalles soit de longueur 0 est inclus dans le cas général.

Pour un agent non déterministe, comme A_2 dans la Fig. 4.6, l'analyse est similaire mais légèrement plus complexe ; en effet, même entre les intervalles, il peut se trouver dans plusieurs localités possibles [2]. Par exemple, au moment $t = 7$, A_2 peut être soit en l_2^3 après avoir effectué une transition au moment 6, soit en l_2^2, et nous ne pouvons pas forcer le système à attendre que A_2 aille dans l_2^3 puisqu'il a la possibilité de choisir l'autre transition. L'idée est alors de considérer les localités de l'agent non déterministe A_i considéré qui sont en elles-mêmes des coupes uniques dans le DAG de ses localités, c'est-à-dire les localités visitées à chaque itération complète (de l_i^1 à l_i^m). Pour A_2 dans la Fig. 4.6, ces localités sont l_2^1, l_2^2 et l_2^4. Ces localités forment une séquence dans L_i : soit P_i cette liste et $\mathsf{succ}(l)$ le successeur de l dans P_i. Entre deux localités l et $\mathsf{succ}(l)$ dans P_i, ou bien il existe une transition unique activée dans un intervalle $[a, b]$ (comme dans le cas déterministe ci-dessus) ou bien il y a au moins deux chemins différents avec éventuellement plusieurs transitions activées à un moment donné dans l'intervalle $[\tilde{a}, \tilde{b}]$, où $\tilde{a}$ est la plus petite borne inférieure de toutes les transitions sortantes depuis l et $\tilde{b}$ est la plus grande borne supérieure de toutes les transitions entrantes vers $\mathsf{succ}(l)$. On peut penser à $[\tilde{a}, \tilde{b}]$ comme l'intervalle d'activation d'une transition virtuelle de l à $\mathsf{succ}(l)$. Ensuite, exactement le même argument que ci-dessus peut être utilisé pour vérifier si l'espace d'états d'un MAPT admet des couches, ce qui constitue une preuve de la proposition suivante.

Cela équivaut à une preuve de la proposition suivante :

Proposition 8. *Un* MAPT *admet un espace d'états en couches dont les bordures se situent à* $t + \ell \cdot \mathsf{ppcm}(E)$ *avec* $\ell \in \mathbb{N}$ *si, pour chaque agent* A_i*, pour chaque* $l \in P_i \setminus \{l_i^m\}$*, et pour* $\tilde{a} \stackrel{\text{df}}{=} \min\{a \mid (l, f, [a, b], l') \in l^\bullet\}$, $\tilde{b} \stackrel{\text{df}}{=} \max\{b \mid (l', f, [a, b], \mathsf{succ}(l)) \in {}^\bullet\mathsf{succ}(l)\}$*, il n'existe aucun* $k \in \mathbb{N}$ *qui satisfasse :* $\tilde{a} + k \cdot E_i - \mathsf{init}_i < t < \tilde{b} + k \cdot E_i - \mathsf{init}_i$*, où* init_i *est la valeur initiale de* C_i*.* □

Une bordure de couche détectée de cette manière sera alors définie par le couple $((l_1, \cdots, l_n), (c_1, \cdots, c_n))$ où pour l'agent A_i, $c_i = (t + \mathsf{init}_i) \mod E_i$ et l_i est la localité atteinte périodiquement à la valeur d'horloge c_i dans tous les chemins possibles.

Rechercher t plus efficacement La proposition 8 permet de rechercher les coupes cohérentes dont les ensembles de localités et d'horloges se retrouvent toutes les $\mathsf{ppcm}(E)$ unités de temps. Par conséquent, il n'est pas nécessaire de prendre en compte les temps t au-delà de $\mathsf{ppcm}(E)$. En outre, on pourrait conclure de cette proposition qu'il est nécessaire de considérer toutes les possibilités de décalage de chaque intervalle $[\tilde{a}, \tilde{b}]$, sur $\mathsf{ppcm}(E)$, c'est à dire toutes les valeurs

2. bien sûr, pas en même temps mais pour des exécutions différentes.

entières pour k. Ce n'est pas le cas : pour chaque $[\tilde{a}, \tilde{b}]$ il suffit de considérer le plus grand k respectant la contrainte de gauche $\tilde{a} + k \cdot E_i - \mathsf{init}_i < t$, c'est à dire, le plus grand k_a tel que $k_a < \frac{t+\mathsf{init}_i-\tilde{a}}{E_i}$, qui est donné par la formule $k_a = \lceil \frac{t+\mathsf{init}_i-\tilde{a}}{E_i} - 1 \rceil$. Nous devons ensuite vérifier que $t \geq \tilde{b} + k_a \cdot E_i - \mathsf{init}_i$.

Si nous voulons éviter les extrémités des intervalles $[\tilde{a}, \tilde{b}]$ afin que le vecteur d'horloge seul soit suffisant pour définir une coupe cohérente sans ambiguïté, on obtient qu'il ne doit pas exister de k tel que $\tilde{a} + k \cdot E_i - \mathsf{init}_i \leq t \leq \tilde{b} + k \cdot E_i - \mathsf{init}_i$. Ceci conduit à une formule plus simple $k_a = \lfloor \frac{t+\mathsf{init}_i-\tilde{a}}{E_i} \rfloor$ et à vérifier si $t > \tilde{b} + k_a \cdot E_i - \mathsf{init}_i$.

Combinaison avec la sémantique accélérée Lorsque nous considérons uniquement les vecteurs de localités et d'horloges et que nous ne tenons pas compte de la valeur de v, une coupe cohérente devient un point de passage obligatoire dans la dynamique d'origine (non accélérée) du système. Chaque état accessible dans la dynamique accélérée est également accessible dans celle d'origine et pour chaque chemin existant entre deux états dans la dynamique accélérée, il existe également au moins un chemin dans la dynamique originale. Cependant, comme les pas de temps peuvent être supérieurs à une unité dans la dynamique accélérée, il peut arriver qu'une augmentation de temps dépasse le vecteur d'horloge correspondant à une coupe cohérente $(\vec{l}, \vec{c})$ présente dans la dynamique originale. Dans un tel cas, un état $(\vec{l}, \vec{c} + \delta)$, atteint après avoir dépassé $(\vec{l}, \vec{c})$ est en fait une coupe cohérente de la dynamique accélérée, comme illustré en Figure 4.7.

On peut observer que dans les deux dynamiques, tous les chemins depuis l'état initial arrivent soit sur $((1,2);(5,5))$ soit sur $((2,2);(4,4))$. Dans la dynamique initiale, la coupe cohérente à $((2,2);(5,5))$ est atteinte et, après une augmentation de temps, la coupe cohérente à $((2,2);(6,6))$ est atteinte. À partir de $((2,2);(6,6))$, trois actions sont possibles (deux transitions et une augmentation du temps). Dans la dynamique accélérée, il est encore possible depuis $((1,2);(5,5))$ d'atteindre $((2,2);(5,5))$, mais pas depuis $((2,2);(4,4))$ puisque l'accélération mène directement à $((2,2);(7,7))$. Depuis $((2,2);(5,5))$, toujours dans la dynamique accélérée, l'accélération conduit également à $((2,2);(7,7))$, cet état correspondant à la fin de la première zone d'actions maximales. Ainsi, $((2,2);(6,6))$ n'est plus une coupe cohérente, car il n'est pas atteignable, mais $((2,2);(5,5))$ non plus n'est pas un coupe cohérente car il existe des chemins dans la sémantique accélérée qui dépassent son vecteur d'horloge sans l'atteindre. Cela montre que, dans la dynamique accélérée, les états avec le vecteur de localités correspondant à une coupe cohérente peuvent être entrés avec différentes valeurs d'horloge, mais à partir de ces états (ici $((2,2);(4,4))$ et $((2,2);(5,5))$) l'accélération conduira toujours aux états avec les mêmes vecteurs de localités et d'horloges (ici $((2,2);(7,7))$).

Proposition 9. *Pour chaque coupe cohérente (périodique, avec la période* $\mathsf{ppcm}(E)$*) caracté-*

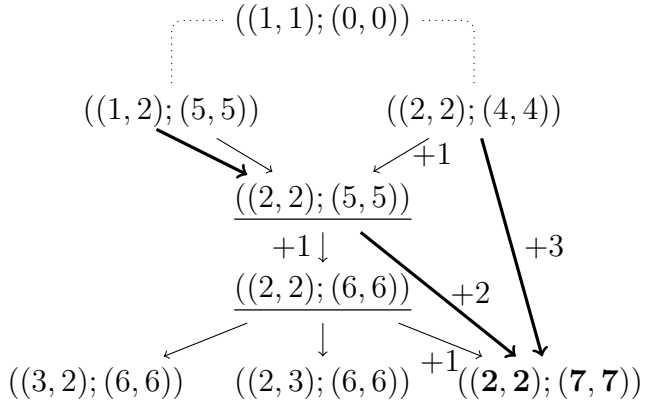

FIGURE 4.7 – Un fragment de dynamique originale et accélérée (omettant les valeurs de v) de l'Exemple 4.6. Les vecteurs des localités (l_1^i, l_2^j) sont notés (i, j). Les arcs gras correspondent aux étapes présentes dans la dynamique accélérée tandis que les arcs ordinaires correspondent aux étapes présentes dans la dynamique d'origine. Les coupes cohérentes dans la dynamique originale sont soulignées et celles de la dynamique accélérée sont en gras.

risé par les vecteurs $(\vec{l}, \vec{c})$ à l'instant t (mesuré depuis le début du système) dans la sémantique d'origine, il existe une coupe cohérente dans la sémantique accélérée pour le même vecteur de localités $\vec{l}$ à l'instant $t + \delta$, avec $\delta \in \mathbb{N}$.

Démonstration. Nous devons montrer qu'un tel δ existe et que $(\vec{l}, \vec{c'})$ est une coupe cohérente dans la sémantique accélérée, où $\vec{c'} = \vec{c} + \delta$.

Nous savons que chaque état accessible dans la dynamique accélérée est également accessible dans celle d'origine et que, pour chaque chemin existant entre deux états de la dynamique accélérée, il existe également au moins un chemin dans celle d'origine. Comme une coupe cohérente est un point de passage obligatoire (quand on néglige les valeurs de la variable v) dans la dynamique originale du système, le seul moyen de l'éviter dans la dynamique accélérée est d'avoir un nouvel arc depuis un état situé avant la coupe vers à un état situé après la coupe (par exemple, dans la Figure 4.7, la coupe originale $(2, 2)(5, 5)$ est accessible dans la sémantique accélérée, mais elle peut également être contournée par l'arc de $(2, 2)(4, 4)$ vers $(2, 2)(7, 7)$, il ne s'agit donc pas d'une coupe cohérente de la sémantique accélérée. Toutes les transitions et les réinitialisations présentes dans la dynamique accélérée étant également présentes dans la dynamique d'origine, seule une augmentation du temps (un saut jusqu'à la fin de la zone d'actions maximale suivante) peut offrir une telle possibilité. Par conséquent, il est suffisant de prouver que chaque fois qu'une augmentation de temps dans la dynamique accélérée passe d'un état s avant une coupe cohérente dans la dynamique originale à un état s' après cette coupe, l'état s' appartient à une coupe cohérente dans la dynamique accélérée.

Soit $t^- = \max\{k \cdot E_i - \mathsf{init}_i \leq t, k > 0, i \in \{1, \ldots, n\}\}$ l'instant où a lieu la dernière réinitialisation avant t (si cette réinitialisation n'existe pas, alors $t^- = 0$), et $t^+ = \min\{k \cdot E_i - \mathsf{init}_i \geq t, k > 0, i \in \{1, \ldots, n\}\}$ l'instant où a lieu de la première réinitialisation après t. Chaque agent étant un DAG, si un agent quitte une localité, il est impossible de rejoindre la même localité avant la prochaine réinitialisation de l'agent. En conséquence, dans l'intervalle $[t^-, t^+]$, un vecteur de localités $\vec{l}$, une fois quitté (c'est à dire en effectuant une transition depuis un état avec $\vec{l}$) ne peut plus être atteint à nouveau. Par conséquent, dans les deux sémantiques, dans l'intervalle $[t^-, t^+]$, il n'est pas possible d'entrer dans $\vec{l}$ strictement après t, ni de quitter $\vec{l}$ strictement avant t, étant donné que $\vec{l}$ doit être atteint au moment t dans chaque chemin (par la définition d'une coupe cohérente).

Par conséquent, dans la sémantique accélérée, on entre toujours dans $\vec{l}$ à $t' \leq t$. Si $t' < t$, comme $\vec{l}$ ne peut pas être quitté avant t, seul un intervalle temporel de d unités de temps est possible, tel que $t' + d \geq t$.

Si $t' = t$, soit la coupe cohérente est atteinte avec $\delta = 0$, soit seul un dépassement temporel de $d > 0$ unités de temps est possible, et conduit à la coupe cohérente. Par exemple, dans l'exemple de la Figure 4.7, si $t = 5$, il y a deux chemins possibles, soit $t' = 4$ et $d = 3$, ou $t' = 5$ et $d = 2$. Lorsque $t' = 5$, nous avons $t' = t$ et nous atteignons la coupe cohérente de la sémantique originale $(((2, 2); (5, 5)))$, qui n'est pas une coupe cohérente dans la sémantique accéléré puisqu'il existe maintenant un chemin qui ne l'atteint pas en passant par le nouvel arc étiqueté $+3$. Par conséquent, seul une augmentation de temps est possible à partir de $((2, 2); (5, 5))$, et nous sommes dans le cas où la coupe cohérente est atteinte après cette augmentation de temps.

L'augmentation du temps de d unités de temps correspond à d augmentations de 1 unité. Cette augmentation du temps passe par dessus un état qui appartient à une coupe cohérente dans la sémantique originale, et tous les chemins de cette coupe mènent à $t' + d$ à un ensemble d'états qui possèdent tous les mêmes vecteurs de localités et d'horloges, étant donné que la sémantique accélérée conduit toujours à la fin de la première zone d'actions maximale. Par définition, cet ensemble d'états est une coupe cohérente car ils partagent tous le même vecteur de localités $\vec{l}$ et le même vecteur d'horloges $\vec{c} + \delta$, avec $\delta = t' + d - t$. □

Si la coupe cohérente $(\vec{l}, \vec{c})$ survient avant la première réinitialisation, alors $t^- = 0$. Si elle se produit lorsqu'une réinitialisation est disponible, alors $t^- = t^+$ et nous avons $\delta = 0$. Le temps t' où $\vec{l}$ est atteint n'est pas nécessairement le même pour tous les chemins, mais l'augmentation du temps de d unités de temps mène toujours à la fin de la première zone d'actions maximale, de sorte que $t' + d = t + \delta$. On notera que la périodicité est conservée puisque la coupe cohérente $(\vec{l}, \vec{c'})$ est atteinte toutes les $\mathsf{ppcm}(E)$ unités de temps.

Exploration de l'espace d'états en couches La fonction $next_border(state, cuts)$, décrite dans l'Algorithme 4, prend un état $state$ et un ensemble non vide de coupes cohérentes $cuts$, et calcule, via une exploration en largeur d'abord, l'ensemble des successeurs en bordure de la couche suivante (autrement dit, la prochaine coupe cohérente rencontrée). Cette fonction s'applique à la fois à la sémantique d'origine et à la sémantique accélérée. Pour la décrire, nous introduisons la fonction $next_state(s)$, qui renvoie l'ensemble des successeurs de l'état s, et la fonction $is_cut(pre_s, s, cuts)$, qui est vrai si l'état s, successeur de l'état pre_s, fait partie d'une coupe dans $cuts$ Formellement, $is_cut(pre_s, s, cuts)$ est évaluée comme vraie si $s \in cuts$ ou $pre_s = (\vec{l}, \vec{c^-})$, $s = (\vec{l}, \vec{c^+})$ et $(\vec{l}, \vec{c}) \in cuts$ avec $\vec{c^-} < \vec{c} \leq \vec{c^+}$. L'algorithme est décrit avec le langage python : $list.add(e)$ ajoute l'élément e dans la file tandis que $list.pop()$ supprime le premier élément (il s'agit d'un comportement premier entré / premier sorti), $border$ et $exploring$ sont initialement vides et la condition de la boucle est vraie tant que $exploring$ est non vide.

Algorithm 4 $next_border(state, cuts)$

```
border[] {Ensemble des états à renvoyer}
exploring[] {File des états à explorer}
exploring.add(state)
while exploring do
  pre_s ← exploring.pop()
  successors ← next_state(pre_s)
  for all s ∈ successors do
    if is_cut(pre_s, s, cuts) then
      border.add(s) {Les états de la coupe sont ajoutés à border}
    else
      exploring.add(s) {Les autres états sont ajoutés à exploring}
    end if
  end for
end while
return border
```

Cet algorithme est utilisé de manière itérative dans une exploration en profondeur d'abord pour passer d'un état à l'un de ses successeurs appartenant à la bordure suivante. Une telle utilisation des couches permet de réduire le nombre de chemins explorés en détectant les diamants simples causés par l'ordre des transitions des agents concurrents.

4.2.2 Exploration basée sur les variables fortes et faibles

L'approche présentée ci-dessus ne permet pas de traiter des diamants s'étalant sur une distance temporelle plus longue que celle qui sépare deux bordures adjacentes. Par exemple, il est

possible que deux états s_1 et s_2 appartenant à la même bordure aient dans le futur un successeur commun s, comme illustré en Figure 4.5. Pour détecter ces diamants s'étendant sur plusieurs couches, et ainsi éviter de repasser plusieurs fois sur les mêmes chemins, il est intéressant d'effectuer l'exploration en largeur d'abord qui calcule les successeurs à la prochaine bordure pour l'ensemble $\{s_1, s_2\}$ au lieu de les prendre séparément. En général, il n'est pas évident de savoir quels états devraient être maintenus ensemble dans le calcul de la prochaine bordure. Pour réaliser un tel regroupement, il peut être intéressant d'exploiter les propriétés des applications cibles, tels que CAVS.

Une solution possible consiste à supposer que l'ensemble de valeurs $\mathcal{V} \stackrel{\text{df}}{=} V_w \times V_s$, où V_w (*weak*) est une partie moins importante de v et que V_s (*strong*) est plus importante, de sorte qu'il est peu probable que les états qui diffèrent sur la valuation de V_s aient un successeur commun, alors que ce n'est pas le cas pour V_w. Symétriquement, les états dont la valuation V_s est identique ont plus de chances d'avoir un successeur commun. Cela peut nous donner un critère pour regrouper les états et passer d'un ensemble d'états à l'ensemble de leurs successeurs à la bordure suivante. Le choix de V_s et de V_w dépend bien entendu du système et devrait être défini par un expert ou à l'aide d'un outil de simulation. Par exemple, les éléments auxquels une nouvelle valeur peut être affectée indépendamment de leur valeur actuelle peuvent être considérés comme faibles, tandis que les éléments dont la valeur change en fonction de leur valeur actuelle (par exemple, la position d'un objet en mouvement) peuvent être considérés comme forts.

La fonction $clustered_next_border(state_set, cuts)$, définie dans l'Algorithme 5 est une variante de $next_border()$, prenant un ensemble d'états et produisant un ensemble de sous-ensembles, qui sont des ensembles d'états pour lesquels les valeurs des variables dans V_s sont identiques. Elle est utilisée de la même manière que $next_border()$ pour explorer en profondeur l'espace d'états en couches, la seule différence étant qu'elle passe d'un sous-ensemble appartenant à une bordure à un sous-ensemble appartenant à la bordure suivante.

Formellement, on implémente une function $strong_equal(s1, s2)$ vérifiant l'équivalence sur V_s. Cette fonction est évaluée comme vraie si $s1$ et $s2$ ont une valuation identique sur V_s. A chaque fois qu'un état s appartenant à une coupe cohérente est détecté, on cherche si il existe un sous-ensemble dans $border$ tel que les états de ce sous-ensemble et s partagent une valuation identique sur V_s. Si un tel sous-ensemble n'existe pas, s est alors placé dans un nouveau sous-ensemble.

On notera que si $V_s = \emptyset$, une telle exploration équivaut à une exploration en largeur d'abord, car les états en bordure de couche sont toujours tous conservés dans le même sous-ensemble. Avec un tel algorithme, pour un espace d'états en couches d'un MAPT, on peut effectuer une exploration en profondeur «à la volée» puisqu'il n'est pas nécessaire de mémoriser les états

Algorithm 5 $clustered_next_border(state_set, cuts)$

```
border[][] {Ensemble des sous-ensembles d'états à renvoyer}
exploring[] {File des états à explorer}
state_added {Variable booléenne}
for all state ∈ state_set do
  exploring.add(state)
end for
while exploring do
  pre_s ← exploring.pop()
  successors ← next_state(pre_s)
  for all s ∈ successors do
    if is_cut(pre_s, s, cuts) then
      state_added ← False
      for all set ∈ border do
        if strong_equal(set[0], s) then
          set.add(s) {Les états de la coupe sont ajoutés à leur sous-ensemble}
          state_added ← true
        end if
      end for
      if ¬state_added then
        border.add([s]) {Création d'un nouveau sous-ensemble}
      end if
    else
      exploring.add(s) {Les autres états sont ajoutés à exploring}
    end if
  end for
end while
return border
```

explorés. Cela peut être utilisé pour rechercher efficacement des états accessibles spécifiques, et peut être accéléré par l'utilisation d'heuristiques qui choisissent les ensembles d'états à explorer en priorité.

4.2.3 Exploration «à la volée» d'un MAPT

Cette section est dédiée aux algorithmes d'exploration de préfixes finis de MAPTs : les états qui n'ont pas de successeurs dans le préfixe considéré seront appelés **finaux**. Les algorithmes sont désignés avec la syntaxe de la logique temporelle CTL. Cette logique temporelle étant censée explorer des chemins infinis, nous considérerons que chaque état final boucle sur lui même.

Nos algorithmes ont deux caractéristiques principales : ils fonctionnent "à la volée", ce qui signifie qu'ils ne mémorisent pas la totalité de l'espace d'états visité (mais seulement une partie de celui-ci), et qu'ils peuvent être ajustés à l'aide d'heuristiques définissant une priorité sur les chemins à explorer, ce qui permet d'accélérer de manière significative le temps de calcul si les états recherchés existent. Pour ce faire, nous nous appuyons sur l'algorithme $clustered_next_border()$ mentionné précédemment. Comme ils ne mémorisent pas tous les états qui ont été explorés, nous avons choisi de ne pas renvoyer de traces d'exécution, contrairement à ce qui est habituellement proposé par les outils standard de vérification de modèle par logique temporelle.

Nous avons formalisé des algorithmes pour les propriétés de base de CTL $EF\ p$ et $EG\ p$, qui signifient respectivement «un état accessible satisfait p» et «il existe un chemin où p est toujours vrai» . Il est possible d'appliquer la négation à toute propriété pour laquelle nous avons un algorithme, de sorte que nous pouvons aussi exprimer AFp et AGp, respectivement équivalentes à $\neg(EG\neg p)$ et $\neg(EF\neg p)$.

L'algorithme pour $EF\ p$ consiste, à partir d'une pile contenant l'état initial, à prendre le premier élément s de la pile, à le retourner comme résultat si p est vrai sur s, et sinon à ajouter le résultat de $clustered_next_border(s, cuts)$ à la pile. L'algorithme continue récursivement jusqu'à atteindre p ou qu'il n'y ai plus d'états à explorer dans le préfixe fini considéré. De plus, nous retournons $true$ si p est atteint entre deux bordures, c'est à dire, dans $clustered_next_border()$. L'algorithme pour $EG\ p$ fonctionne de la même manière, mais l'état s est renvoyé si p est vrai sur s et si s est final, et $clustered_next_border()$ n'est ajouté à la pile que si p est vrai sur s. De plus, les états où p n'est pas vrai ne sont pas explorés dans $clustered_next_border()$. De cette façon, seuls les états où p est vrai sont explorés.

Nous pouvons également définir des algorithmes pour les requêtes CTL imbriquées construites avec des opérateurs logiques binaires. Nous allons considérer deux d'entre

elles : $EF(p \wedge EFq)$, ce qui signifie que «un état accessible satisfait p et, à partir de cet état, un état accessible satisfait q», et $EF(p \wedge EGq)$, qui signifie que «un état accessible satisfait p et à partir de cet état, il existe un chemin où q est toujours vrai». On peut remarquer que l'opérateur *leads to* ($--> $) utilisé dans l'outil UPPAAL suit l'équivalence : $p --> q <=> AG(\neg p \vee AFq) <=> \neg EF(p \wedge EG\neg q)$. Bien que seules ces deux requêtes soient définies ici, tout type de requête CTL imbriquée peut être implémenté.

Ces requêtes imbriquées sont implémentées à l'aide d'une fonction de marquage (un indicateur booléen). $EF(p \wedge EFq)$ est implémenté comme suit. Chaque fois que p est vrai sur un état, l'état est marqué. Chaque fois qu'un état est marqué, tous ses successeurs sont marqués. A partir d'une pile contenant l'état initial, le premier élément s de la pile est renvoyé si q est vrai sur s et que s est marqué. Sinon, le résultat de $clustered_next_border(s, cuts)$ est ajouté à la pile. Le même processus de marquage est effectué entre deux bordures, c'est à dire dans $clustered_next_border()$. Nous continuons récursivement jusqu'à ce qu'un état valide la propriété ou qu'il n'y ait plus d'états à explorer. En ce qui concerne $EF(p \wedge EGq)$, les états sont marqués chaque fois que p et q sont vrais ou que l'état est le successeur d'un état marqué et que q est vrai. Si un état final marqué est atteint, il valide la propriété et est renvoyé. De nouveau, le même processus de marquage est effectué dans $clustered_next_border()$.

Les algorithmes correspondant aux requêtes présentées ici se trouvent en Annexe C.

4.3 Comparaison de performances

Dans cette section, nous illustrons les performances de nos algorithmes d'exploration. Pour ce faire, nous utilisons des MAPTs représentant des systèmes de véhicules communicants autonomes, pour lesquels les deux contraintes 1 et 2 sont satisfaites. La première permet d'utiliser l'accélération, ce qui réduit la taille de l'espace d'états ainsi que le nombre de diamants. La seconde garantit que l'espace d'état est un DAG. En conséquence de 2, l'espace d'états est infini, à cause de la partie X de $\mathcal{V}$. Dans les études de cas suivantes, les positions longitudinales des véhicules sur la route joueront ce rôle. La route que nous observons est techniquement infinie, mais comme nous ne nous intéressons qu'à l'analyse d'une partie de celle-ci, nous pouvons limiter l'exploration à une valeur fixe de X. Le système converge donc vers une limite qui, une fois atteinte, est considérée comme un état final.

Dans ce qui suit, nous comparons d'abord le temps d'exploration obtenu avec ou sans accélération. Ensuite, nous discutons des avantages et des inconvénients de l'utilisation de divers types d'explorations basées sur les couches. Dans la troisième partie de cette section, nous fournissons quelques heuristiques et expérimentons afin d'observer leurs performances. Enfin, nous comparons cette méthode avec le cadre logiciel VERIFCAR présenté dans le chapitre précédent

et fournissons un moyen de traiter efficacement l'analyse du comportement du système.

Trois modèles avec différentes tailles d'espaces d'état sont utilisés pour les démonstrations. Il s'agit du cas de base, de la variante avec infrastructure et de la variante avec négociation du scénario 2 (Init. variant 1) introduit dans la sous-section 3.3.3. Ces modèles ont été implémentés en tant que MAPTs. On les désignera respectivement par Modèle 1, Modèle 2 et Modèle 3. Les expériences ont été réalisées en implémentant les modèles avec l'outil de modélisation de réseau de Petri de haut niveau ZINC et en utilisant sa bibliothèque pour implémenter nos algorithmes d'exploration. Les sources des modèles et algorithmes utilisés sont disponibles sur [74].

4.3.1 Efficacité de la dynamique accélérée.

Une première exploration en largeur de l'espace d'états sur chacun des trois modèles a été réalisée en utilisant à la fois la sémantique d'origine et la sémantique accélérée. La Table 4.1 fournit pour chaque modèle le nombre d'états dans son espace d'états ainsi que le temps d'une exploration complète (TEC) pour les deux sémantiques et pour UPPAAL. On peut voir comment l'utilisation de la sémantique accélérée réduit le temps d'exploration. Par conséquent, la sémantique accélérée a été utilisée dans toutes les expériences suivantes.

	Modèle 1	Modèle 2	Modèle 3
TEC sémantique originale (s)	14.5	144	574
TEC sémantique accélérée (s)	10.8	52.1	420
TEC UPPAAL (s)	5	27	379
Taille de l'espace d'états	7751	52732	285944

TABLE 4.1 – Comparaison des temps d'explorations complètes avec un algorithme en largeur d'abord sur les différents modèles avec ou sans accélération.

Il est intéressant de noter que la principale amélioration de la sémantique accélérée par rapport à la sémantique originale consiste à n'explorer qu'un seul état de chaque zone d'actions. Ainsi, plus un système comporte de transitions avec de larges intervalles de temps non déterministes, plus le gain de temps fourni par la sémantique accélérée est important. Ici, les intervalles de temps non déterministes présents dans les Modèles 1 et 3 sont assez courts (il s'agit de l'action de communication des véhicules, qui permet 2 valeurs d'horloges), de sorte que le nombre de chemins ignorés dans la dynamique accélérée n'est pas très important. D'autre part, le modèle 2 présente une transition avec un intervalle de temps non déterministe plus large (il s'agit de l'action de communication du terminal, qui permet 5 valeurs d'horloges), expliquant pourquoi la différence entre les deux sémantiques est plus prononcée pour ce modèle. On peut s'attendre à ce que la sémantique accélérée soit encore plus utile lorsqu'on utilise des modèles

similaires à celui décrit en Figure 4.6.

4.3.2 Efficacité de l'exploration par couches.

Ici, nous comparons, pour plusieurs algorithmes d'exploration, le temps d'exploration total et le temps pour atteindre la première occurrence d'un état final. Ils sont explorés en largeur d'abord, en profondeur d'abord sans couches et en profondeur d'abord avec couches (avec et sans l'utilisation de variables fortes / faibles).

Algorithme d'exploration	Temps d'exploration totale (s)			Première occurrence d'un état final (s)		
	Modèle 1	Modèle 2	Modèle 3	Modèle 1	Modèle 2	Modèle 3
Largeur d'abord	10.8	52.1	420	10.7	52	419.9
Prof. sans couches	∞	∞	∞	3.3	4.6	3.9
Prof. avec couches ($V_w = \emptyset$)	11	∞	∞	4.5	6.6	4.1
Prof. avec couches (V_w petit)	11	250	2015	4.5	7.4	6
Prof. avec à couches (V_w grand)	11	71	667	4.5	14.4	7.2

TABLE 4.2 – Comparaison du temps d'exploration total et du temps nécessaire pour atteindre la première occurrence d'un état final pour les algorithmes d'exploration en largeur d'abord, en profondeur d'abord avec et sans couches et avec ou sans utilisation de variables faibles. ∞ signifie que l'exploration a été arrêtée après 50 heures de calculs sans résultat.

La table 4.2 montre les résultats. On peut voir que l'algorithme en largeur d'abord a le meilleur temps d'exploration, mais le temps nécessaire pour atteindre un état final correspond à son temps d'exploration total. D'un autre côté, l'algorithme en profondeur d'abord standard est le plus rapide pour atteindre un état final, mais il n'explore pas complètement l'espace d'états, même après 50 heures de calcul. Les résultats pour le modèle 1 montrent que, tant que l'approche par couche est utilisée, le temps total d'exploration est très proche de celui de l'algorithme en largeur d'abord. Cela indique qu'il n'y a presque pas de diamants couvrant plusieurs couches, ce qui signifie que différents états appartenant à une bordure entre deux couches ne partagent presque jamais un successeur commun. Pour cette raison, l'utilisation de variables faibles n'a aucun effet. Bien que ce cas soit assez simple, il met clairement en évidence l'avantage de l'exploration par couches : sans pratiquement aucune augmentation du temps total d'exploration, il est possible d'atteindre un état final beaucoup plus rapidement.

Ceci est également vrai pour les modèles 2 et 3, bien qu'ils aient des espaces d'états beaucoup plus complexes. En effet, dans ces cas, l'algorithme basé sur les couches qui ne s'appuie pas sur des variables faibles pour agréger des états n'est pas en mesure d'explorer tout l'espace d'états, même après 50 heures de calcul. On peut aussi remarquer que même avec un petit nombre [3] bien choisi de variables faibles (6 sur 39), il est possible d'explorer pleinement

3. Il s'agit ici des données relatives à la négociation.

l'espace d'états. Dans les deux cas, l'exploration est environ cinq fois plus longue que le temps d'exploration du premier algorithme en largeur d'abord. Lorsqu'on utilise un grand nombre [4] de variables faibles (30 sur 39), l'exploration est beaucoup plus courte (environ $1,5$ fois le temps de l'algorithme en largeur d'abord). On peut noter cependant que plus V_w est grand, plus il faut de temps pour atteindre un état final. En effet, comme les états sont agrégés couche par couche, une valeur trop grande de V_w donnerait lieu à une exploration similaire à une largeur d'abord, dans laquelle tous les états sont conservés ensemble et les états finaux ne sont atteints qu'à la fin du calcul.

Dans les expériences qui suivent, les algorithmes de profondeur d'abord sont basés sur l'utilisation des couches avec V_w non vide.

4.3.3 Heuristiques

Les algorithmes d'exploration basés sur les couches permettent l'utilisation d'heuristiques. Ces heuristiques guident l'exploration, en choisissant parmi tous les états non explorés, celui qui mènera probablement à un état satisfaisant la propriété vérifiée. L'heuristique que nous utilisons consiste à associer un poids à chaque état. Lorsqu'un nouvel état est découvert, il est placé dans une liste d'états à explorer classée par poids. La liste des états à explorer est triée par poids croissant ou décroissant, en fonction de la propriété à vérifier. Le poids est une prédiction de la distance entre l'état actuel et un état satisfaisant la propriété. Le prochain état à explorer est le dernier de la liste, c'est-à-dire le plus élevé si les poids sont croissants ou le plus faible si ils sont décroissants.

Par conséquent, une propriété peut être associée à une heuristique qui prend un état en tant qu'entrée et renvoie un poids en tant que sortie. Ci-dessous se trouve une liste des heuristiques que nous avons utilisées à des fins expérimentales, ainsi que la propriété à laquelle chacune est associée :

1. $distance_vh_1_vh_2$: renvoie la position longitudinale du véhicule vh_1 moins celle du véhicule vh_2. Elle peut être utilisé avec des poids triés par ordre croissant et la propriété $EF\ arrival_vh_1_before_vh_2$, où $arrival_vh_1_before_vh_2$ est vraie dans un état où le véhicule vh_1 atteint la fin de la portion de route avant le véhicule vh_2. L'idée est de vérifier en priorité les états où vh_1 est le plus en avance sur vh_2.
2. $estimated_travel_time_vh$: renvoie le temps parcouru depuis l'état initial plus le temps estimé pour atteindre la fin de la portion de route, en supposant que la vitesse actuelle soit maintenue. Elle peut être utilisé avec des poids triés par ordre croissant et la propriété $EF\ travel_time_vh_sup_n$, où $travel_time_vh \geq n$ est vraie dans un état si vh a atteint

4. Il s'agit ici de toutes les variables qui représentent pas la position des véhicules.

la fin de la portion de route en n unités de temps ou moins. L'idée est de vérifier en priorité les états où vh devrait atteindre l'extrémité de la route avec le temps le plus court.

3. $time_to_overtake_vh_1_vh_2$: est le moment où les deux véhicules atteignent la même position longitudinale s'ils conservent leur vitesse actuelle. Elle peut être utilisé avec des poids triés par ordre décroissant et la propriété $EF\ ttc_vh_1_vh_2 \leq n$, où $ttc_vh_1_vh_2$ est la valeur de l'indicateur de temps avant collision entre vh_1 et vh_2 (le délai avant une collision entre les deux véhicules s'ils conservent leur vitesse actuelle), et n est une valeur de temps avant collision. L'idée est de vérifier en priorité les états où l'un des véhicules se rapproche de l'autre avec une vitesse supérieure.

Ces heuristiques ont été utilisées sur le modèle 3, les résultats étant présentés dans la Table 4.3. Pour rappel, le scénario du modèle 3 considère que trois véhicules sont positionnés comme indiqué sur la figure 4.8 sur une portion de route à deux voies de 500 m de long, avec une voie de jonction. Initialement, le véhicule A se trouve sur la voie de droite à la position 0 m avec une vitesse de 30 m / s, le véhicule B sur la voie de gauche à la position 30 m avec une vitesse de 15 m / s et le véhicule C sur la voie de jonction à position 40 m avec une vitesse de 20 m / s. Ils ont tous comme objectif d'être sur la voie de droite à la fin de la portion de route.

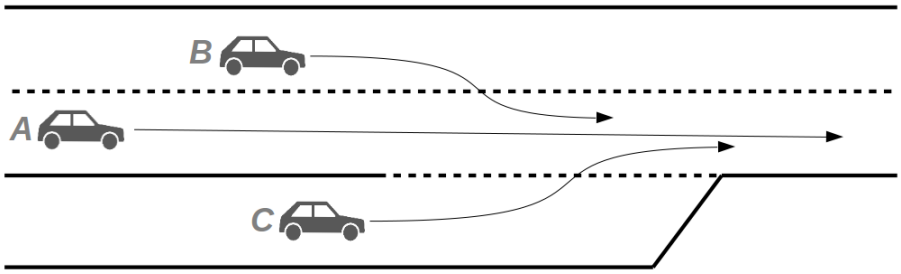

FIGURE 4.8 – Positions initiales et trajectoires possibles des véhicules autonomes pour le scénario du modèle 3.

Algorithme d'exploration	$EF\ arrival_B_before_A$	$EF\ travel_time_A \geq 15.9$	$EF\ ttc_A_C \leq 1.14$	$EF\ ttc_A_B \leq 0$
Largeur d'abord	416	427	292	95
Prof. sans heuristiques	234—357	167—340	247—547	277—483
Prof. avec heuristiques	131	149	103	13

TABLE 4.3 – Comparaison du temps d'accessibilité pour les algorithmes d'exploration en largeur d'abord et en profondeur d'abord avec et sans heuristiques. Comme la profondeur d'abord sans heuristique est non déterministe, les deux valeurs correspondent aux exécutions les plus rapides et les plus lentes obtenues pour chaque requête (cinq exécutions étant effectuées à chaque fois).

Les deux premières requêtes ne peuvent être vraies que dans un état final (la couche la plus profonde). C'est pourquoi le temps d'accessibilité est proche du temps d'exploration complet

avec l'algorithme en largeur d'abord. En général, le temps d'accessibilité en largeur d'abord dépend de la profondeur du premier état qui satisfait la propriété. On peut observer que pour la quatrième requête, l'état est effectivement atteint à une profondeur inférieure, ce qui se répercute sur le temps d'accessibilité.

Comme l'algorithme de profondeur d'abord sans heuristique choisit au hasard les chemins à explorer en premier, le temps d'accessibilité varie. Le nombre d'états dans l'ensemble de l'espace d'états qui satisfait la propriété a donc une incidence sur le temps d'accessibilité moyen avec cet algorithme. En d'autres termes, lorsqu'il y a plus de possibilités de vérifier la propriété, le temps moyen nécessaire est plus court. Comme nous ne voulons pas compter sur la chance, cela n'est pas satisfaisant.

En revanche, l'algorithme de profondeur d'abord avec heuristique explore les états dans un ordre donné (en fonction de leur poids) et par conséquent, le temps d'accessibilité est toujours identique. Les heuristiques que nous avons utilisées pourraient bien sûr être modifiées et améliorées, mais elles sont suffisantes pour montrer une augmentation significative du temps d'accessibilité. Même pour la quatrième requête, où la largeur d'abord est plus rapide que l'algorithme de profondeur d'abord, l'heuristique permet d'identifier rapidement l'état qui satisfait la propriété.

4.3.4 Comparaison avec VERIFCAR

Nous allons maintenant comparer le temps d'atteignabilité obtenu sur le modèle 3 avec les algorithmes d'exploration en profondeur d'abord avec heuristiques avec ceux obtenus par le cadre logiciel VERIFCAR sur le même modèle. Pour ce faire, on applique les requêtes qui ont été utilisées avec VERIFCAR pour produire les résultats de la Table 3.4 du chapitre précédent, à savoir l'ordre d'arrivée, le temps de trajet et le TTC pour la décision avec négociation sur la seconde variante d'initialisation du Scénario 2. Nous avons observé que UPPAAL construisait d'abord l'espace d'états en environ 106 s, puis pouvait répondre presque instantanément si un état recherché existait. Il peut donc répondre à plusieurs requêtes après la construction de l'espace d'états, contrairement à nos algorithmes d'exploration dynamiques basés sur des méthodes heuristiques, qui doivent explorer l'espace d'états depuis le début pour chaque nouvelle requête. Cependant, la plupart des états recherchés pouvaient être atteints facilement avec les algorithmes d'exploration, et le calcul ne prend que 4 secondes environ. Les requêtes qui ont été utilisées dans Table 4.3 sont celles où les états étaient plus difficiles à atteindre.

Ces résultats indiquent qu'en termes de temps d'accessibilité global, nos algorithmes peuvent tenir la comparaison avec UPPAAL. En revanche, si la propriété atteignabilité n'est pas vraie, ils sont plus lents que UPPAAL, qui, selon le type de requête, prend entre 34 et

370 secondes, ce qui correspond au temps d'exploration total pour le modèle 3 avec UPPAAL. Comme mentionné précédemment, le temps d'exploration total de l'espace d'états avec les algorithmes de profondeur d'abord sur ce modèle est dans notre cas de 667 secondes. Ce n'est pas une surprise puisque UPPAAL est un outil mature utilisant de nombreuses abstractions efficaces. Cependant, il est limité en termes d'expressivité, au moins sur deux aspects qui nous intéressent. Tout d'abord, il n'est pas possible de vérifier directement les limites d'un indicateur donné, et ces limites devraient être obtenues par dichotomie, nécessitant plusieurs cycles pour chaque indicateur, comme proposé dans la méthodologie de VERIFCAR. Deuxièmement, il est limité à un sous-ensemble de CTL (acceptant principalement les requêtes non imbriquées). Nos algorithmes n'ont pas de telles restrictions.

En effet, il est possible de faire une exploration totale de l'espace d'états, tout en conservant pour chaque état les valeurs les plus basses et les plus élevées atteintes sur le chemin menant à cet état, pour un ensemble donné d'indicateurs. Ainsi, toutes les informations nécessaires à l'analyse du comportement du système peuvent être obtenues après une seule exploration totale. Cela est effectué comme une exploration standard en largeur d'abord (stockage des états dans un fichier), à la différence que chaque état est associé à un ensemble de couples (min, max), constituant les bornes des indicateurs considérés sur le chemin exploré. Chaque fois qu'un état est exploré, la valeur de chaque indicateur est calculée et il réécrit min s'il est plus petit et max s'il est plus grand. Ainsi, chaque état s contient, pour chaque indicateur, les valeurs les plus petites et les plus élevées qui existent sur le chemin allant de l'état initial à s. Comme plusieurs chemins peuvent mener à s, nous considérerons que s atteint par le chemin $P1$ et s atteint par le chemin $P2$ ne sont équivalents que si leur ensemble d'indicateurs respectif sont également équivalent. De cette façon, il est possible d'avoir plusieurs versions d'un même état, mais avec des valeurs d'indicateurs différentes. Si l'on est intéressé par l'accessibilité des états (par exemple, savoir si un indicateur est égal à une valeur), cela peut facilement être réalisé de la même manière, en ajoutant des variables booléennes dans l'ensemble des indicateurs. À la fin de l'exploration, nous obtenons ainsi l'ensemble de tous les états finaux, ainsi que toutes les informations qui ont été acheminées sur leurs chemins respectifs. Il contient donc toutes les informations nécessaires pour analyser finement le système. Dans le cas du modèle 3, l'obtention de l'ordre d'arrivée ainsi que les bornes du temps de trajet et du pire TTC prennent 708 secondes. En comparaison, le temps nécessaire à UPPAAL pour obtenir les mêmes informations avec la procédure de dichotomie est de 3553 secondes.

De plus, la forme de DAG de l'espace d'états nous permet d'implémenter tout type de requêtes CTL. A des fins expérimentales, nous avons utilisé la requête $EF(p \wedge EGq)$, qui est la négation de l'opérateur *leads to* $p --> \neg q$ (le seul opérateur imbriqué disponible dans UPPAAL), ainsi que la requête $EF(p \wedge EFq)$, qui ne peut pas être exprimée dans UPPAAL.

Dans le chapitre précédent, $arrival_C_before_A --> arrival_B_before_A$) a été utilisé et a atteint un état invalidant la propriété en 110 secondes. Sa négation EF ($arrival_C_before_A \wedge EG\ \neg arrival_B_before_A$) peut être exprimée ici, et trouve l'état validant la propriété en environ 10 secondes. Les propriétés EF ($ttc_A_B \leq 1 \wedge EF\ arrival_A_before_C$ et EF ($ttc_A_B \leq 1 \wedge EF\ arrival_A_before_B$), qui ne peuvent pas être exprimées dans UPPAAL, peuvent être résolues avec nos algorithmes. La première exprime la possibilité pour le véhicule A d'arriver devant le véhicule C après qu'une situation dangereuse se soit produite, entraînant un temps avant collision inférieur à 1 seconde. La seconde est similaire pour les véhicules A et B dans les mêmes conditions. La première requête est fausse et il faut explorer tout l'espace d'états pour donner une réponse (donc en 680 secondes), tandis que la seconde est vraie et on trouve un état satisfaisant la propriété en 10 secondes environ.

Enfin, il convient de mentionner qu'une discrétisation est nécessaire pour UPPAAL et que, par conséquent, des approximations peuvent être obligatoires dans certains cas, entraînant une perte de précision et de réalisme. Outre une meilleure expressivité, le processus de vérification sur les MAPTs présenté dans ce chapitre ne nécessite pas d'approximations, ce qui permet d'atteindre un niveau de réalisme supérieur.

La Table 4.4 fait la synthèse des comparaisons effectuées. Les deux premières requêtes de la table sont évaluées comme vraies. La première étant une propriété sur tous les chemins, elle nécessite d'explorer l'ensemble de l'espace d'états afin de constater qu'il n'existe pas de contre exemple. À l'inverse, la seconde est validée dès lors qu'un état est trouvé qui satisfait la formule booléenne $arrival_C_before_B$. Les trois autres requêtes sont les requêtes imbriquées mentionnées précédemment.

Requête	MAPTs	UPPAAL
$AF\ travel_time_A \geq 13.0$	680	365
$EF\ arrival_C_before_B$	4	106
$arrival_C_before_A --> arrival_B_before_A$	13	110
$EF\ (ttc_A_B \leq 1 \wedge EF\ arrival_A_before_C)$	13	_
$EF\ (ttc_A_B \leq 1 \wedge EF\ arrival_A_before_B)$	680	_

TABLE 4.4 – Comparaison des temps de calculs (en secondes) pour l'évaluation d'une requête donnée entre UPPAAL et les algorithmes des MAPTs.

4.4 Résumé du chapitre

Dans ce chapitre, nous exploitons les particularités des modèles de véhicules autonomes communicants que nous avons formalisés précédemment afin de proposer des algorithmes d'ex-

ploration spécifiques. Ils permettent une plus grande expressivité que les algorithmes d'UPPAAL et peuvent être plus efficaces sur certains problèmes d'accessibilité. Pour ce faire on présente un nouveau formalisme, G-MAPT, qui est un modèle composé de plusieurs agents périodiques (mais acycliques sur chaque période) pouvant mettre à jour des variables partagées. La sémantique d'un G-MAPT est un système de transitions, où les états sont composés d'un ensemble de localités (une par agent), d'un ensemble de valeurs discrètes d'horloge (une par agent) et des valeurs (réelles) des variables partagées. Le changement d'état est possible par des franchissements de transition, qui modifient la localité d'un agent et potentiellement les variables partagées, l'augmentation unitaire du temps, où des réinitialisations d'horloge à chaque fin de cycle d'un agent. On montre qu'il est possible de transformer un G-MAPT en un automate temporisé étendu avec des fonctions ou en un réseau de Petri de haut niveau ayant un comportement similaire.

Les MAPTs (*Multi-Agent with timed Periodic Tasks*) sont un sous-ensemble des G-MAPTs qui ont la particularité de ne présenter aucun blocage et d'avoir un comportement global acyclique. Ces contraintes permettent la définition d'une sémantique accélérée, intéressante pour la relation de causalité entre les actions, et l'exploration de l'espace d'états selon un découpage en couches. La sémantique accélérée consiste à se déplacer temporellement à partir d'un état dans des zones d'actions, c'est à dire des intervalles de temps où les mêmes transitions et réinitialisations sont franchissables. De plus, on propose une optimisation consistant à ne pas explorer toutes les zones, ce qui a pour conséquence de supprimer des chemins ou de réduire leur taille sans pour autant perdre des comportements possibles en terme de séquence de transitions.

Afin d'explorer efficacement le système en profondeur d'abord de manière dynamique, c'est à dire sans avoir besoin de conserver en mémoire les états explorés précédemment, il est nécessaire de détecter les diamants dans l'espace d'états. Le découpage en couches de l'espace d'états réponds à ce problème en permettant d'explorer le système en profondeur d'abord de couches en couches, chaque couche étant elle même explorée en largeur d'abord. Une couche est délimitée par une coupe de l'espace d'états telle que tous les états de la coupe partagent les mêmes ensembles de localités et de valeurs d'horloges, et se situent à la même distance temporelle de l'état initial. De telles couches se comportent comme des points de passage obligatoires dans la dynamique du système qui vont permettre de regrouper ensemble les états identiques. Une variante possible consiste à explorer simultanément plusieurs couches lorsque ces dernières sont susceptibles de contenir des états identiques.

Sur la base de cette exploration par couches sont développés des algorithmes d'exploration de préfixes finis de MAPTs. Ces algorithmes calculent des propriétés exprimables dans la logique temporelle CTL. Ils fonctionnent "à la volée", ce qui signifie qu'ils ne mémorisent pas la totalité de l'espace d'états visité. De plus, ils peuvent être ajustés à l'aide d'heuristiques définis-

sant une priorité sur les chemins à explorer, ce qui permet d'accélérer de manière significative le temps de calcul si les états recherchés existent. L'expérimentation à l'aide du compilateur ZINC met en évidence l'efficacité de ces algorithmes et de la sémantique accélérée. Si cette méthode a l'inconvénient de ne pas renvoyer de traces d'exécutions, elle permet une plus grande expressivité que le cadre logiciel VERIFCAR, tout en obtenant des temps de calculs comparables pour trouver des états accessibles.

Chapitre 5

Approche combinée avec la simulation

Sommaire

5.1 Choix de l'outil de simulation 113
5.2 Etude d'un modèle de suivi de voiture : IDM 115
5.3 Comparaisons des valeurs des indicateurs 116
5.4 Application de la méthode 118
5.5 Résumé du chapitre 120

Dans ce chapitre, nous décrivons une méthode basée sur la combinaison des outils de vérification de modèle et méthodes décrites précédemment avec une approche de simulation. Notre objectif est de tirer parti de leurs avantages respectifs, toujours dans le cadre de l'analyse comportementale de véhicules autonomes. L'idée est de confirmer que le niveau de réalisme du modèle est suffisant, et donc que les requêtes de logique temporelle faites sur celui-ci retournent des résultats cohérents avec le réel. Cela est fait par une exécution dans l'outil de simulation des traces trouvées lors de la vérifications. Le niveau d'abstraction doit donc être choisi avec soin de manière à garantir une représentation convaincante du comportement des véhicules dans la simulation. En raison de l'explosion de l'espace d'états, le même niveau d'abstraction ne peut pas être utilisé dans la vérification des modèles dans le cadre des CAVs Etant donné que des abstractions supplémentaires sont nécessaires pour effectuer des analyses via *model checking*, il peut en résulter des écarts par rapport à la simulation, de sorte que nous devons vérifier que les résultats obtenus sont comparativement similaires.

Notre méthode est esquissée dans la figure 5.1. Le premier pas (étape 0 de la figure) consiste à produire des données à l'aide de simulation par ordinateur. Sur la base de ces données, un expert humain émet des hypothèses sur le comportement des véhicules autonomes pour une première situation. Ce pourrait être par exemple : tous les véhicules sont en sécurité à tout moment.

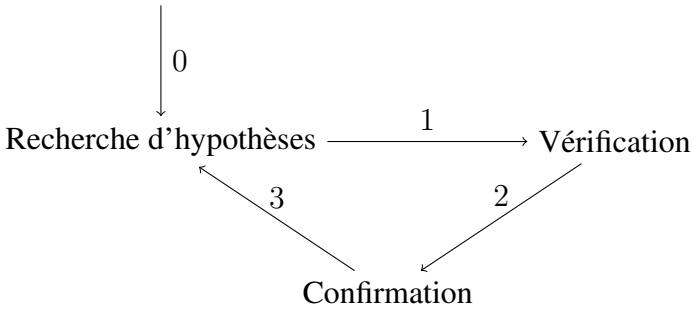

FIGURE 5.1 – Étapes de la méthode.

Ce type de requête nécessite une vérification exhaustive dans le cas d'une évolution non déterministe du système. L'étape 1 consiste à exprimer la requête dans la logique temporelle [1] et à la vérifier exhaustivement sur le modèle. Si le *model checking* permet d'identifier une exécution menant à un contre-exemple, l'exécution correspondante est ensuite confirmée par simulation, afin de s'assurer qu'il ne s'agit pas d'un faux positif dû aux abstractions (étape 2). Dans le cas où des contre-exemples invalidants l'hypothèse sont confirmés, le processus peut continuer avec un nouveau, grâce à l'enrichissement des données à travers le processus de vérification (étape 3). Plus précisément, cette méthode consiste à ajouter des exécutions au jeu de données, qui apportent des informations critiques permettant de comprendre le comportement du véhicule et ses causes.

Pour illustrer notre méthode, nous proposons une étude de cas, à savoir un modèle de suivi de voiture (*car-following model*) dans une situation donnée, ce qui ne pose pas de difficulté pour pour la simulation, mais où le *model checking* doit faire face à la possibilité d'une accumulation d'erreur due à la nécessaire discrétisation des équations de nombres réels. Un modèle de suivi de voiture est une équation mathématique définissant l'accélération du véhicule par rapport à la dynamique du véhicule qui le précède, tels que Gipps [75] ou IDM [12].

En particulier, nous nous intéressons aux questions suivantes :

— Quelles sont les conditions nécessaires pour qu'un véhicule se trouve dans une situation dangereuse ?

— Existe-t-il des exécutions dans lesquelles un véhicule est toujours en sécurité ?

— Est-il possible d'arrêter la voiture sans dépasser un seuil de décélération ?

Dans ce chapitre, nous présentons d'abord IDM, un algorithme décisionnel pour les véhicules autonomes, ainsi que les indicateurs que l'ont utilisera pour l'analyse comportementale. Pour plus de confiance dans nos résultats, une comparaison entre les outils de simulation et de

1. La traduction des propriétés dans la logique temporelle peut être partiellement automatisée à l'aide d'un ensemble prédéfini de requêtes.

model checking est développée afin de trouver des indicateurs non impactés ou peu impactés par les abstractions utilisées. Enfin, nous utilisons notre méthode pour vérifier des propriétés d'IDM sur certains scénarios non déterministes.

5.1 Choix de l'outil de simulation

Dans notre analyse combinée, nous utilisons une version légèrement modifiée (sans communication ni mouvement latéral) du modèle utilisé dans VERIFCAR. C'est donc l'outil de vérification UPPAAL qui a été utilisé et les propriétés sont exprimées dans le sous-ensemble de CTL reconnu par UPPAAL.

L'environnement considéré dans l'étude de cas reprend les caractéristiques du système donné en section 1.2 du premier chapitre : nous supposons que chaque véhicule connaît la position et la vitesse exacte du véhicule qui le précède chaque fois qu'une décision est prise, ce qui signifie que les informations fournies par les capteurs sont considérées comme parfaites. Cependant, l'enjeu de cette étude de cas est d'exprimer la fonction continue d'une équation de suivi de voiture malgré la limitation de l'outil de vérification aux calculs discrets, tout en atteignant une précision satisfaisante et un espace d'états suffisamment petit pour pouvoir être analysé dans un délai raisonnable. Pour ce faire, nous avons mis à l'échelle les variables avant la division, mis en œuvre les fonctions d'arrondi, de puissance et de racine carrée (qui ne sont pas prises en charge par UPPAAL) et avons judicieusement choisi la granularité des variables utilisées dans le modèle. Nous avons obtenu un modèle satisfaisant les exigences de précision et qui peut être analysé par un vérificateur de modèle en (au plus) quelques minutes pour quelques véhicules sur une portion de route de quelques centaines de mètres.

Simuler un système de CAVs peux se résoudre par différentes approches. La simulation est une méthode générale dans laquelle on construit d'abord un modèle pour représenter plusieurs aspects d'un système (environnement, comportement, interactions, phénomènes physiques, ...). Elle consiste en une évaluation de la dynamique du modèle dans le temps. Dans le contexte des véhicules, cela permet de reproduire leur physique et leurs interactions avec l'environnement.

Un niveau d'abstraction doit être défini de manière à s'approcher de la réalité en fonction de l'observation souhaitée. Différentes approches ont été développées pour la simulation de véhicules autonomes, parmi lesquelles on pourrait distinguer trois familles principales.

Une première approche consiste à reproduire de manière réaliste le comportement des véhicules en reproduisant à la perfection les lois de la physique ainsi que leurs paramètres spécifiques (tels que l'inertie, l'impulsion ou le frottement). La simulation peut ensuite évaluer divers aspects, tels que l'étude d'une trajectoire précise [76] ou, dans le contexte des véhicules

communicants, une étude de la fiabilité et de l'intégrité des informations transmises entre les véhicules [77] .

Une deuxième approche est axée sur le trafic en général et son évolution dans le temps sous trois angles [20] : macroscopique, microscopique et mésoscopique. En particulier, dans le cas des études microscopiques, où les véhicules sont manipulés individuellement dans une petite zone, cette approche permet de représenter la composante longitudinale du mouvement des véhicules grâce à des modèles de suivi de voiture, dont la plupart sont garantis sans collision (tels que IDM) et la composante latérale grâce aux modèles de changement de voie (comme MOBIL [78]).

Enfin, la troisième approche est orientée agent dans le sens où un véhicule est assimilé à un agent qui réagit en fonction de la perception de son environnement [79]. Chaque agent a également la possibilité de communiquer avec d'autres agents ou avec son environnement pour échanger des informations ou négocier une décision à prendre dans le futur.

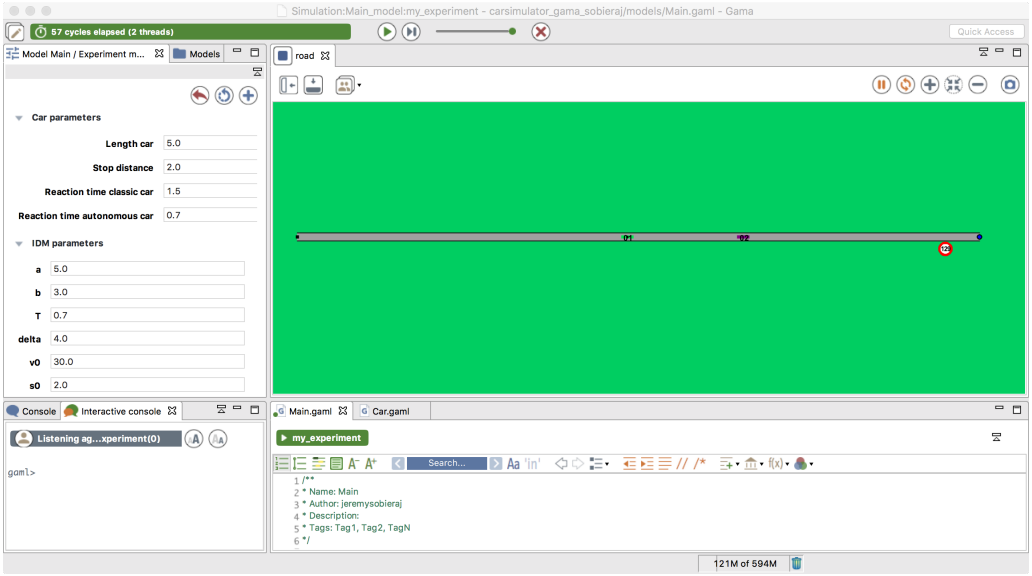

FIGURE 5.2 – Interface de l'outil GAMA

Les simulateurs permettent de modéliser et d'étudier diverses propriétés à différents niveaux de réalisme et d'échelle. Nous avons décidé d'illustrer notre étude de cas en utilisant l'outil GAMA [11] (Figure 5.2), qui est un simulateur de systèmes multi-agents utilisant un langage de programmation orienté agent (GAML). Chaque véhicule est caractérisé par une position, une vitesse et une accélération et se trouve dans un environnement à deux dimensions, longitudinale et latérale. À chaque pas de temps, chaque véhicule met à jour sa position en suivant un mouvement rectiligne uniforme. Il peut effectuer les actions suivantes : accélérer, ralentir, garder la même vitesse ou changer de voie. La proximité de cette représentation avec celle utilisée dans

nos modèles était adéquate pour l'implémentation d'un modèle équivalent dans GAMA. Cette implémentation a par ailleurs été permise par les extensions proposées par [80].

5.2 Etude d'un modèle de suivi de voiture : IDM

Pour notre étude de cas, nous avons choisi IDM afin d'observer des situations en combinant simulation et outils de vérification formelle. IDM permet de déterminer l'accélération d'un véhicule pour une situation donnée en observant l'état du véhicule qui le précède à un moment donné. Il peut être utilisé pour décrire un véhicule autonome utilisant un radar de régulation de distance mais peut également simuler le comportement d'un véhicule conduit par l'homme, selon les paramètres utilisés.

Considérons deux véhicules tels que décrits dans la Figure 5.3 : le véhicule suiveur i (dont le comportement est déterminé par IDM) et le véhicule *leader* $i-1$ (dont comportement est arbitraire). À chaque pas de temps, le véhicule i met à jour son accélération, qui varie en fonction de deux critères principaux : d'une part, le véhicule i tend à atteindre la vitesse maximale v_0 autorisée sur la portion de route (sa vitesse de croisière), d'autre part, il doit respecter une distance de sécurité minimale s^* avec le véhicule de devant. Cette distance varie selon la vitesse relative entre les deux véhicules.

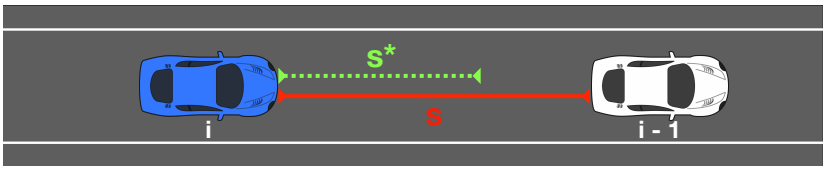

FIGURE 5.3 – Fonctionnement d'IDM : à tout moment, le véhicule i adapte sa vitesse en fonction de celle du véhicule $i-1$.

L'équation d'IDM permettant de calculer l'accélération du véhicule suiveur est comme suit :

$$acc_i = \frac{dv_i}{dt} = a \left[1 - \left(\frac{v_i}{v_0} \right)^{\delta} - \left(\frac{s^*(v_i, \Delta v_i)}{s} \right)^2 \right]$$

où la distance désirée de pare-chocs à pare-chocs $s^*(v_i, \Delta v_i)$ est :

$$s^*(v_i, \Delta v_i) = s_0 + \max \left[0, \left(v_i \times T + \frac{v_i \times \Delta v_i)}{2\sqrt{ab}} \right) \right]$$

avec a l'accélération maximale, v_i la vitesse du véhicule i, v_0 la vitesse maximale autorisée pour le véhicule i, δ l'exposant d'accélération (un coefficient "d'agressivité" du comportement),

$\Delta v_i = v_i - v_{i-1}$ la vitesse relative entre les véhicules, s_0 la distance minimale entre pare-chocs (la distance de sécurité), T le temps de réaction estimé et b la décélération maximale souhaitée.

Les paramètres, à savoir a, v_0, δ, s_0, T et b, doivent être fixés à l'initialisation. Pour tous les véhicules contrôlés avec IDM, nous avons décidé d'utiliser les valeurs suivantes correspondant à un véhicule autonome récent sur une autoroute. Le comportement correspond à une conduite sans changement de vitesses brutaux avec un temps de réaction légèrement plus court que celui d'un humain [12] :

— a : $5.0m.s^{-2}$

— v_0 : $30.0m.s^{-1}$ (moins de $110km.h^{-1}$) ;

— δ : 4 ;

— s_0 : $2m$;

— T : $0.7s$;

— b : $3m.s^{-2}$.

On notera qu'il n'y a pas de limite à la décélération dans IDM (seule la décélération souhaitée b est fixée). Cela signifie qu'il existe potentiellement des situations où la valeur de décélération dépasse b.

Dans ce qui suit, nous utiliserons l'indicateur de temps avant collision (TTC) pour étudier la sécurité et donc des cas potentiels de décélération anormalement élevée impliquant de l'insécurité ou de l'inconfort. Pour tous les scénarios présentés dans ce qui suit, la portion de route observée mesure 200 mètres de long, la longueur de chaque véhicule est exactement de 5 mètres et la prise de décision se produit toutes les 100 millisecondes.

5.3 Comparaisons des valeurs des indicateurs

Nous voulons tout d'abord vérifier la différence de comportement du système entre les approches de simulation et de vérification formelle. Pour ce faire, on comparera le temps de trajet à la fin de la portion de route et la position à un moment donné (trois secondes après le début du scénario). Le scénario 1 se compose de trois véhicules contrôlés par IDM. Au départ, le véhicule A est en position $0m$ et sa vitesse est de $30m.s^{-1}$, le véhicule B est en position de $50m$ et sa vitesse est de $25m.s^{-1}$, et le véhicule C est en position $100m$ et sa vitesse est de $20m.s^{-1}$. La Figure 5.4 illustre ce scénario.

Ici, les véhicules A et B doivent adapter leur vitesse pour éviter les collisions, ce qui semble être un bon exemple pour détecter une éventuelle accumulation d'erreur. Les résultats sont présentés dans le tableau 5.1 et montrent que la position et le temps de parcours des véhicules

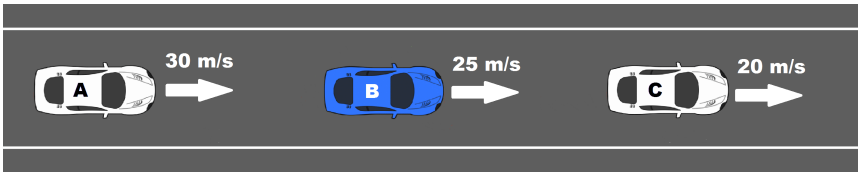

FIGURE 5.4 – Situation initiale du scénario 1 avec les vitesses respectives de chaque véhicule.

sont très proches, le pourcentage d'erreur le plus élevé correspondant au temps de parcours du véhicule B (0,52 %). Ces résultats montrent que le comportement moyen obtenu pour les deux outils est suffisamment similaire pour utiliser la position et le temps de déplacement comme des indicateurs fiables.

Critère	Temps de parcours			Position à 3 s		
	Véh. A	Véh. B	Véh. C	Véh. A	Véh. B	Véh. C
Simulation	$7.38s$	$5.68s$	$3.92s$	$80.51m$	$125.69m$	$174.03m$
Model checking	$7.39s$	$5.71s$	$3.92s$	$80.74m$	$125.26m$	$173.99m$

TABLE 5.1 – Comparaison du temps de déplacement et de la position à 3 secondes pour les trois véhicules du scénario 1.

A présent, nous vérifions le comportement vis-à-vis d'indicateurs plus fins tels que le TTC ou les valeurs d'accélération. Le scénario 2 comporte deux véhicules. Le véhicule A est contrôlé par IDM, initialement à la position $0m$ avec une vitesse de $20m.s^{-1}$, tandis que le véhicule B est initialement à la position de $50m$ avec une vitesse de $30m.s^{-1}$ et est contrôlé comme suit : B commence par décélérer à $-7m.s^{-2}$ pendant une seconde, accélère ensuite à $5m.s^{-2}$ pendant une seconde, puis décélère à nouveau à $-7m.s^{-2}$ jusqu'à ce qu'il s'arrête, comme illustré dans la Figure 5.5.

On peut remarquer que la situation initiale est sans danger pour le véhicule (la distance de sécurité est respectée). Sur ce scénario, nous vérifions les valeurs et moments d'apparition du pire TTC possible et de l'accélération minimale (autrement dit, la décélération la plus forte) pour le véhicule A. Sur ce même véhicule, nous vérifions également le moment où sa valeur d'accélération devient négative pour la première fois.

Les résultats sont présentés dans le tableau 5.2 et montrent que pour les deux outils, tous les événements se produisent dans les mêmes unités de temps. De plus, les valeurs de TTC sont très proches avec seulement 0.02 s de différence (1,13 % d'erreur), ce qui signifie que nous pouvons utiliser le TTC en tant qu'indicateur avec une confiance raisonnable. Par ailleurs, la valeur de l'accélération minimale est très différente, avec plus de $2m.s^{-2}$ de différence, ce qui

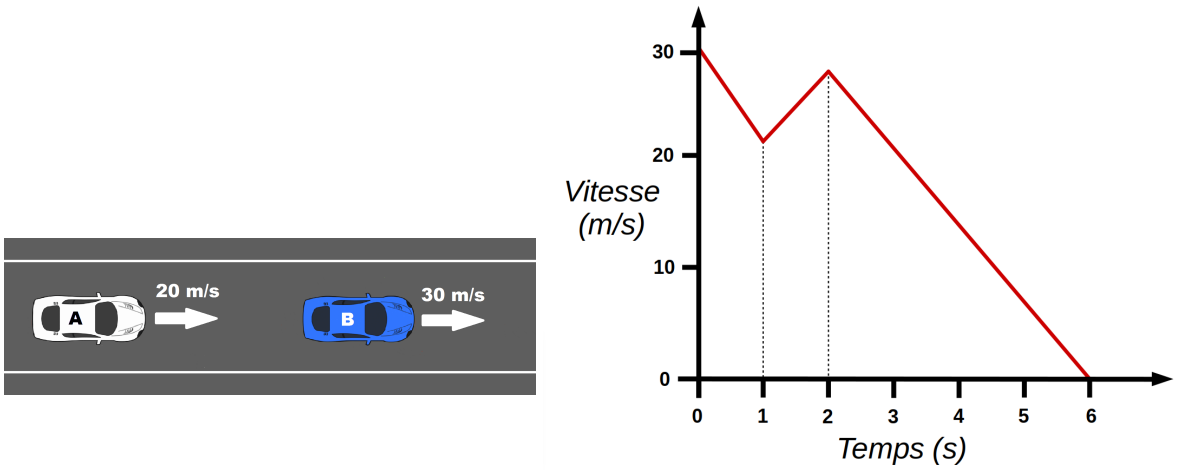

FIGURE 5.5 – Situation initiale du scénario 2 ainsi que l'évolution de la vitesse du véhicule B.

Critère	min TTC (valeur)	min TTC (temps)	min accéleration (valeur)	min accéleration (temps)	première décéleration (temps)
Simulation	$1.78s$	$6.0s$	$-7.36m.s^{-2}$	$6.0s$	$2.70s$
Model checking	$1.76s$	$6.0s$	$-9.40m.s^{-2}$	$6.0s$	$2.70s$

TABLE 5.2 – Comparaison des critères d'accélération et du TTC du comportement du véhicule A dans le scénario 2.

donne 27,71 % d'erreur. Cela est dû à l'accumulation des erreurs de calcul en valeurs discrètes lors de la prise de décision d'IDM. En raison d'un niveau d'abstraction différent, la valeur d'accélération calculée à une étape donnée dans la vérification est légèrement différente de celle obtenue dans la simulation. Si l'accélération calculée par l'équation d'IDM dans le domaine discret du model checker est supérieure à celle de la simulation, cela signifie que la vitesse à l'étape suivante sera supérieure à celle prévue par IDM, et le temps avant collision inférieur à ce qu'il aurait dû être. En conséquence, la prochaine accélération calculée par IDM sera cette fois inférieure à celle de la simulation, puisqu'elle compense cette différence. On peut donc observer des différences d'accélération importantes localement, même si les valeurs moyennes restent proches.

5.4 Application de la méthode

Dans cette section, nous donnons un exemple non déterministe sur lequel appliquer notre méthode combinée. Nous définissons le scénario 3 comme une variante non déterministe du scénario 2, où le véhicule B commence par une accélération de $-7m.s^{-2}$, change cette valeur pour $5m.s^{-2}$ à un moment aléatoire, puis la change à nouveau à $-7m.s^{-2}$ à un autre moment

aléatoire. Nous appelons le moment du premier événement e_1 et celui du second e_2, et définissons $\alpha = e_2 - e_1$ comme durée de l'accélération positive du véhicule B. Ces valeurs sont illustrées en Figure 5.6. On notera que si e_2 n'existe pas (l'événement ne se produit pas avant que le véhicule soit hors de la partie observée de route), on considère que α est infini.

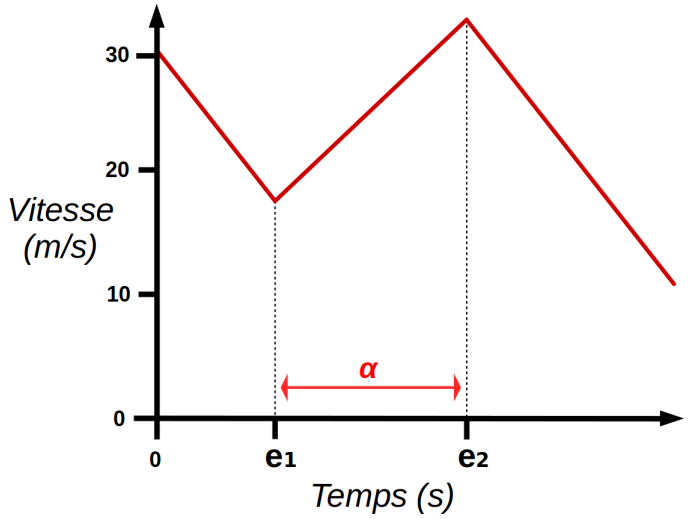

FIGURE 5.6 – Valeurs de e_1, e_2 et α pour un des comportements possibles du véhicule B.

Tout d'abord, nous lançons plusieurs exécutions de simulation aléatoires et obtenons des valeurs de TTC et d'accélération minimales. Sur la base de ces données, trois hypothèses on été posées :

1. Il existe une durée γ telle que si $e_1 \leq \gamma$, le TTC ne sera jamais sous 1.7 s,
2. Pour avoir un TTC inférieur à 1.7 s, α doit être compris entre $e_1 * 0.5$ et e_1,
3. Il n'existe pas d'exécution possible où le véhicule B s'arrête et sa valeur d'accélération n'est jamais inférieure à deux fois la valeur du paramètre de décélération souhaitée b (c'est-à-dire jamais inférieur à $-6m.s^{-2}$

Ces hypothèses donnent les requêtes CTL suivantes :

1. $\neg EF\ ttc < 1.7 \wedge e_1 \leq \gamma$
2. $AG\ (ttc < 1.7 \Rightarrow \alpha < e_1 \wedge \alpha > e_1/2)$
3. $\neg EG\ acceleration(A) \geq -6 \wedge on_the_road(A)$, où $acceleration$ est la valeur de l'accélération du véhicule A et $on_the_road(A)$ est une variable booléenne, qui est vraie si le véhicule A n'a pas encore atteint la fin de la portion de route.

La première hypothèse est facile à confirmer, mais il est plus intéressant de trouver un maximum pour γ. En utilisant le vérificateur de modèle, nous trouvons par dichotomie une valeur de

2.3 s, mais il faut garder à l'esprit qu'il peut y avoir un léger écart avec la simulation. Nous explorons donc le voisinage de cette valeur par simulation afin de l'affiner. La simulation indique une exécution avec $e_1 = 2.2s$, où un TTC inférieur à 1.7 s est trouvé, mais aucune à $e_1 = 2.1s$. Nous pouvons donc supposer que le maximum pour γ est en réalité de 2.1.

La deuxième hypothèse semble être fausse. En fait, le vérificateur de modèle trouve une exécution violant cette propriété à la fois pour la limite supérieure (avec $e_1 = 2,4$ et $e_2 = 4,9$) et pour la limite inférieure (avec $e_1 = 3.3$ et $e_2 = 4.4$). Nous affinons ensuite les limites supérieure et inférieure et obtenons finalement un résultat, où $\alpha \in [e_1 * 0.3, e_1 * 1.1]$. L'exploration de ces bornes par simulation ne montre aucun contre exemple.

Enfin, la troisième requête donne un résultat positif, mais nous devons garder à l'esprit que, comme nous l'avons montré, la valeur de l'accélération peut être potentiellement très différente entre la vérification et la simulation. La vérification de la même requête avec une valeur de $-7m.s^{-2}$ au lieu de $-6m.s^{-2}$ donne toujours un résultat positif, ajoutant une certaine confiance au résultat.

Le processus de vérification a pris moins de 10 secondes pour chaque requête, malgré la complexité du système due à la taille des plages de variables et du grand nombre de positions dans lesquelles chacun des agents peut se trouver. Cependant, le système étudié ne présente pas un niveau de non-déterminisme particulièrement élevé, car nous avons limité le nombre d'actions possibles que le véhicule *leader* peut effectuer à un moment donné. Il était ainsi plus facile de vérifier la cohérence de nos résultats par rapport à notre cas d'étude. Pour cette raison, on pourrait faire valoir qu'il aurait été possible de vérifier tous les comportements possibles avec la seule simulation dans un délai relativement raisonnable, ce qui pourrait bien fonctionner dans ce cas particulier. Cependant, dans le cas de comportements plus complexes, cela ne semble pas être une méthode raisonnable. De plus, bien que le temps de vérification augmente également avec la taille et la complexité du système, il est encore possible, dans une certaine mesure, de vérifier de manière exhaustive les hypothèses de systèmes non déterministes complexes dans un délai raisonnable (quelques heures au plus). On notera que la performance dépend principalement du niveau de non déterminisme. Par exemple, l'ajout d'autres véhicules déterministes à un scénario aura un coût très faible, tandis que l'augmentation des comportements possibles peut sérieusement affecter les performances.

5.5 Résumé du chapitre

Ce chapitre présente une méthode combinant vérification de modèle et simulation afin de tirer profit des avantages respectifs de ces deux approches. La méthode consiste à énoncer puis affiner une hypothèse sur le comportement d'un système. L'hypothèse est soumise à une véri-

fication exhaustive sur un modèle à haut niveau d'abstraction. Si un contre-exemple est trouvé, on utilise un simulateur avec un niveau d'abstraction moins élevé, donc plus proche du réel, afin de confirmer qu'il ne s'agit pas d'un faux positif. On peut alors conclure sur la validité de l'hypothèse et affiner cette dernière si nécessaire. Le processus se répète alors jusqu'à obtenir une hypothèse valide.

Un cas d'étude est décrit où sont utilisées l'outil de vérification de modèle UPPAAL et le simulateur de systèmes multi-agents GAMA. Le cas d'étude vise à analyser le comportement du modèle de suivi de voiture IDM, qui permet de déterminer l'accélération d'un véhicule en observant l'état du véhicule qui le précède à un moment donné. Il s'agit d'un modèle fréquemment utilisé en simulation de véhicules en raison de sa simplicité mais qui représente un défi dans le cadre de la vérification de modèles. En effet, il s'agit d'une équation retournant un nombre réel qui, dans UPPAAL, doit être discrétisé conduisant inévitablement à une perte d'information. C'est donc un choix pertinent afin de mettre à l'épreuve l'efficacité de la méthode.

Avant d'appliquer la méthode, une première comparaison des valeurs des indicateurs obtenues par les deux outils sur un scénario déterministe nous permet d'éliminer les indicateurs trop volatils et de ne conserver que les indicateurs fiables, c'est à dire ceux que l'accumulation d'erreurs affecte peu. La méthode combinée est alors appliquée sur un scénario dans lequel un véhicule au comportement aléatoire (le véhicule est en train de freiner brutalement, mais va reprendre une accélération positive pendant un laps de temps non déterminé) est positionné devant un véhicule contrôlé par IDM. Nous avons alors pu affiner rapidement, et avec un niveau de confiance élevé, une hypothèse sur les conditions conduisant aux situations les plus dangereuses.

Conclusion

Notre objectif était d'utiliser la vérification formelle dans le cadre d'un système de véhicules autonomes, comme cas d'étude de systèmes multi-agent temps réel et coopératifs. Nous avons tout d'abord spécifié les caractéristiques du système. Composé de véhicules en mouvement qui évoluent dans un environnement modifiable, le système permet l'utilisation d'algorithmes de prise de décision réalistes. La latence inhérente aux communications entre véhicules est représentée, et l'ensemble de données tient compte de la perception propre à chaque véhicule. Sur la base de cette spécification, deux méthodes d'analyse formelle ont été proposées, le cadre logiciel VERIFCARet le formalisme des MAPTs.

Le cadre logiciel VERIFCAR est basé sur les automates temporisés et dédié à la modélisation et à la vérification des CAVs. VERIFCAR convient à l'étude du comportement des CAV à un niveau d'abstraction plutôt élevé et aborde des questions de sécurité, d'efficacité et de robustesse avec un accent particulier sur l'impact des latences, des retards et pannes de communications sur le comportement des CAV. Le cadre logiciel développé est paramétrique et peut facilement être adapté pour étudier divers types de scénarios avec une précision réglable en fonction des besoins. Il est particulièrement bien adapté à l'analyse exhaustive de situations critiques impliquant un petit nombre de véhicules, tels que les dépassements ou les voies d'insertion. VERIFCAR implémente un modèle d'automates temporisés utilisant des synchronisations par *broadcast* permettant de réduire le nombre de chemins existants dans le système, et donc à accélérer les processus de vérification. Au moyen de discrétisations et d'approximations appropriées, les techniques exhaustives de vérification de modèle ont été rendues possibles tout en limitant les phénomènes d'explosion du nombre d'états. Enfin, VERIFCAR fournit des indicateurs pertinents pour l'analyse des comportements, ainsi qu'une méthode d'analyse utilisant la vérification de modèle. Nous avons appliqué VERIFCAR pour évaluer et comparer des politiques de décision pour les systèmes CAV sur des scénarios donnés. Pour ce faire, nous avons implémenté des algorithmes de décision, calculé les indicateurs associés et discuté des résultats obtenus. Nous avons également vérifié la robustesse dans des environnements défaillants, soulignant les vulnérabilités et illustrant ainsi la manière dont VERIFCAR pourrait être utilisé pour détecter et contrecarrer ces vulnérabilités. Cette approche a permis de comparer diverses

stratégies de coopération (négociations entre CAVs et communications via une infrastructure routière). Ce travail à fait à ce jour l'objet de deux publications, l'une dans une conférence internationale [81], l'autre dans une revue scientifique [8]. En outre, le code source des modèles et requêtes sont disponibles sur [73].

Nous avons ensuite introduit les G-MAPTs, des modèles multi-agents temporisés dans lesquels chaque agent est associé à un schéma temporisé régulier sur lequel reposent toutes les actions possibles de l'agent. Le formalisme permet de modéliser facilement des systèmes présentant un niveau élevé de simultanéité entre actions, où les actions ne sont pas déterministes et peuvent avoir lieu à différents moments dans le temps, telles que les CAVs. Nous avons ensuite formalisé les MAPTs (*Multi-Agent with timed Periodic Tasks*), en limitant les G-MAPTs. Un MAPT est ainsi un G-MAPT acyclique et dans lequel toute transition peut être franchie dans le futur. Ces contraintes permettent une sémantique accélérée et la décomposition du systèmes en couches d'états. La sémantique accélérée est une abstraction qui diminue la taille de l'espace d'états en réduisant autant que possible le nombre d'augmentation de temps du système. Cela à pour conséquence de supprimer des chemins sans pour autant perdre des comportements possibles en terme de séquence de transitions. Nous avons également présenté comment extraire une structure en couches d'un MAPT, qui permet de détecter les diamants tout en explorant le système en profondeur d'abord. Nous avons fourni une traduction des G-MAPT en réseaux de Petri de haut niveau, ce qui nous a permis de mettre en place un environnement de contrôle dédié à ce formalisme avec l'outil académique ZINC. Les algorithmes mis en œuvre dans de tels environnements explorent les espaces d'états de manière dynamique et peuvent être utilisés avec des méthodes heuristiques permettant de réduire le temps de calcul nécessaire pour atteindre certains états du modèle. Des expériences ont mis en évidence l'efficacité de nos abstractions, et une comparaison des systèmes de vérification de modèle CAV avec VERIFCAR a été effectuée. Bien que notre environnement de contrôle ne renvoie pas de traces d'exécutions et ne soit pas meilleur pour des temps d'explorations complètes que l'outil UPPAAL sur lequel repose VERIFCAR, il a une meilleure expressivité à la fois au niveau du modèle (nous pouvons utiliser des nombres réels) et sur les requêtes (nous pouvons utiliser des requêtes CTL imbriquées). Les résultats d'expérimentation concernant les heuristiques sont prometteurs pour les problèmes d'atteignabilité et nous avons également fourni un algorithme d'exploration permettant de rassembler toutes les informations nécessaires à l'analyse du système en une seule exécution, ce qui, comparativement à VERIFCAR, réduit considérablement le temps nécessaire à rassembler les informations permettant l'analyse du système. Bien que nous ayons développé cette méthode en gardant à l'esprit l'étude de cas de véhicules autonomes, ce formalisme, ainsi que toutes les abstractions et algorithmes présentés, ont une portée plus générale et peuvent s'appliquer à tout système temps réel multi-agents où les agents adoptent un comportement cy-

clique. Des articles traitant de ce travail sont actuellement en cours de rédaction. Le code source des modèles et des algorithmes permettant leur exploration sont disponibles sur [74].

Finalement, nous avons proposé une méthode combinant vérification de modèle et simulation afin d'accroître la confiance dans le réalisme des modèles vérifiés. Cette méthode permet principalement de contrôler que les erreurs dues aux approximations inhérentes à la vérification de modèle ne conduisent pas à un comportement s'éloignant trop du réel. Elle nous indique notamment, pour un système donné, quels indicateurs sont fiables et lesquels devraient être évités. Nous avons illustré par une étude de cas qu'il était possible d'obtenir efficacement des informations utiles sur le comportement des véhicules autonomes. En particulier, l'analyse du modèle de suivi de trajectoire IDM a mis en évidence certains comportements problématiques non triviaux. Il a ainsi été mis en avant que l'erreur induite par les approximations de la modélisation en UPPAAL était faible, les comportements obtenus étant très proches. Les méthodes de vérification formelles présentés dans le cadre de la thèse (le cadre logiciel VERIFCAR et le formalisme des MAPTs) peuvent être utilisées conjointement avec des simulations, contribuant à garantir la fiabilité des résultats obtenus sur les systèmes étudiés. Ce travail à fait l'objet d'un article publié dans une conférence internationale [82].

L'ensemble des techniques proposées ici peuvent être utilisées pour étudier les protocoles de communication entre véhicules autonomes afin d'en améliorer la qualité de la prise de décision. Un travail en partenariat avec l'Institut de Recherche Technologique SystemX, dans le cadre d'un projet financé Mévaro [83], est actuellement en cours. Il est prévu d'appliquer nos méthodes dans le cadre de la conception d'un module de prise de décision de véhicule autonome dans un contexte de redondance d'information. Ce projet devrait permettre de mettre à l'épreuve les résultats théoriques en utilisant les outils et algorithmes développés durant la thèse dans un contexte industriel.

A moyen terme, nous souhaitons donner plus de flexibilité au modèle, notamment vis à vis de la représentation de l'environnement. Le modèle devrait permettre des situations à plusieurs routes se joignant sur des carrefours ou des rond-points. En outre, nous prévoyons de mettre en place des interfaces pour nos différents outils afin de permettre la génération de modèles en fonction de paramètres rentrés par l'utilisateur. L'automatisation de l'analyse des systèmes CAVs est également prévue afin de rendre l'approche plus facile d'accès et d'éviter les erreurs humaines.

Enfin le formalisme des G-MAPTs ouvre de nombreuses perspectives théoriques et applications pratiques. Le formalisme peut être utilisé dans d'autres domaines d'application, différents des systèmes CAV, et l'application à des cas d'études variés est envisagée. Les abstractions portant sur la dynamique abstraite peuvent quand à elles être poussé encore d'avantage. En effet, l'extraction de cette dynamique (périodique dans le cadre des G-MAPTs) pourrait être

effectuée en préalable à l'exploration du système. Il n'y aurait alors plus besoin de gérer les aspects temporels pendant l'exploration de l'espace d'état, ni de vérifier les gardes des différentes transitions, les transitions autorisées à partir d'un état étant déjà connues. Cela permettrait par ailleurs l'utilisation de l'exploration en couches sur l'ensemble des G-MAPTs, et ce tout en réduisant les temps d'exploration.

Annexe A

Coopération par négociation

Dans cette variante de prise de décision, un tableau de booléens $comm[id].nego[]$ est ajouté dans la structure $comm[id]$ contenant l'ensemble des variables liées aux données échangées et reçues par le véhicule id. Le tableau donne à l'indice j la dernière valeur de $nego$ envoyée au véhicule id par le véhicule j. Si cette valeur est $vraie$, cela indique que j souhaite que id modifie son comportement. La diffusion de cette variable est ajoutée dans la fonction $communicate(id)$.

Algorithm 6 Pseudo-code de haut niveau pour la fonction de prise de décision. Les aspects de négociation sont ajoutés en vert.

Calcul de l'ensemble des trajectoires temporelles des véhicules de devant
Choix de la voie L à atteindre en fonction de la navigation
for $Acceleration \leftarrow MaxAcc$ **to** $MinAcc$ **do**
  for $Lane \in [1, NbLanes]$, en commençant par L et en explorant le voisinage de L **do**
    for $Delay \leftarrow 0$ **to** $MaxDelay$ **do**
      if Le comportement choisi ne génère pas de conflit avec les véhicules se trouvant devant **then**
        for all véhicule j ayant demandé la négociation **do**
          Calcul de la trajectoire temporelle souhaitée du véhicule j
          for $D \leftarrow Delay$ **to** $MaxDelay$ **do**
            if Le nouveau comportement ne génère pas de conflit avec les véhicules se trouvant devant ni avec le véhicule j **then**
              $Delay \leftarrow D$
            end if
          end for
        end for
        if $Delay = 0$ (Attente non nécessaire) **then**
          Choix de la $Direction$ en fonction de la $Lane$ choisie
        else
          $Direction \leftarrow 0$ (Le véhicule va tout droit car il est en attente)
        end if
        for all véhicule de devant j en train de changer de voie **do**
          if Trajectoire initialement souhaitée en conflit avec j **then**
            La demande de négociation avec j est mémorisée pour être communiquée
          end if
        end for
      end if
    end for
  end for
end for
Comportement d'urgence

Annexe B

Coopération par infrastructures

Dans cette variante de prise de décision, les variables $comm[id].ordered_acc$ et $comm[id].ordered_delay$ sont ajoutées dans la structure $comm[id]$ contenant l'ensemble des variables liées aux données échangées et reçues par le véhicule id. Elles correspondent aux ordres envoyés par le terminal au véhicule. La fonction $communicate(id)$ communique également au terminal les informations transmises par le véhicule id. Additionnellement, la structure de données du terminal contient :

- $to_order[]$, un tableau de booléens indiquant à quel véhicule le terminal doit communiquer ;
- $ordered_acc[]$, un tableau de variable indiquant l'accélération à communiquer à chaque véhicule ;
- $ordered_delay[]$, un tableau de variable indiquant le délai à communiquer à chaque véhicule.

La fonction de communication du terminal, décrite ci-après, est différente de celle des véhicules :

```
1 for(n in range(nb_car)){
2   if(to_order[n]){
3     comm[n].ordered_acc := ordered_acc[n];
4     comm[n].ordered_delay := ordered_delay[n];
5   }
6 }
```

La prise de décision du terminal est également différente de celle des véhicules et est décrite par l'algorithme suivant :

La modification au niveau de la prise de décision pour les véhicules consiste alors simplement à remplacer $MaxAcc$ par la valeur d'accélération envoyée par le terminal et la va-

Algorithm 7 Pseudo-code de haut niveau pour la fonction de prise de décision du terminal.

```
Calcul de l'ensemble des trajectoires temporelles des véhicules
for all véhicule i do
  Calcul de la trajectoire temporelle souhaitée du véhicule i
  if véhicule de devant j en train de changer de voie et en conflit avec i then
    Overall_Acc ← Acceleration(i) + Acceleration(j)
    for Acceleration ← MaxAcc to Overall_Acc − MaxAcc do
      for Delay ← 0 to MaxDelay do
        if Le comportement choisi pour j ne génère pas de conflit avec les véhicules se
        trouvant devant then
          Les valeurs d'Acceleration et de Direction sont mémorisées pour être envoyées
          au véhicule j
        end if
      end for
    end for
  end if
end for
```

leur 0 du délai par celle envoyée par le terminal. Ainsi, les valeurs d'accélération et de délais possibles pour le véhicule id seront respectivement $[MinAcc, comm[id].ordered_acc]$ et $[comm[id].ordered_delay, MaxDelay]$.

Annexe C

Algorithmes des requêtes pour l'exploration des MAPTs

Tout d'abord, définissons les fonctions et les constantes qui seront utilisées :

- Une file d'états T est définie avec $T[]$ et les fonctions suivantes sont utilisable :
  - $T.add(s)$ qui ajoute s en fin de file si $s \notin T$.
  - $T.add(S)$ qui pour tout $s \in S$, applique $T.add(s)$.
  - $T.pop()$ qui renvoie l'état à la fin de la file et le supprime.
- La formule booléenne $final$ est vraie pour tout état final du système et fausse sinon.
- La fonction $check(p, s)$ est vraie si la formule booléenne p est vraie dans l'état s.
- La fonction $next_border(s)$ renvoie tous les successeurs de l'état s.
- La constante init indique l'état initial du système.
- La fonction $mark(s)$ marque l'état s.
- La fonction $is_marked(s)$ retourne vrai si s est marqué.
- La fonction $parent_marked(s)$ retourne vrai si le prédécesseur de s est marqué.

Algorithm 8 EFp : À partir de l'état initial, on calcule chaque successeur récursivement jusqu'à atteindre p ou qu'il n'y ai plus d'états à explorer.

```
exploring[]
exploring.add(init)
while exploring do
  successors ← next_border(exploring.pop())
  for all s ∈ successors do
    if check(p, s) then
      return true
    else if not check(final, s) then
      exploring.add(s)
    end if
  end for
end while
return false
```

Algorithm 9 EGp : À partir de l'état initial, si p est vrai, on calcule chaque successeur où p est vrai de manière récursive jusqu'à atteindre un état final ou qu'il n'y ai plus d'états à explorer.

```
exploring[]
exploring.add(init)
while exploring do
  successors ← next_border(exploring.pop())
  for all s ∈ successors do
    if check(p, s) then
      if check(final, s) then
        return true
      else
        exploring.add(s)
      end if
    end if
  end for
end while
return false
```

Algorithm 10 $EF(p \wedge EFq)$: À partir de l'état initial, on calcule chaque successeur de manière récursive jusqu'à atteindre q sur un état marqué ou qu'il n'y ai plus d'états à explorer. Chaque fois que p est vrai sur un état, l'état est marqué. Tous les successeurs d'un état marqué sont marqués.

```
exploring[]
exploring.add(init)
while exploring do
  successors ← next_border(exploring.pop())
  for all s ∈ successors do
    if check(p, s) or parent_marked(s) then
      mark(s)
    end if
    if is_marked(s) and check(q, s) then
      return true
    else if not check(final, s) then
      exploring.add(s)
    end if
  end for
end while
return false
```

Algorithm 11 $EF(p \wedge EGq)$: À partir de l'état initial, on calcule chaque successeur récursivement jusqu'à atteindre un état final marqué ou qu'il n'y ai plus d'états à explorer. Chaque fois que p et q sont vrais sur un état, l'état est marqué. Un successeur d'un état marqué est marqué si q est vrai sur celui-ci.

```
exploring[]
exploring.add(init)
while exploring do
  successors ← next_border(exploring.pop())
  for all s ∈ successors do
    if (check(p, s) or parent_marked(s)) and check(q, s) then
      mark(s)
    end if
    if is_marked(s) and check(final, s) then
      return true
    else if not check(final, s) then
      exploring.add(s)
    end if
  end for
end while
return false
```

Bibliographie

[1] Jean-Louis Boulanger. *Industrial use of formal methods : formal verification*. John Wiley & Sons, 2013.

[2] Thomas Tilley, Richard Cole, Peter Becker, and Peter Eklund. *A Survey of Formal Concept Analysis Support for Software Engineering Activities*, pages 250–271. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[3] Edmund M. Clarke. Model checking. In S. Ramesh and G. Sivakumar, editors, *Foundations of Software Technology and Theoretical Computer Science*, pages 54–56, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[4] E. Allen Emerson and Joseph Y. Halpern. "sometimes" and "not never" revisited : on branching versus linear time temporal logic. *J. ACM*, 1(33) :151–178, 1986.

[5] K. Vogel. A comparison of headway and time to collision as safety indicators. *Accident Analysis & Prevention*, 35(3) :427 – 433, 2003.

[6] M. Minderhoud and P. Bovy. Extended time-to-collision measures for road traffic safety assessment. *Accident Analysis & Prevention*, 33(1) :89 – 97, 2001.

[7] Franck Pommereau. ZINC : a compiler for “any language”-coloured Petri nets. Technical report, IBISC, university of Evry / Paris-Saclay, 2018.

[8] Johan Arcile, Raymond Devillers, and Hanna Klaudel. Verifcar : a framework for modeling and model checking communicating autonomous vehicles. *Autonomous Agents and Multi-Agent Systems*, 33(3) :353–381, May 2019.

[9] Uppaal. `http://www.uppaal.org/`.

[10] Kim G. Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1-2) :134–152, Oct 1997.

[11] Patrick Taillandier, Duc An Vo, Edouard Amouroux, and Alexis Drogoul. GAMA : A simulation platform that integrates geographical information data, agent-based modeling and multi-scale control. In *Lecture Notes in Computer Science (including subseries Lec-*

ture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), volume 7057 LNAI, pages 242–258, 2012.

[12] Martin Treiber and Arne Kesting. Microscopic calibration and validation of car-following models–a systematic approach. *Procedia-Social and Behavioral Sciences*, 80 :922–939, 2013.

[13] M. Wooldridge. *An introduction to multi-agent systems - Second Edition*. John Wiley & Sons, 2009.

[14] R. Alur and D. Dill. Automata for modelling real-time systems. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP'90)*, volume 443 of *LNCS*, pages 322–335. Springer-Verlag, 1990.

[15] Banu Y. Ekren and Sunderesh S. Heragu. Simulation based performance analysis of an autonomous vehicle storage and retrieval system. *Simulation Modelling Practice and Theory*, 19(7) :1640 – 1650, 2011.

[16] Hussein Dia. An agent-based approach to modelling driver route choice behaviour under the influence of real-time information. *Transportation Research Part C : Emerging Technologies*, 10(5) :331 – 349, 2002.

[17] T. Shamir. How should an autonomous vehicle overtake a slower moving vehicle : design and analysis of an optimal trajectory. *IEEE Transactions on Automatic Control*, 49(4) :607–610, April 2004.

[18] Arda Kurt, John L Yester, Yutaka Mochizuki, and Ümit Özgüner. Hybrid-state driver/-vehicle modelling, estimation and prediction. In *13th International IEEE Conference on Intelligent Transportation Systems*, pages 806–811. IEEE, 2010.

[19] F. Bai and H. Krishnan. Reliability analysis of DSRC wireless communication for vehicle safety applications. In *IEEE Intelligent Transportation Systems Conference*, pages 355–362, Sept 2006.

[20] M. Treiber and A. Kesting. *Trajectory and Floating-Car Data*, pages 7–12. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[21] S. Zhang, W. Deng, Q. Zhao, H. Sun, and B. Litkouhi. Dynamic trajectory planning for vehicle autonomous driving. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 4161–4166, Oct 2013.

[22] C. Katrakazas, M. Quddus, W.-H. Chen, and L. Deka. Real-time motion planning methods for autonomous on-road driving : State-of-the-art and future research directions. *Transportation Research Part C : Emerging Technologies*, 60 :416 – 442, 2015.

[23] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How. Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on Control Systems Technology*, 17(5) :1105–1118, Sept 2009.

[24] A. Furda and L. Vlacic. Enabling safe autonomous driving in real-world city traffic using multiple criteria decision making. *IEEE Intelligent Transportation Systems Magazine*, 3(1) :4–17, Spring 2011.

[25] M. Likhachev and D. Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*, 28(8) :933–945, 2009.

[26] S. Glaser, B. Vanholme, S. Mammar, D. Gruyer, and L. Nouveliere. Maneuver-based trajectory planning for highly autonomous vehicles on real road with traffic and driver interaction. *IEEE Transactions on Intelligent Transportation Systems*, 11(3) :589–606, Sept 2010.

[27] K.D. Stanley P. Sorensen C. Samaras J.M. Anderson, N. Kalra and T. A. Oluwatola. *Autonomous Vehicle Technology. A Guide for Policymakers*. Research Reports. RAND Corporation, 2016. ISBN : 978-0-8330-8398-2.

[28] J.W.C. van Lint, S.P. Hoogendoorn, and H.J. van Zuylen. Accurate freeway travel time prediction with state-space neural networks under missing data. *Transportation Research Part C : Emerging Technologies*, 13(5) :347 – 369, 2005.

[29] Gang-Len Chang and Hani S. Mahmassani. Travel time prediction and departure time adjustment behavior dynamics in a congested traffic system. *Transportation Research Part B : Methodological*, 22(3) :217 – 232, 1988.

[30] M. Foughali, B. Berthomieu, S. Dal Zilio, F. Ingrand, and A. Mallet. Model Checking Real-Time Properties on the Functional Layer of Autonomous Robots. In *International Conference on Formal Engineering Methods (ICFEM 2016)*, Tokyo, Japan, November 2016.

[31] M. Kamali, L. A. Dennis, O. McAree, M. Fisher, and S. M. Veres. Formal verification of autonomous vehicle platooning. *Science of Computer Programming*, 148 :88 – 106, 2017. Special issue on Automated Verification of Critical Systems (AVoCS 2015).

[32] M. O'Kelly, H. Abbas, and R. Mangharam. APEX : Autonomous vehicle plan verification and execution. In *SAE World Congress*, 2016.

[33] S. Kong, S. Gao, W. Chen, and E. Clarke. dreach : δ-reachability analysis for hybrid systems. In Christel Baier and Cesare Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 200–205, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[34] A. Platzer and J.-D. Quesel. European train control system : A case study in formal verification. In Karin Breitman and Ana Cavalcanti, editors, *Formal Methods and Software Engineering*, pages 246–265, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[35] M. M. Quottrup, T. Bak, and R. I. Zamanabadi. Multi-robot planning : a timed automata approach. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, volume 5, pages 4417–4422 Vol.5, April 2004.

[36] E Allen Emerson. Temporal and modal logic, handbook of theoretical computer science (vol. b) : formal models and semantics, 1991.

[37] Stefan Edelkamp, Alberto Lluch Lafuente, and Stefan Leue. Directed explicit model checking with hsf-spin. In *Proceedings of the 8th International SPIN Workshop on Model Checking of Software*, SPIN '01, pages 57–79, Berlin, Heidelberg, 2001. Springer-Verlag.

[38] Kenneth L. McMillan. *Symbolic Model Checking*, pages 25–60. Springer US, Boston, MA, 1993.

[39] Patrice Godefroid, J Van Leeuwen, J Hartmanis, G Goos, and Pierre Wolper. *Partial-order methods for the verification of concurrent systems : an approach to the state-explosion problem*, volume 1032. Springer Heidelberg, 1996.

[40] Doron Peled. Combining partial order reductions with on-the-fly model-checking. In David L. Dill, editor, *Computer Aided Verification*, pages 377–390, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[41] Pierre Wolper and Patrice Godefroid. Partial-order methods for temporal verification. In Eike Best, editor, *CONCUR'93*, pages 233–246, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[42] Alice Miller, Alastair Donaldson, and Muffy Calder. Symmetry in temporal logic model checking. *ACM Comput. Surv.*, 38(3), September 2006.

[43] E. M. Clarke, E. A. Emerson, S. Jha, and A. P. Sistla. Symmetry reductions in model checking. In Alan J. Hu and Moshe Y. Vardi, editors, *Computer Aided Verification*, pages 147–158, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[44] G. Bhat, R. Cleaveland, and O. Grumberg. Efficient on-the-fly model checking for ctl. In *Proceedings of Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 388–397, June 1995.

[45] Ilan Beer, Shoham Ben-David, and Avner Landver. On-the-fly model checking of rctl formulas. In Alan J. Hu and Moshe Y. Vardi, editors, *Computer Aided Verification*, pages 184–194, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[46] Doron Peled. Combining partial order reductions with on-the-fly model-checking. In David L. Dill, editor, *Computer Aided Verification*, pages 377–390, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[47] Viktor Gyuris and A. Prasad Sistla. On-the-fly model checking under fairness that exploits symmetry. In Orna Grumberg, editor, *Computer Aided Verification*, pages 232–243, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[48] Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8) :677–691, Aug 1986.

[49] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking : 1020 states and beyond. *Information and Computation*, 98(2) :142 – 170, 1992.

[50] Nina Amla, Xiaoqun Du, Andreas Kuehlmann, Robert P. Kurshan, and Kenneth L. McMillan. An analysis of sat-based model checking techniques in an industrial environment. In Dominique Borrione and Wolfgang Paul, editors, *Correct Hardware Design and Verification Methods*, pages 254–268, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[51] Aaron R. Bradley. Sat-based model checking without unrolling. In Ranjit Jhala and David Schmidt, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 70–87, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[52] Edmund Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1) :7–34, Jul 2001.

[53] Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, Yunshan Zhu, et al. Bounded model checking. *Advances in computers*, 58(11) :117–148, 2003.

[54] Maria Sorea. Bounded model checking for timed automata. *Electronic Notes in Theoretical Computer Science*, 68(5) :116–134, 2003.

[55] K. L. McMillan. Applications of craig interpolants in model checking. In Nicolas Halbwachs and Lenore D. Zuck, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 1–12, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[56] Niklas Eén and Niklas Sörensson. Temporal induction by incremental sat solving. *Electronic Notes in Theoretical Computer Science*, 89(4) :543–560, 2003.

[57] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1) :109 – 137, 1984.

[58] Jos CM Baeten. *Applications of process algebra*, volume 17. Cambridge university press, 2004.

[59] Xavier Nicollin and Joseph Sifakis. An overview and synthesis on timed process algebras. In Kim G. Larsen and Arne Skou, editors, *Computer Aided Verification*, pages 376–398, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

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

[61] Gerd Behrmann, Alexandre David, and Kim G. Larsen. *A Tutorial on Uppaal*, pages 200–236. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[62] James L. Peterson. *Petri Net Theory and the Modelling of Systems*. Prentice Hall, 1981.

[63] Kurt Jensen. *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use - Volume 1*. EATCS Monographs on TCS. Springer, 1992.

[64] C. Ghezzi, D. Mandrioli, S. Morasca, and M. Pezze. A unified high-level petri net formalism for time-critical systems. *IEEE Transactions on Software Engineering*, 17(2) :160–172, Feb 1991.

[65] T. L. Willke, P. Tientrakool, and N. F. Maxemchuk. A survey of inter-vehicle communication protocols and their applications. *IEEE Communications Surveys Tutorials*, 11(2) :3–20, Second 2009.

[66] S. Biswas, R. Tatchikou, and F. Dion. Vehicle-to-vehicle wireless communication protocols for enhancing highway traffic safety. *IEEE Communications Magazine*, 44(1) :74–82, Jan 2006.

[67] K. Bilstrup, E. Uhlemann, E. G. Strom, and U. Bilstrup. Evaluation of the ieee 802.11p mac method for vehicle-to-vehicle communication. In *IEEE Vehicular Technology Conference*, pages 1–5, Sept 2008.

[68] C. Urmson et al. Autonomous driving in urban environments : Boss and the urban challenge. *Journal of Field Robotics Special Issue on the 2007 DARPA Urban Challenge, Part I*, 25(8) :425–466, June 2008.

[69] J. Levinson and S. Thrun. Robust vehicle localization in urban environments using probabilistic maps. In *IEEE International Conference on Robotics and Automation*, pages 4372–4378, May 2010.

[70] Akers. Binary decision diagrams. *IEEE Transactions on Computers*, C-27(6) :509–516, June 1978.

[71] R. Alur and D. Dill. A theory of timed automata. *Theoretical computer science*, 126(2) :183–235, 1994.

[72] R. J. Blokpoel, D. Krajzewicz, and R. Nippold. Unambiguous metrics for evaluation of traffic networks. In *IEEE Intelligent Transportation Systems Conference*, pages 1277–1282, Sept 2010.

[73] Source templates of the verifcar framework. `https://forge.ibisc.univ-evry.fr/jarcile/VerifCar/`. Accessed : 2019-10-11.

[74] Source of the mapts models and exploration algorithms. `https://forge.ibisc.univ-evry.fr/jarcile/MAPTs/`. Accessed : 2019-10-11.

[75] P. G. Gipps. A behavioural car-following model for computer simulation. *Transportation Research Part B*, 15(2) :105–111, 1981.

[76] Sumin Zhang, Weiwen Deng, Qingrong Zhao, Hao Sun, and Bakhtiar Litkouhi. Dynamic trajectory planning for vehicle autonomous driving. In *Proceedings - 2013 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2013*, pages 4161–4166, 2013.

[77] Fan Bai and Hariharan Krishnan. Reliability Analysis of DSRC Wireless Communication for Vehicle Safety Applications. *IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 355–362, 2006.

[78] Arne Kesting, Martin Treiber, and Dirk Helbing. General Lane-Changing Model MOBIL for Car-Following Models. *Transportation Research Record : Journal of Transportation Research Board*, 1999(1) :86–94, 2007.

[79] Jacques Ferber and Gerhard Weiss. *Multi-agent systems : an introduction to distributed artificial intelligence*, volume 1. Addison-Wesley Reading, 1999.

[80] Jérémy Sobieraj. *Methods and tools for the design of Cooperative Intelligent Transportation Systems*. Theses, Université Paris-Saclay ; Université d'Evry-Val-d'Essonne, November 2018.

[81] Johan Arcile, Raymond R. Devillers, Hanna Klaudel, Witold Klaudel, and Bozena Wozna-Szczesniak. Modeling and checking robustness of communicating autonomous vehicles. In *Distributed Computing and Artificial Intelligence, 14th International Conference, DCAI 2017, Porto, Portugal, 21-23 June, 2017*, pages 173–180, 2017.

[82] J. Arcile, J. Sobieraj, H. Klaudel, and G. Hutzler. Combination of simulation and model-checking for the analysis of autonomous vehicles' behaviors : a case study. In *Multi-Agent Systems and Agreement Technologies*. Springer International Publishing, 2018.

[83] Abderrahmane Sali. Etude de la robustesse du modèle décisionnel des véhicules autonomes et connectés en présence de redondances d'informations. Mémoire de stage de master, IBISC, university of Evry / Paris-Saclay, 2019.

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Conception, modélisation et vérification formelle d'un système temps-réel d'agents coopératifs : Application aux véhicules autonomes communicants

Mots clés : Modèle formel, Vérification, Temps réel, Abstraction, Véhicules autonomes

Résumé : Cette thèse est motivée par la question de la validation de propriétés dans un système composé de plusieurs agents mobiles prenants individuellement des décisions en temps réel. Chaque agent a une perception de l'environnement qui lui est propre et peut communiquer avec les autres agents à proximité. L'application qui a été choisie comme cas d'étude est celle des véhicules autonomes, qui du fait du large nombre de variables impliquées dans la représentation de tels systèmes, rend impossible des approches naïves. Les problématiques traitées concernent, d'une part, la modélisation d'un tel système, notamment le choix du formalisme et du niveau d'abstraction du modèle, et d'autre part, la mise en place d'un protocole d'évaluation de la prise de décision des véhicules. Ce dernier point inclut la question de l'efficacité de l'exploration de l'espace d'états du modèle. La thèse présente un ensemble de travaux, pouvant être complémentaires, visant à traiter ces problématiques. Tout d'abord, le système, composé des véhicules autonomes et de leur environnement, est défini avec précision. Il permet notamment d'observer l'impact des communications entre véhicules sur leur comportement. Le cadre logiciel VERIFCAR dédié à l'analyse de prise de décision de véhicules autonomes communicants est ensuite présenté. Il inclut un modèle paramétrique d'automates temporisés offrant la possibilité de vérifier des propriétés de logique temporelle. Une méthodologie d'analyse utilisant ces propriétés est présentée. On propose également une approche complémentaire permettant dans certains cas une meilleure efficacité et une plus grande expressivité. Elle est fondée sur le formalisme des MAPTs (*Multi-Agent with timed Periodic Tasks*), qui a été conçu pour la modélisation de systèmes temps réel d'agents coopératifs. Des algorithmes permettant une exploration dynamique des états de ce type de modèles (c'est à dire sans que l'espace d'états ne doive être préalablement construit) sont présentés. Enfin, une méthode combinée alliant la simulation aux outils de vérification de modèle afin de contrôler le niveau de réalisme est décrite et appliquée au cas d'étude.

Title : Design, formal modeling and verification of a real-time system of cooperative agents: Application to communicating autonomous vehicles

Keywords : Formal modeling, Verification, Real-time, Abstraction, Autonomous vehicles

Abstract : This thesis is motivated by the question of the validation of properties in a system composed of several mobile agents individually making decisions in real time. Each agent has a perception of their own environment and can communicate with other agents nearby. The application that has been chosen as a case study is that of autonomous vehicles, which because of the large number of variables involved in the representation of such systems, makes naive approaches impossible. The issues addressed concern, on the one hand, the modeling of such a system, in particular the choice of the formalism and the level of abstraction of the model, and on the other hand, the implementation of an evaluation protocol of decision making of vehicles. This last point includes the question of the efficiency of the exploration of the state space of the model. The thesis presents a set of works, which can be complementary, aiming to treat these problems. First, the system, consisting of autonomous vehicles and their environment, is precisely defined. It allows in particular to observe the impact of communications between vehicles on their behavior. The VerifCar software framework dedicated to decision-making analysis of communicating autonomous vehicles is then presented. It includes a parametric model of timed automata with the ability to check temporal logic properties. An analysis methodology using these properties is presented. A complementary approach is also proposed, which in some cases allows for greater efficiency and greater expressiveness. It is based on the formalism of MAPTs (*Multi-Agent with timed Periodic Tasks*), which was designed for modeling real-time systems of cooperative agents. Algorithms allowing a dynamic exploration of the states of this type of model (that is to say without the state space having to be built beforehand) are presented. Finally, a combined method combining simulation and model verification tools to control the level of realism is described and applied to the case study.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France