Analyse de l'erreur en vérification probabiliste

Mémoire

Gildas Syla Déo Kouko

Maîtrise en informatique
Maître ès sciences (M.Sc.)

Québec, Canada

Résumé

La vérification de systèmes est aujourd'hui un sujet de recherche récurrent et différentes techniques permettent de vérifier formellement des systèmes critiques dont il faut impérativement garantir la correction. Nous consacrons ce mémoire à l'une des techniques les plus utilisées et les plus efficaces, l'*évaluation de modèle*.

Schématiquement, pour vérifier un système par *évaluation de modèle*, on abstrait d'abord son comportement sous la forme d'un système de transitions appelé *modèle*. Ensuite, on formule une propriété désirée du système dans une logique temporelle. Enfin, on utilise un outil logiciel appelé *vérificateur* pour vérifier automatiquement si le *modèle* satisfait la propriété.

Dans ce mémoire, nous voulons vérifier des propriétés d'atteignabilité dans des modèles probabilistes appelés processus de Markov étiquetés (en anglais, LMP pour Labelled Markov processes) et qui ont possiblement un ensemble d'états non dénombrable. Malheureusement, le *vérificateur* CISMO dédié à une famille de LMP ne gère pas les propriétés d'atteignabilité et aucun autre outil ne peut vérifier les LMP.

Pour améliorer CISMO et atteindre notre objectif, nous avons rendu d'abord plus expressive sa logique de spécification de propriétés pour qu'elle exprime les propriétés d'atteignabilité sur les LMP. Ces propriétés expriment le fait qu'un état souhaité dans un système peut être atteint avec une certaine probabilité. Ensuite, nous avons implémenté dans CISMO une nouvelle approche de vérification d'une famille de propriétés d'atteignabilité qui contribue à l'évolution de la vérification probabiliste.

Nous utilisons le *théorème de la moyenne* pour prouver que, pour tout LMP acceptable par CISMO et toute propriété d'atteignabilité, il existe une chaîne de Markov à temps discret (en anglais, DTMC pour Discrete Time Markov Chains) équivalent au LMP de point de vue atteignabilité moyenne et auquel on peut appliquer les algorithmes connus pour les systèmes probabilistes finis. Le DTMC est construit de telle sorte que nous

inférons que le LMP satisfait la propriété d'atteignabilité, si et seulement si le DTMC la satisfait. Théoriquement, notre approche donne un résultat ultime exact et nous l'avons prouvé.

À l'implémentation, nous utilisons une méthode d'intégration numérique pour déterminer les probabilités de transition dans le DTMC. Malgré les imprécisions numériques qui peuvent nuire au résultat d'une vérification, nous avons prouvé que notre approche a du sens en quantifiant les erreurs. Nous avons démontré d'une part que les erreurs numériques sont toujours bornées supérieurement dans le DTMC et avons montré d'autre part, qu'il existe une relation de bisimulation entre le LMP et le DTMC.

Notre méthode est originale et repousse les limites de l'*évaluation de modèle*, notamment l'explosion combinatoire de l'espace d'états ou de chemins dans la vérification de systèmes probabilistes infinis.

Abstract

Systems verification is nowadays a major issue and various techniques verify formally critical systems for which correction must be ensured. The focus of this master's thesis is on one of the most used and most effective systems verification techniques, *model-checking*.

Conceptually, to apply *model-checking* to a system, we first abstract its behavior in the form of a transitions system, the model. Then, we formulate a system property of interest in a temporal logic. Finally, a software called *model-checker* is used to verify automatically if the *model* satisfies the property.

In this paper, we want to check reachability property in the probabilistic models called labelled Markov process (LMP) and which have possibly an uncountable set of states. Unfortunately, the model-checker CISMO dedicated to a family of LMP does not handle reachability properties and no other tool can verify LMP.

To improve CISMO and achieve our goal, we first made more expressive its properties specification logic so that it can express reachability property on LMP. These properties express the fact that a desired state in a system can be reached with a certain probability. Secondly, we implemented in CISMO a new approach for the verification of a family of reachability properties. This is a contribution to the evolution of the probabilistic verification.

We use the *mean theorem* to prove that, for any LMP acceptable by CISMO and for any reachability property, there is a discrete time process (DTMC) equivalent to the LMP according to the average reachability and on which we can apply known algorithms for probabilistic systems which have a countable set of states. The DTMC is constructed in such a way that we can infer the LMP satisfies the reachability property, if and only if the DTMC also satisfies it. Theoretically, our approach gives a precise final result and we prove it.

At implementation, since the DTMC is subjected to numerical errors the result can be false, as expected. We use a numerical integration method to determine the transitions probabilities in the DTMC. Despite the errors that can affect the outcome of a verification, we have shown that our approach makes sense at implementation by quantifying the errors. We have shown on one hand that numerical errors are always bounded from above in the DTMC and we established, on the other hand, bisimulation relations between LMP, DTMC constructed theoretically, and DTMC generated algorithmically with errors.

Our method is original and pushes the limits of *model-checking*, especially combinatorial explosion of the states space or paths in the verification of infinite probabilistic systems.

Table des matières

Liste des figures

«Autant on use du temps à redorer son maquillage intellectuel, autant on renaît de ses cendres. Je suis l'espèce de la famille des hominidés la plus sublime de la planète terre capable d'agir, au jour le jour, sur mon moi et de sanctionner mes actes. C'est le cycle de résurrection quotidienne de l'homo sapiens que je suis.»
Gildas Kouko.

Remerciements

Arrivé au terme de la rédaction de ce mémoire, il m'est particulièrement agréable d'exprimer ma gratitude et mes remerciements à tous ceux qui, par leur enseignement, leur soutien et leurs conseils, m'ont aidé à sa réalisation.

Ma gratitude va d'abord à ma co-directrice de recherche, Josée Desharnais, et co-directeur de recherche, François Laviolette, qui m'ont honoré de leur confiance en mes habiletés. J'ai particulièrement été impressionné par leur enthousiasme pour ce sujet, leurs qualités scientifiques et humaines. De plus, leur patience, leur écoute attentive et leurs nombreuses lectures et corrections de mes textes m'ont permis de beaucoup apprendre et de m'améliorer en rédaction. Puissent ces lignes soient l'expression de ma plus profonde reconnaissance. Merci à vous.

Ensuite, je manifeste ma gratitude à Pierre Marchand et sa famille pour leur soutien incessant et leur grande attention.

Je témoigne ma profonde gratitude à Alain Capo Chichi pour son soutien incommensurable. Merci Alain et puisse ce travail être compté parmi les fruits de vos actes.

Je voudrais également remercier mes amis et mes collègues qui m'ont aidé de différentes façons, de par leur affection et leurs encouragements entre autres.

Enfin, je rends hommage aux membres de ma famille pour leur soutien tout au long de mes études. Ce travail est le vôtre.

Chapitre 1

Introduction

Les systèmes informatiques se trouvent pratiquement partout de nos jours : pilotage automatique des avions et des fusées, contrôle de feux de signalisation, télécommunication, centrales nucléaires, contrôle d'ascenseur optimisant le temps d'attente, etc. L'automatisation via les systèmes informatiques est donc inscrite dans l'agenda de la modernisation. Les systèmes informatiques occupent alors une place de plus en plus importante dans la vie quotidienne et on les utilise davantage pour des tâches sensibles et critiques comme dans les systèmes d'aide à la décision médicale. Les tâches qu'il gèrent sont devenues de plus en plus complexes et critiques à telle enseigne que la moindre défaillance peut entraîner de graves dégâts tant financiers qu'humains.

Malheureusement, les médias rapportent souvent des erreurs de conception coûteuses. Un exemple malheureux est le bogue notoire du dysfonctionnement dans l'unité de calcul en virgule flottante du Pentium, qui a porté un coup sévère à l'image de Intel. En effet, en octobre 1994, le professeur Thomas Nicely fut saisi de perplexité. Les très gros calculs qu'il effectuait, dans son domaine de recherche, donnaient des résultats totalement contradictoires avec des calculs antérieurs. Cela venait-il de son compilateur ? Celui qu'il utilisait manifestait, il est vrai, un comportement parfois erratique. Il avait donc réécrit ses programmes en fonction de ce problème et avait relancé ses calculs. Mais il fut surpris de constater que les calculs effectués sur l'un de ses ordinateurs, équipé d'un microprocesseur Pentium donnaient un résultat différent sur d'autres machines. Persévérant, le mathématicien découvrit alors le pot aux roses : le microprocesseur Pentium effectuait lui-même parfois des divisions fausses. Ce bogue du Pentium n'est pas le seul raté informatique célèbre. Pour en savoir plus, le lecteur peut consulter [1].

Le développement de gros programmes informatiques s'est donc souvent accompagné

d'une prolifération de bogues et d'erreurs d'exécutions. Le besoin de méthodes efficaces de validation de programmes se fait alors criant. Autrement dit c'est une nécessité impérieuse pour les concepteurs de programmes informatiques de disposer de méthodes rigoureuses leur permettant de s'assurer que les applications remplissent bien les fonctions qui leurs sont attribuées et de détecter les éventuelles erreurs dans leurs programmes avant de les mettre en service. Il est bien évidemment préférable de détecter le plus tôt possible les erreurs pour que leur correction soit plus simple et moins coûteuse. Au cours des dernières années, plusieurs erreurs qui se sont produites auraient pu être évitées si une meilleure vérification avait été effectuée.

La vérification d'un système peut se faire directement sur le système réel ou sur une représentation abstraite de celui-ci, qu'on appelle *modèle*. Le modèle est un système de transitions, c'est-à-dire un graphe orienté dans lequel les sommets sont appelés états. Il existe quatre grandes classes de techniques de vérification : la *simulation*, les *tests*, la *démonstration de théorèmes* et la vérification par *évaluation de modèle* (en anglais, model-checking). Les deux dernières techniques sont des méthodes formelles, c'est-à-dire qu'elles exploitent des fondements mathématiques non seulement pour décrire un système dans un langage formel, mais aussi pour démontrer qu'il respecte un ensemble de comportements désirés.

Lorsqu'on opte pour la simulation ou les tests, on effectue la vérification d'un programme en observant les sorties obtenues sur certaines entrées. La simulation s'effectue sur le modèle tandis que les tests sont faits sur le système réel. La simulation et les tests sont deux techniques qui donnent de bons résultats. Toutefois, ils permettent de détecter des erreurs sans pouvoir en assurer l'absence car il est rarement possible pour un programme complexe de vérifier toutes les entrées possibles, de parcourir tous les chemins possibles et il peut être très long de connaître la sortie associée à une entrée.

La démonstration de théorèmes [2, 3] quant à elle, est une approche basée sur les preuves. Le système à vérifier est décrit par un ensemble de formules Γ et la spécification des propriétés désirées du système, est elle aussi décrite par un ensemble Φ de formules. Ensuite le démonstrateur essaye de prouver syntaxiquement chaque propriété dans Φ à partir de Γ. La démonstration de théorèmes est robuste et automatisable en partie. Chaque preuve peut nécessiter une intervention humaine et du fait qu'elle n'est pas entièrement automatique, il n'est pas possible d'obtenir une borne sur le temps et la mémoire qu'il faudra pour faire une démonstration.

Pour détecter des erreurs dans des systèmes, la communauté scientifique s'est long-

temps appuyée sur la mécanique des tests, des simulations et des démonstrations de théorèmes. Mais dès le début des années 1980, elle s'intéresse à l'évaluation de modèle, une technique exhaustive et en grande partie automatique. Le travail de l'ingénieur se limite à la construction d'un modèle du système et à la formalisation des propriétés à vérifier. L'évaluation de modèle permet d'assurer algorithmiquement avec des outils du domaine de la logique qu'un modèle satisfait une propriété. C'est d'ailleurs à cette méthode que nous nous intéressons dans ce mémoire.

L'évaluation de modèle s'effectue en trois étapes. La première consiste à modéliser le système à l'aide d'un langage formel particulier utilisable par un outil logiciel de vérification appelé *vérificateur* (*model-checker*, en anglais). Elle conduit à l'obtention d'un modèle du système qui réduit la complexité de celui-ci en éliminant les détails qui n'influencent pas son comportement de façon significative. Le modèle reflète alors ce qu'un concepteur croit important pour la compréhension et la prédiction d'un phénomène. Parlant de modèle, si le nombre d'états de celui-ci est dénombrable, on dit qu'il s'agit d'un système fini, sinon il est infini. Pour mieux distinguer un modèle fini d'un modèle infini, nous illustrons les deux modèles brièvement.

La figure 1.1 illustre le modèle d'un système de chauffage vu sous l'angle d'un système fini. Le modèle dispose de 3 états : s_0, s_1 et s_2. Chaque état est identifié par une étiquette appelée *proposition atomique*. Chaque *proposition atomique* qualifie la température que dégage le système dans l'état qu'elle identifie. Ainsi, dans s_0, le système fourni une température chaude. Les arcs orientés qui relient les états traduisent les transitions possibles. On ne peut pas alors en cas de panne, transiter de s_2 à s_1 puisqu'il n'existe pas d'arc orienté de s_2 vers s_1. La flèche entrante sur s_1 indique qu'il est l'état initial du système.

Dans ce mémoire, nous focalisons sur les systèmes probabilistes infinis. Le qualificatif «probabiliste» vient indiquer que nous intégrons les probabilités dans la modélisation des systèmes traités. Dans ce cadre, nous codons la probabilité de réalisation d'une transition entre deux états donnés, plutôt que la seule existence d'une telle transition. Autrement dit nous pondérons chaque transition par un réel compris entre 0 et 1 qu'on appelle probabilité de transition. La figure 2.4 de la page 22, expliquée plus à la section 2.2, illustre un modèle probabiliste infini d'une distributrice de café modélisant les quantités de café versable. Le modèle montre que la machine peut verser de 1 à 500 millilitres (ensemble de réels représentant les états) de café. x est une variable désignant une quantité versable et elle a valeur d'état ultime d'une transition. Ainsi, si nous

supposons que la distributrice a versé du café et quelle doit en servir à nouveau, alors la probabilité qu'elle transite dans un état où elle sert 300 millilitres vaut $\frac{300}{500} = \frac{3}{5}$. Nous reviendrons plus loin sur les modèles en informatiques et préciserons leur définition formelle.

La première étape de l'évaluation de modèle est difficile et souvent cruciale pour la pertinence des résultats qu'on obtient par la suite. Il n'existe pas de méthode universelle pour modéliser un système ; c'est un travail d'ingénieurs qualifiés ayant à la fois une bonne connaissance du système et de modèles en informatique pouvant être utilisés car il existe plusieurs manières d'analyser et de représenter le comportement d'un système. Pour diverses raisons, certains optent pour l'analyse des traitements internes qu'effectue le système. Mais nous, nous concentrons plutôt nos études sur les interactions que celui-ci peut avoir avec son environnement, c'est-à-dire, les systèmes qui l'entourent et ses utilisateurs. Dans une telle perspective, le système est dit réactif puisqu'il réagit à son environnement par un comportement directement déterminé par les stimulus auxquels il est exposé. La connaissance des systèmes réactifs par le monde informatique remonte aux années 1970. Ils diffèrent des systèmes transformationnels qui sont des programmes classiques disposant de leurs entrées dès leur initialisation et délivrant leurs résultats lors de leur terminaison.

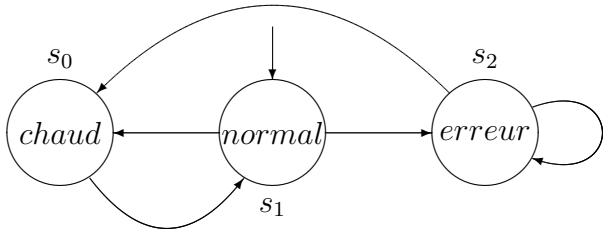


FIGURE 1.1 – Exemple de modèle : un système de chauffage.

La deuxième étape de l'évaluation de modèle est la spécification des propriétés du système qu'on souhaite vérifier. On les définit à l'aide d'un langage (sémantique et syntaxe) particulier. En général, on utilise une logique temporelle qui permet de spécifier une propriété dans le temps. Par exemple dans la logique temporelle linéaire, explicitée à la section 2.5.1, en considérant la figure 1.1 et en utilisant les combinateurs booléens $\Rightarrow$ (implication logique) et $\neg$ (négation), $erreur \Rightarrow \neg chaud$ représente la propriété «*s'il y a une erreur, alors il ne fait pas chaud*». Cette propriété est vraie dans tous les états du modèle car les propositions atomiques *chaud* et *erreur* n'identifient pas un

même état. Un enjeu évident est de réussir à spécifier totalement le comportement d'un système, ce qui n'est pas réalisable de façon automatique. Pour cela, on se cramponne plus à deux grandes familles de propriétés que sont : la vivacité ou vitalité qui exprime que «*quelque chose de bon arrivera assurément*» et la sûreté qui exprime que «*quelque chose de mauvais n'arrive jamais*». Dans un contexte de vérification probabiliste, à titre illustratif, la propriété «*la probabilité qu'une erreur survienne est moins que* 10%» en est une de sûreté et la propriété «*une ressource demandée sera rendue disponible avec une probabilité de* 20%» en est une de vitalité. Ces formes de propriétés expriment le fait qu'un état souhaité est toujours atteignable ou que des événements satisfaisants se produiront avec une certaine probabilité. On parle donc de propriété d'atteignabilité et c'est le type de propriété qui nous tiendra en haleine dans ce mémoire.

La dernière étape de l'évaluation de modèle est la vérification proprement dite. On utilise pour cela un vérificateur qui prend en entrée un modèle et une propriété et informe si le modèle satisfait ou pas la propriété. La plupart des algorithmes de vérification vont même un peu plus loin en fournissant un contre-exemple de la propriété quand celle-ci ne peut pas être satisfaite par le modèle (c'est la notion de trace d'erreur). Parfois, la vérification peut ne pas être effectuée simplement parce que le vérificateur n'a pas assez de mémoire à disposition pour mener à bien ses traitements. On dit dans ce cas qu'on est sujet au problème de l'explosion combinatoire de l'espace d'états [4].

Malgré l'efficacité de la vérification par évaluation de modèle, le problème de l'explosion combinatoire de l'espace d'états tend à limiter son champ d'utilisation, surtout pour les systèmes infinis. Par exemple, les techniques actuelles ne sont guère capables de construire des preuves pour des systèmes dépassant quelques dizaines de millions d'états, ce qui est souvent largement insuffisant. Plusieurs approches de vérification visent à repousser les limites de l'explosion combinatoire, un but dans ce mémoire. L'approche la plus répandue ces dernières années est l'évaluation de modèle symbolique [5] qui utilise les diagrammes de décisions binaires à terminaux multiples (MTBDD, de l'anglais *Multi Terminal Binary Decision Diagrams*) pour optimiser la vérification et la représentation de modèles en mémoire.

En résumé, les ingrédients de évaluation de modèle sont : un modèle, des propriétés et un vérificateur souvent dédié à une classe donnée de systèmes en raison de la distinction entre systèmes finis et systèmes infinis. La construction d'un vérificateur exige de faire des choix, notamment celui des formalismes pour représenter des modèles et des propriétés, des structures de données et des algorithmes pour vérifier la satisfaction de

propriétés dans les modèles. Un avantage de l'évaluation de modèle, relativement aux autres méthodes de vérification, est son automatisation presque complète. Son efficacité dépend en général de la taille de l'espace d'états accessibles. Elle trouve ses limites dans les ressources de l'ordinateur pour manipuler les états accessibles mais des techniques d'abstractions, éventuellement guidées par un utilisateur, peuvent être utilisées pour améliorer l'efficacité de ses algorithmes. Un autre avantage crucial des vérificateurs est leur aptitude à trouver les contre-exemples qui ne satisfont pas une propriété. Ceux-ci sont fort utiles à la compréhension des situations d'erreurs et à la correction lors des phases amont de la conception de tout système.

Nous abordons les objectifs de notre projet de recherche à la prochaine section.

1.1 Objectifs du mémoire

Le champ général de notre projet de recherche est la vérification de systèmes probabilistes interactifs, plus particulièrement, la vérification par évaluation de modèle. Dans ce projet, nous focalisons sur les systèmes probabilistes réactifs à espace d'états infini possiblement dénombrable. De nos jours, seuls les systèmes finis sont très connus du monde informatique et il existe plusieurs vérificateurs qui leur sont dédiés. À notre connaissance, seul le vérificateur CISMO (*Continuous State Space Model checker*), basé sur la théorie des processus de Markov étiquetés (LMP, de l'anglais *Labelled Markov Process*) développée par Desharnais et al, vérifie les *modèles probabilistes* à espace d'états continu.

CISMO a été développé en JAVA par Richard [6] lors de sa maîtrise puis amélioré par Paquette [7] dans le cadre de sa maîtrise également. Il peut vérifier la classe de LMP restreinte aux contraintes et langages utilisés pour décrire les LMP dans le vérificateur. Nous décrivons sommairement CISMO à la section 2.3. Actuellement, il ne gère pas les propriétés d'atteignabilité et aucun autre outil ne peut vérifier les LMP. Pour améliorer CISMO, nous choisissons axer nos recherches sur l'évaluation de propriétés d'atteignabilité dans les LMP. Nos contributions au chapitre 3 dans CISMO se résument essentiellement à :

1. élargir sa logique $\mathtt{L}_0$ en une logique $\mathtt{L}_1$ permettant d'énoncer des propriétés d'atteignabilité ;
2. implémenter une approche de génération de système probabiliste fini à partir d'un LMP et d'une propriété d'atteignabilité. Notre choix de transformer un LMP en

un système fini, participe de notre souci d'utiliser les avancées algorithmiques connues aux systèmes finis pour repousser les limites de l'évaluation de modèle.

Au-delà de l'impact positif de nos résultats de recherches sur CISMO, c'est aussi la vérification de systèmes probabilistes infinis qui connaît une fois encore un progrès. La prochaine section détaille notre contribution à l'avancée de la vérification probabiliste.

1.2 Méthodologie et contributions

Pour atteindre les objectifs de ce mémoire, nous explorons quelques modèles probabilistes en informatique et présentons dans un premier temps deux récentes approches de vérification probabiliste par évaluation de modèle. Il s'agit des méthodes proposées dans les articles [8] et [9] publiés en 2010.

L'approche proposée dans [8] permet de générer un système probabiliste fini à partir d'un système probabiliste infini en vue de calculer une probabilité d'atteignabilité. Celle présentée dans [9] est une méthode de vérification multi-objectifs. Elle permet de vérifier à la fois deux propriétés de sûreté dans un système probabiliste fini. Les deux approches garantissent toutes de bonnes performances algorithmiques dans l'analyse de l'erreur en vérification probabiliste. Mais est-ce possible d'appliquer ces nouvelles techniques aux LMP qui peuvent avoir un ensemble d'états infini ? En nous inspirant des deux approches, nous arrivons à la conclusion que cette question trouve une réponse immédiate dans la transformation d'un LMP en système fini.

Les LMP représentables pour CISMO sont au coeur de ce mémoire et nous focalisons sur la vérification de propriétés d'atteignabilité. Étant donné que CISMO ne gère pas celles-ci et qu'aucun autre outil ne peut vérifier les LMP, nous avons le choix entre deux stratégies. La première requiert un travail énorme car elle consiste à refaire, dans CISMO, et pour les LMP, tout ce qui a été fait pour les systèmes finis dans d'autres outils comme par exemple PRISM [10]. La seconde approche, étudiée également, est d'approximer un LMP à vérifier par un système probabiliste fini et de faire la vérification de propriétés sur celui-ci à l'aide d'un vérificateur dédié aux systèmes finis, comme PRISM également. Nous proposons une telle approche pour les LMP représentables pour CISMO et pour une certaine famille de propriétés d'atteignabilité.

À travers notre approche de vérification, nous prouvons que pour un LMP acceptable par CISMO et pour une propriété d'atteignabilité donnée, il existe un système probabiliste fini équivalent au LMP de départ. Notre contribution réside dans l'usage d'un

fondement mathématique, le théorème de la moyenne d'une fonction continue sur un intervalle. Il nous permet de transformer un problème difficile à résoudre en un autre beaucoup plus simple à solutionner. Nous nous servons du théorème de la moyenne pour calculer des probabilités moyennes de transition dans les modèles de LMP que nous traitons et en raison de son usage, nous qualifions notre approche d'atteignabilité moyenne. En fait, les probabilités de transitions dans nos modèles de LMP sont des fonctions ayant la forme d'un polynômes.

Notre approche est originale, applicable dans un contexte de distribution de probabilités continue comme discrète et nous l'avons implémentée dans CISMO. Étant donné qu'il existe plusieurs vérificateurs performants pour les systèmes probabilistes finis, à partir d'un LMP et d'une propriétés d'atteignabilité, nous nous limitons à la génération du système fini dans un format compréhensible et traduisible dans d'autres formalismes. La vérification ultime étant faite dans un vérificateur pour système fini, nous inférons la satisfaction de la propriété d'atteignabilité par le LMP à partir du fait que le système fini la satisfait. Notre stratégie de vérification garantit donc une indépendance de CISMO vis-à-vis des autres vérificateurs.

Dans la plupart des cas, notre approche permet de définir un système probabiliste fini qui, pour ce qui est de la propriété étudiée, est équivalent au LMP de départ de point de vue atteignabilité moyenne. Quand le système fini n'est qu'une approximation, c'est-à-dire assorti d'erreurs il se peut que le résultat de la vérification soit ultimement erroné. Nous démontrons que notre résultat est exact théoriquement ; bien sûr, lors de l'implémentation, il se peut que des imprécisions numériques que nous avons quantifiées faussent le résultat, mais ceci fait partie de toute analyse de systèmes avec valeurs numériques.

Nous abordons l'organisation de notre mémoire à la prochaine section.

1.3 Organisation du mémoire

Nous articulons ce mémoire autour de cinq chapitres dont le premier est une introduction aux principes fondamentaux de la vérification par évaluation de modèle et le dernier la conclusion de notre mémoire.

Au chapitre deux (2), nous introduisons des notions préliminaires à l'étude de modèles probabilistes en informatique et en présentons quelques variantes : chaînes de Markov à temps discret, processus décisionnels de Markov, chaînes de Markov à temps continu,

automates temporisés et processus de Markov étiquetés. Cette exploration de modèles probabilistes amène le lecteur à bien distinguer un système fini d'un système infini et le prépare à bien comprendre les récentes approches de vérification qui nous inspirent dans nos recherches et que nous présentons dans ce chapitre également.

Au chapitre trois (3), nous élargissons la logique de CISMO aux propriétés d'atteignabilité et décrivons l'approche d'atteignabilité moyenne dans les LMP. À partir d'une propriété d'atteignabilité et d'un LMP, nous montrons comment construire théoriquement et algorithmiquement une chaîne de Markov à temps discret pour décider de la satisfiabilité de la propriété d'atteignabilité dans le LMP. Des imprécisions numériques pouvant créer parfois une différence entre chaîne de Markov à temps discret théorique et celle obtenue algorithmiquement, nous tâchons de situer les erreurs numériques dans nos travaux. Aussi, nous présentons une brève comparaison des deux modèles en utilisant des notions connues pour l'étude comparative de modèles probabilistes.

Au chapitre quatre (4), nous présentons, des exemples à l'appui, les détails de l'implémentation de l'approche d'atteignabilité moyenne dans CISMO.

Le prochain chapitre aborde l'état de l'art de notre mémoire.

Chapitre 2

État de l'art

Dans ce chapitre, nous présentons d'abord les notions préliminaires à l'étude de modèles probabilistes en informatique et définissons formellement quelques variantes des chaînes de Markov. Il s'agit notamment des chaînes de Markov à temps discret (DTMC, de l'anglais *Discrete Time Markov Chains*), des processus décisionnels de Markov (MDP, de l'anglais *Markov Process Decision*), des chaînes de Markov à temps continu (CTMC, de l'anglais *Continuous Time Markov Chains*), des automates temporisés (TA, de l'anglais *Timed Automata*) et des processus de Markov étiquetés (LMP, de l'anglais *Labelled Markov Processes*). Notons que les chaînes de Markov à temps continu et les automates temporisés ne sont pas d'une importance majeure dans ce mémoire. Nous les évoquons pour que le lecteur s'approprie d'autres modèles et appréhende mieux les différences entre un modèle à temps continu et un modèle à temps discret. À l'opposé, une bonne connaissance des chaînes de Markov à temps discret, des processus décisionnels de Markov et des processus de Markov étiquetés est requise. Les récentes techniques de vérification qui nous ont inspirées dans nos recherches et l'approche d'atteignabilité moyenne utilisent des notions que nous avons définies par rapport aux chaînes de Markov à temps discret et aux processus décisionnels de Markov. En plus, l'approche d'atteignabilité moyenne transforme un processus de Markov étiquetés en une chaîne de Markov à temps discret.

Ensuite, nous abordons les systèmes probabilistes infinis et présentons le vérificateur CISMO. À travers une synthèse explicite, nous définissons le modèle de LMP représentable pour CISMO et passons en revue non seulement sa logique de spécification de propriétés mais aussi les structures de données qui le supportent.

Enfin, nous présentons les deux récentes approches [8, 9] de vérification de propriétés par

évaluation de modèle. Celle exposée dans [9] partage un point commun avec l'approche d'atteignabilité moyenne que nous proposons, celui d'approximer un système infini par un système fini pour évaluer des propriétés d'atteignabilité. L'approche présentée dans [8] quant à elle, a fortement nourri nos réflexions dans la recherche des solutions pour atteindre les objectifs de notre mémoire. Nous reviendrons à la section 2.6 sur le lien des approches exposées dans [8, 9] avec notre travail. Il est bien vrai que les deux approches nous ont inspiré dans nos recherches, mais soulignons qu'elles ne sont pas totalement requises pour comprendre notre contribution à la vérification probabiliste.

2.1 Introduction aux modèles probabilistes

Un champ important de la théorie des probabilités est celui des chaînes de Markov. Une chaîne de Markov est une suite d'événements aléatoires dans le temps. La propriété fondamentale de celle-ci, dite propriété de Markov, est que son évolution future ne dépend du passé qu'au travers de l'état courant. Nous revenons sur les chaînes de Markov à la section 2.1.2 avec l'exemple 2.1.1 pour mieux étayer quand un système est markovien ou non. Mais notons déjà que dans l'analyse de systèmes informatiques, le modèle des chaînes de Markov est le modèle probabiliste de base comme le mentionne [11].

Dans ce mémoire, nous faisons de la vérification probabiliste par évaluation de modèle. Elle intègre l'analyse probabiliste à l'évaluation de modèle classique pour en faire un seul outil de vérification comme le mentionne [12]. Les probabilités interviennent dans l'analyse d'un système pour deux raisons. La première est qu'elles peuvent être utilisées pour abstraire les comportements aléatoires ou non, surtout en présence d'un système infini où plusieurs paramètres doivent être pris en compte dans sa modélisation. La seconde raison est la non-maîtrise de l'environnement externe au système. Ceci fait qu'on ne peut décrire le comportement exact de l'environnement dans lequel le système est plongé. On a donc recours aux lois probabilistes pour appréhender le comportement prévisible de l'environnement partiellement connu.

Les vérificateurs usuels prennent en entrée une description d'un modèle et une propriété, typiquement une propriété exprimée dans une logique temporelle [13] et laisse savoir si le modèle la vérifie. Dans le cas de l'évaluation de modèle probabiliste, le modèle est nécessairement une variante des chaînes de Markov. Un espace de probabilités induit sur les comportements du système, permet de calculer la probabilité de l'occurrence de certains événements pendant son activité. Ceci permet d'établir des propriétés

quantitatives sur le système, en complément des propriétés qualitatives habituellement établies en évaluation de modèle classique.

Décrivons maintenant le contexte de notre travail en abordant les notions préliminaires à l'étude de modèle probabilistes.

2.1.1 Une goutte de la théorie des probabilités

Nous présentons dans un cadre formel, une base théorique suffisante pour définir formellement par la suite des modèles probabilistes. Nous ne prétendons pas donner une introduction exhaustive de la théorie des probabilités. L'ouvrage [14] est une référence en la matière.

Ensemble des observables versus événements

Pour étudier un phénomène probabiliste, on considère un ensemble particulier Ω contenant tous les résultats possibles des observations. On dit alors que Ω est l'ensemble des observables et tout sous-ensemble de l'ensemble des observables est un événement auquel on peut associer une probabilité d'apparition. Dans le cas général, on n'assigne pas de probabilité à tous les événements possibles, mais à une collection d'événements.

Sur quel ensemble définir des probabilités ?

Pour examiner un phénomène probabiliste, on souhaite disposer d'une collection qui permettrait de définir la probabilité de Ω qui vaudrait 1. On veut que la collection soit stable par complémentaire afin que lorsqu'on dispose de la probabilité p d'un événement, on puisse calculer la probabilité du complémentaire qui vaudrait $1 - p$. On souhaite également disposer d'une collection qui permet d'évaluer la probabilité d'une union d'événements lorsqu'on connaît la probabilité de chaque événement d'un ensemble d'événements. Formellement, on définit une probabilité sur une collection satisfaisant les propriétés de σ-algèbre ou tribu comme suit :

Définition 2.1.1. Soient Ω et Σ deux ensembles tels que $\Sigma \subset P(\Omega)$. Σ est une σ-algèbre si elle satisfait les propriétés suivantes :

1. $\emptyset \in \Sigma$.
2. Si $A \in \Sigma$ alors, son complémentaire $A^c = \Omega \setminus A$ est aussi dans Σ.
3. Si on a une suite finie ou dénombrable $(A_i)_{i\in\mathbb{N}}$ d'éléments de Σ, alors $\bigcup_{i\in\mathbb{N}} A_i \in \Sigma$.

En exprimant l'intersection à l'aide des opérations d'union et de complémentaire, on déduit que $\Omega \in \Sigma$ et qu'une σ-algèbre est fermée par l'intersection dénombrable. Bien évidemment qu'étant donné un ensemble d'observables Ω, il peut exister plusieurs σ-algèbres associées.

Lorsqu'un ensemble d'observables Ω est muni d'une σ-algèbre Σ, on dit que (Ω, Σ) est un espace mesurable et tout élément de Σ est appelé ensemble mesurable. Nous utilisons la notion de σ-algèbre sur des ensembles mesurables dans les définitions 2.1.4 et 2.4.2 abordant respectivement les notions de transition probabiliste partielle et de probabilité d'ensemble de chemins dans un modèle probabiliste. Dans ce mémoire, nous définissons l'espace d'états probabiliste des LMP (*Labelled Markov Process*) par des réels sur lesquels nous calculons des probabilités d'exécution de transitions et effectuons des opérations usuelles ensemblistes. Pour ce fait, nous nous restreignons à l'espace mesurable des nombres réels avec la σ-algèbre borélienne [15]. La tribu borélienne est la plus petite σ-algèbre sur $\mathbb{R}$ contenant tous les intervalles. Relativement à la notion d'espace mesurable, nous définissons ci-dessous la notion de fonction mesurable et l'utilisons dans la description des LMP plus loin.

Définition 2.1.2. On appelle fonction mesurable d'un espace mesurable (Ω, Σ) dans un espace (Ω', Σ') toute fonction $\mu : \Omega \longmapsto \Omega'$ telle que : $\forall E \in \Sigma', \mu^{-1}(E) \in \Sigma$

μ^{-1} représente l'image réciproque par μ, c'est-à-dire $\mu^{-1}(E) = \{x \in \Omega \mid (\mu(x) \in E)\}$. μ assure donc que l'image réciproque de tout ensemble mesurable de Σ' est aussi un ensemble mesurable appartenant à Σ.

Nous avons introduit la notion d'espace mesurable dans le but d'y associer une fonction particulière appelée mesure qui attribue une valeur à chaque ensemble mesurable. Dans ce mémoire, nous considérons la mesure de poids total 1, appelé mesure de probabilité, que nous définissons ci-dessous.

Définition 2.1.3. On appelle mesure de probabilité, ou distribution sur un espace mesurable (Ω, Σ) toute application μ de Σ dans $\mathbb{R}^+$ telle que :

1. μ est défini pour tout élément de Σ et $\mu(\emptyset) = 0$,
2. $\mu(\Omega) = 1$,
3. Pour tout ensemble dénombrable $\{A_i \mid i \in \mathbb{N}\}$ d'éléments de Σ disjoints deux à deux, on a : $\mu(\bigcup_{i \in \mathbb{N}} A_i) = \sum_{i \in \mathbb{N}} \mu(A_i)$.

Nous notons $D(\Omega)$ l'ensemble des distributions sur Ω et le triplet (Ω, Σ, μ) est appelé espace de probabilité.

L'espace d'états mesurable de nos LMP étant défini par des réels nous calculons des mesures d'intervalles. Pour cela, nous définissons au 2.1.4 la notion de transition probabiliste partielle utilisée dans la définition 2.2.1 des LMP. Dans la définition 2.1.4, la notation $\mu(x, .)$ représente la fonction de transition probabiliste partielle μ où le paramètre x est fixe et le second libre. La fonction μ est dite partielle parce que son image n'est pas $\{0, 1\}$ mais plutôt $[0, 1]$, ce qui permet des situations où $0 < \mu(x, E) < 1$. Le fait que la probabilité soit strictement supérieure à zéro sous-tend la possibilité d'une transition de x vers E, mais le fait qu'elle soit strictement inférieure à 1 traduit la possibilité qu'un LMP ne réagisse pas à un stimuli de son environnement. Nous revenons sur le détail de ce cas de figure à la section 2.2 consacré au système infini.

Définition 2.1.4. Soit (Ω, Σ) un espace mesurable. Une fonction de transition probabiliste partielle sur (Ω, Σ) est une fonction $\mu : \Omega \times \Sigma \to [0, 1]$ telle que pour chaque $x \in \Omega$, la fonction $\mu(x, .)$ est une mesure de probabilité et pour tout $E \in \Sigma$, la fonction $\mu(., E)$ est mesurable.

$\mu(x, E)$ est une probabilité conditionnelle. Elle représente la probabilité d'aller dans un état dans $E \in \Sigma$ sachant que le système se trouve au préalable dans l'état x. Mais en aucun cas, la probabilité ne dépend des états précédents (propriété des chaînes de Markov).

Les notions préliminaires à l'étude de modèles probabilistes étant décrites, nous abordons la description de quelques modèles probabilistes finis.

2.1.2 Quelques variantes de chaînes de Markov à espace d'états discret

L'analyse des systèmes probabilistes a connu ces dernières années un essor considérable et plusieurs méthodes de vérification ont été développées à l'aide d'outils classiques des théories de probabilités. L'outil le plus connu est la chaîne de Markov introduite par le mathématicien russe Andrei Andreyevich Markov en 1906, alors qu'il étudiait les probabilités d'apparition des lettres de l'alphabet dans les textes littéraires. Il fut en effet amené à évaluer les probabilités d'apparition des lettres en fonction des lettres précédentes, ce qui l'a conduit à définir la propriété des chaînes de Markov. Il a élaboré

un modèle où la connaissance de l'état courant d'un système est suffisante pour prédire l'évolution de celui-ci. Pour en savoir plus sur les chaînes de Markov et ses notions applicatives, le lecteur peut consulter [16, 17]. Cependant, pour se faire rapidement une idée de ce qu'est un système qui respecte la propriété de Markov, nous donnons un exemple ci-dessous.

Exemple 2.1.1. Prenons l'exemple d'un train pouvant se déplacer entre Québec et Toronto. Soit il part de Québec et va vers Toronto, soit il fait le trajet en sens inverse. Supposons que chaque état du système (trajet modelisé) corresponde à la dernière gare dans laquelle le train s'est arrêté. L'information contenue dans l'état courant est alors insuffisante pour prédire l'état suivant. En effet, il est également nécessaire de connaître la direction du train. Si les deux dernières gares, c'est-à-dire les deux derniers états, sont connues, il est possible de déduire le sens du train. Le système n'est toutefois pas markovien puisqu'il est nécessaire de connaître l'état courant et l'état précédent afin de prédire l'état suivant. En revanche, si un état est constitué du nom de la dernière gare et du sens de déplacement du train ou bien des deux dernières gares, alors nous sommes en mesure de prévoir le prochain arrêt à partir de l'état courant et le système est donc markovien.

Classiquement, on distingue trois axes de classification des variantes de chaînes de Markov : le temps, la nature de l'espace d'états et la nature de la distribution des probabilités sur les états. D'abord, lorsqu'on considère le temps comme un ensemble dénombrable, la chaîne de Markov est dite à temps discret et chaque transition dure une unité de temps. Ainsi, dans tout autre cas, le temps est inclus dans $\mathbb{R}$ et on parle de chaîne de Markov à temps continu. Ensuite, si la nature de l'espace d'états est dénombrable, on parle de chaîne de Markov à espace d'états discret ou fini. Dans le cas contraire, elle est à espace d'états infini. Enfin, on distingue les chaînes de Markov avec une distribution de probabilités continue (loi normale, loi exponentielle, loi uniforme, etc.) sur les états de celles ayant une distribution de probabilités discrète sur les états. Dans ce mémoire, nous consacrons nos recherches aux LMP assortis d'un espace de temps discret et d'une distribution de probabilités continue, spécifiquement la loi uniforme. Abordons maintenant des définitions formelles de variantes de chaînes de Markov usuelles pour étayer la catégorisation évoquée.

Présentons en premier la chaîne de Markov à temps discret [18] définie formellement à la définition 2.1.5. Dans celle-ci, à chaque changement d'état, le nouvel état est choisi avec une distribution de probabilités discrète. Une chaîne de Markov à temps discret possède

un espace d'états fini et on utilise des étiquettes appelées propositions atomiques pour distinguer les états qui satisfont des propriétés élémentaires. Nous nommons AP (de l'anglais, *Atomic Propositions*) l'ensemble des propositions atomiques utilisées dans les modèles présentés dans ce mémoire.

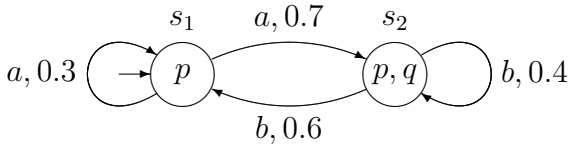
Définition 2.1.5. Une chaîne de Markov à temps discret (DTMC, de l'anglais *Discrete Time Markov Chains*) est un tuple $M = (S, i, \mu, AP, L)$ où :

- S est l'ensemble des états ;
- $i \in S$ est l'état initial ;
- Act est un ensemble fini d'actions ;
- $\mu : S \times Act \to D(S)$ est telle que $\mu(s, a, .)$ est une distribution de probabilité discrète pour tout $a \in Act$ et $s \in S$. Nous écrivons $\mu_a(s, s')$ avec $s' \in S$ plutôt que $\mu(s, a, \nu)$ avec $\nu(s') > 0$ pour désigner la probabilité de transiter de s vers s' en exécutant a ;
- AP est un ensemble de propositions atomiques ;
- $L : S \to 2^{AP}$ est une fonction d'étiquetage qui associe à chaque état $s \in S$, un ensemble $L(s)$ de propositions atomiques.

Notons qu'un DTMC possède un unique état initial et son évolution à partir de tout état est décrite par une seule distribution de probabilité pour une action donnée. On dit alors que c'est un modèle probabiliste *déterministe*.

Dans un DTMC, on connaît a priori les distributions de probabilités qui régissent les transitions. Pour ce fait, pour modéliser un système avec un DTMC, on peut ignorer les actions si celles-ci ne sont sont pas indispensables. La figure 2.1 illustre un DTMC ainsi que sa fonction d'étiquetage et sa matrice de transitons dans laquelle nous ignorons les actions. Dans la figure, p et q sont des propositions atomiques et par exemple, la probabilité de transiter de l'état s_1 vers l'état s_2 en exécutant l'action a vaut 0.7.

Les DTMC sont adéquats pour modéliser les systèmes dans lesquels on est toujours certain de l'effet d'une action exécutée lors d'une transition. Pour prendre en compte le besoin de choisir une action dans l'incertain lors d'une transition dans un système, on utilise une autre variante de chaîne de Markov en lieu et place d'un DTMC. Il s'agit de l'automate probabiliste [19, 20]. Il est similaire aux processus décisionnels de Markov [20, 13] classiquement utilisés en recherche opérationnelle et qui, contrairement aux DTMC sont des modèles probabilistes non déterministes. C'est-à-dire ils peuvent posséder plusieurs états initiaux et plusieurs distributions de probabilités peuvent régir



$$\mu = \begin{pmatrix} 0.3 & 0.7 \\ 0.6 & 0.4 \end{pmatrix}$$

$$S = \{s_1, s_2\}\ i = s_1 \text{ et } AP = \{p, q\}$$

$$L : S \to 2^{\{p,q\}}$$
$$s_1 \to \{p\}$$
$$s_2 \to \{p, q\}$$

FIGURE 2.1 – Exemple d'un DTMC avec ses fonctions μ et L.

leurs transitions pour une action donnée. La prochaine définition formalise un processus décisionnel de Markov.

Définition 2.1.6. Un processus décisionnel de Markov (MDP, de l'anglais (*Markov Decision Process*) est un tuple $M = (S, I, Act, \mu, AP, L)$ où :

- S est l'ensemble des états ;
- $I \subseteq S$ est l'ensemble des états initiaux ;
- Act est un ensemble fini d'actions ;
- $\mu : S \times Act \to 2^{D(S)}$ est telle que $\mu(s, a, .)$ est une distribution de probabilité discrète pour tout $a \in Act$ et $s \in S$. Nous écrivons $\mu_a(s, s')$ avec $s' \in S$ plutôt que $\mu(s, a, \nu)$ avec $\nu(s') > 0$ pour désigner la probabilité de transiter de s vers s' en exécutant a ;
- AP est un ensemble de propositions atomiques ;
- $L : S \to 2^{AP}$ est une fonction d'étiquetage qui associe à chaque $s \in S$, un ensemble $L(s)$ de propositions atomiques.

La figure 2.2 illustre un MDP puisqu'on note deux distributions de probabilités qui décrivent les transitions partant de s_1 pour l'action a. Une distribution est affectée à la transition de s_1 vers s_2 et la seconde aux transitions de s_1 vers s_4 et de s_1 vers s_3 .

Les DTMC et MDP ne représentent pas le temps d'une manière continue. Ils sont donc inefficaces pour modéliser les systèmes manipulant des aspects «temps-réels». Dans la littérature, il existe des travaux [21, 22, 23, 24] appliqués à ce sujet et qui utilisent des

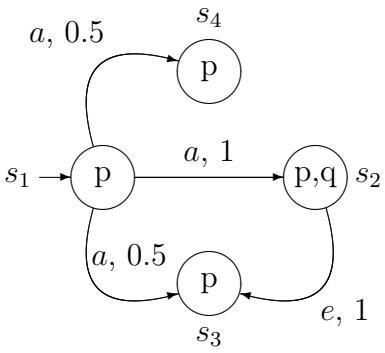


FIGURE 2.2 – Exemple d'un MDP.

variantes de chaînes de Markov appropriées. Les plus usuelles sont les chaînes de Markov à temps continu [21, 22] et les automates temporisés [23, 24]. Nous nous limitons à leur définition formelle dans ce mémoire car elles ne sont pas requises pour comprendre notre contribution plus loin. Nous les évoquons pour que le lecteur s'approprie d'autres modèles et appréhende mieux le contexte dans lequel on utilise les modèles à temps discret.

Une chaîne de Markov à temps continu sert à modéliser un processus indexé par un temps continu et contrairement à un système à temps discret, les transitions peuvent survenir à tout moment. Ci-dessous sa définition formelle.

Définition 2.1.7. Une chaîne de Markov à temps continu (CTMC, de l'anglais *Continuous Time Markov Chains*) est un tuple $C = (S,\, i, T, AP, L)$ où :

- $S, i \in S, AP$ et $L : S \to 2^{AP}$ ont la même définition que dans les DTMC ;
- $T : S \times S \to R_{\geq 0}$ est la matrice des poids de transition.

La matrice T assigne un délai de transition à chaque paire d'états. Les délais de transition suivent une loi exponentielle et une transition ne peut avoir lieu qu'à condition que $T(s, s') > 0$.

Un autre modèle, appelé automate temporisé, sert également à modéliser un processus indexé par un temps continu mais permet d'intégrer des informations quantitatives sur l'écoulement du temps. C'est un automate fini classique muni d'horloges qui évoluent de manière continue et synchrone avec le temps. Chaque transition contient non seulement une garde sur la valeur des horloges décrivant quand la transition peut être exécutée, mais aussi un ensemble d'horloges qui doivent être remises à zéro au terme de celle-ci. Chaque état de contrôle, aussi appelé location, peut contenir un invariant (une contrainte sur les horloges) qui peut restreindre le temps d'attente dans l'état et

donc forcer l'exécution d'une transition. La prochaine définition est celle d'un automate temporisé.

Définition 2.1.8. Un automate temporisé (TA, de l'anglais *Timed Automata*) est un 6-uplet $(L, l_0, X, Inv, T, \Sigma)$ où :

- L est un ensemble fini d'états de contrôle ou de localités ;
- $l_0 \in L$ est la localité initiale ;
- X est un ensemble fini d'horloges ;
- Σ est un alphabet d'actions ;
- $T \subseteq L \times C(X) \times \Sigma \times 2^X \times L$ est un ensemble fini de transitions ; $e = (l, g, a, r, l') \in T$ représente une transition de l vers l' telle que g est la garde associée à e et appartient à $C(X)$ qui contient l'ensemble des contraintes sur les horloges, r est l'ensemble d'horloges devant être remises à zéro et a est une action ;
- $Inv : L \longrightarrow C(X)$ associe un invariant à chaque état de contrôle.

La figure 2.3 illustre un automate temporisé. Nous avons $L = \{l_0, l_1, l_2, l_3\}$ avec l_0 l'état initial, $X = \{x, y\}$ et $\Sigma = \{a, b, c\}$. À titre explicatif, l'horloge y est remise à 0 suite à une transition de l_0 vers l_1 et la transition de l_1 vers l_2 ne peut être effectuée que quand y vaut une unité de temps.

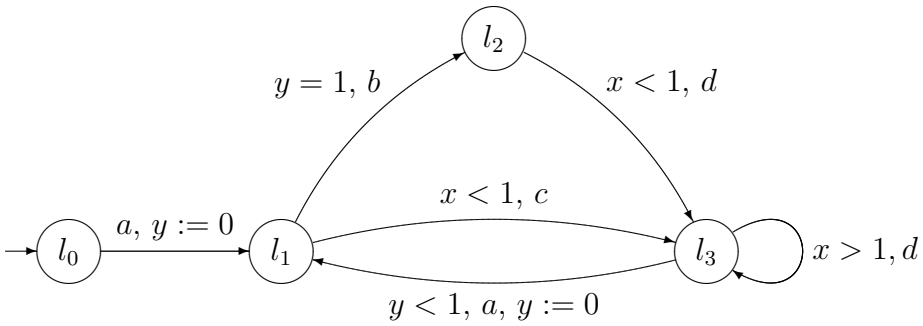


FIGURE 2.3 – Exemple d'un automate temporisé.

De nos jours, les DTMC, MDP, CTMC et TA peuvent être modélisés et vérifiés efficacement par des vérificateurs implémentant des algorithmes d'évaluation de modèles. Nous pouvons citer PRISM [10], SPIN [25], UPPAL [26]. Comme mentionné plus haut,

nous focalisons dans nos recherches sur les processus de Markov étiquetés, une extension des chaînes de Markov qui généralisent les MDP aux espaces d'états infini. Seul le vérificateur CISMO, que nous allons améliorer d'ailleurs, les vérifie actuellement. À la prochaine section, nous abordons les systèmes probabilistes infinis et précisément la classe de processus de Markov étiquetés représentables pour CISMO.

2.2 Systèmes probabilistes infinis

Dans cette section, nous introduisons de façon générale les systèmes probabilistes infinis avant de définir la classe des processus de Markov étiquetés au coeur de nos travaux. Contrairement aux systèmes finis largement étudiés dans la littérature, l'engouement des chercheurs pour les systèmes infinis est récent. Parfois et dépendamment des objectifs de la vérification d'un système, on n'a pas d'autres choix que de le considérer comme un système infini. Élucidons cet argument à travers l'exemple d'une distributrice de café.

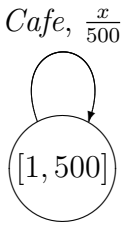
Une première vision de la distributrice pourrait amener à dénombrer les états suivants :

- *Prêt* : l'état initial.
- *Choix* : pour initier une transition à partir de l'état initial, l'utilisateur insère de l'argent. Ce faisant, la distributrice passe à l'état *Choix* où l'utilisateur peut demander du café ou un remboursement.
- *Servir* : Si l'utilisateur demande du café, la machine le sert et revient à l'état initial. Sinon, elle ressort l'argent encaissé et revient à l'état initial. Dans les deux cas, la distributrice transite par l'état *Servir* avant de se réinitialiser.

Cette première vision de la distributrice du café fait d'elle un système fini.

Imaginons maintenant la distributrice avec un seul bouton étiqueté *Cafe* et supposons que nous modélisons les quantités de café versables sachant que la distributrice peut servir de 1 à 500 millilitres de café. Si un état est un nombre réel représentant une quantité de café versable possible, alors la distributrice de café est un système infini. De plus, si nous supposons que la distribution de probabilité sur les états est continue, alors la probabilité d'avoir une quantité de café précise à chaque fois qu'on actionne *Cafe* serait nulle. Par contre, celle d'obtenir une quantité appartenant à un intervalle particulier serait toujours non nulle. La figure 2.4 illustre le modèle de la distributrice de café selon la deuxième vison. Dans le modèle, x est une variable désignant une quantité versable et à valeur d'état ultime lors d'une transition. Ainsi, à titre illustratif, la

probabilité que la distributrice effectue une transition à partir d'un état quelconque pour servir 300 millilitres de café vaut $\frac{300}{500} = \frac{3}{5}$. Remarquons qu'en remplaçant l'état *Servir* de la vision simpliste par la description de la seconde vison (système infini) nous obtenons un modèle complet à espace d'états non dénombrables qui répond aux préoccupations des deux visions.



La variable $x \in [1, 500]$ est utilisée pour représenter l'état ultime d'une transition.

FIGURE 2.4 – Exemple de système infini : une distributrice de café.

Maintenant que le lecteur a une meilleure idée de ce qu'est un système infini, abordons la description des processus de Markov étiquetés.

Un processus de Markov étiqueté sert à modéliser un système réactif et probabiliste. Comme son nom l'indique, c'est une chaîne de Markov et précisément à temps discret. À la différence des modèles déjà présentés, ils généralisent les MDP à un espace d'état continu et supportent des distributions de probabilités continues comme discrètes. Les LMP ont fait l'objet de plusieurs travaux [6, 7, 27, 28, 29] et celui de Richard [6] a permis la mise au point de CISMO [30]. Des exemples de systèmes ont été modélisés avec le formalisme des LMP et vérifiés par CISMO. Parmi ces systèmes, nous distinguons le thermostat [6] et la distributrice de café.

L'analyse d'un système représentable par un processus de Markov étiqueté n'est pas souvent exemptée de failles. Il arrive que des systèmes infinis en exécution présentent des comportements qui échappent aux analyses des concepteurs. Le fait que la distributrice de café ne réponde (verser du café) pas à un appui du bouton *Cafe* illustre ce cas de figure. Même si nous ignorons les facteurs exacts pour lesquels la machine ne répond pas, nous devons abstraire ce comportement imprévisible par des probabilités lors de la modélisation puisque nous savons que c'est un cas de figure qui peut poindre. C'est donc l'absence de réactions suite à un stimulus qui explique le fait que dans un système

réactif, du moins dans celui que nous étudions, la somme des probabilités de transition d'un état vers tous les autres états n'est pas supposée donner forcément 1. Ceci justifie la notion de transition probabiliste partielle défini au 2.1.4.

Rappelons que dans notre projet de recherche, nous nous restreignons aux processus de Markov étiquetés ayant une distribution de probabilité uniforme sur les états et représentables pour CISMO. Pour faciliter donc la compréhension des exemples de modèles de processus de Markov étiquetés à la suite de leur définition formelle, nous rappelons au lecteur d'abord la loi uniforme et précisons la notation adoptée en la matière.

La loi uniforme est la loi de probabilité la plus simple. Dans notre contexte, elle modélise l'expérience aléatoire consistant à choisir un réel au hasard dans un intervalle. Il existe deux versions de loi uniforme : l'une discrète et l'autre continue.

Abordons en premier la loi de probabilité uniforme continue. Soit μ une mesure de probabilité et X une variable aléatoire suivant une loi uniforme sur $[a, b] \subset \mathbb{R}$. Nous avons :

$$\forall [a', b'] \subset [a, b],\ \mu(X \in [a', b']) = \tfrac{b'-a'}{b-a}$$

Notons que, que les bornes inférieure et supérieure d'un intervalle soient ouvertes ou fermées, la façon de calculer la probabilité ne change pas. Aussi, la probabilité de choisir un réel précis dans $[a, b]$ est nulle.

Dans les travaux antérieurs sur CISMO, les probabilités de transition sont définies à l'aide de fonctions de répartition. Nous ferons de même dans ce mémoire sans élaborer la façon dont une probabilité de transition est calculée dans les travaux antérieurs. Pour en prendre connaissance, le lecteur peut consulter [7, 6]. Pour faire le lien avec une fonction de densité, une fonction de répartition F d'une variable aléatoire X de densité f se définit comme ci-dessous.

$$F(x) = \mu(X \leq z) = \textstyle\int_{-\infty}^{z} f(x)dx$$

En nous rapportant à une transition dans un processus de Markov étiqueté, $F(x)$ permet de calculer la probabilité que celui-ci effectue une transition vers un état plus petit que z à partir d'un état donné. La fonction de densité d'une loi uniforme sur $[a, b] \subseteq \mathbb{R}$

étant :

$$f(x) = \begin{cases} \frac{1}{b-a} & si \qquad a < x < b \\ 0 & sinon \end{cases}$$

alors sa fonction de répartition est :

$$F(x) = \begin{cases} \frac{x-a}{b-a} & si \quad a \leq x \leq b \\ 0 & si \quad x < a \\ 1 & si \quad x > b \end{cases}$$

Venons en maintenant à la loi uniforme discrète. Une variable aléatoire qui peut prendre $n \in \mathbb{N}$ valeurs possibles x_1, x_2, ... , x_n équiprobables, suit une loi uniforme discrète lorsque la probabilité de n'importe quelle valeur x_i $(0 < i < n)$ est égale à $\frac{1}{n}$.

Dans les lignes qui suivent, nous utilisons la notation $U(a, b)$ dans une fonction de répartition d'un processus de Markov étiqueté pour matérialiser une loi uniforme continue sur $[a, b] \subseteq \mathbb{R}$. En présence d'une loi uniforme discrète sur un ensemble de réel E, nous utilisons directement la probabilité $\frac{1}{n}$ avec $n \in \mathbb{N}$ la cardinalité de E. Pour éclaircir les idées sur la loi uniforme continue, nous l'appliquons dans l'exemple simple ci-dessous.

Exemple 2.2.1. Considérons une variable aléatoire X suivant la loi uniforme continue sur $[0, 2]$. La probabilité de choisir un réel dans $[0, 2]$ et qu'il appartienne à $[1, \frac{3}{2}]$ vaut : $\mu(X \in [1, \frac{3}{2}]) = \frac{\frac{3}{2}-1}{2-0} = \frac{1}{4}$

La même probabilité peut être obtenue avec la fonction de répartition. La fonction de densité étant $f = \frac{1}{2-0} = \frac{1}{2}$, nous avons : $\mu(X \leq \frac{3}{2}) = \int_{-\infty}^{\frac{3}{2}} f(x)dx = F(\frac{3}{2}) - F(1) = \frac{\frac{3}{2}-1}{2-0} = \frac{1}{4}$

La loi de probabilité uniforme étant décrite, nous allons poursuivre avec la description des processus de Markov étiquetés. La prochaine définition est leur description formelle et originale. Dans celle-ci, conformément à la définition 2.1.4, $\mu_a(x, E)$ désigne la probabilité de se rendre dans un état $s' \in E$ à partir de $x \in S$ en exécutant $a \in Act$.

Définition 2.2.1. [6] Un processus de Markov étiqueté (LMP, de l'anglais *Labelled Markov Process*) est un tuple de la forme $(S, \Sigma, i, AP, Act, Label, \{\mu_a \mid a \in Act\})$ où :

- $S \in \Sigma$ est l'ensemble des états ;
- Σ est une σ-algèbre borélienne sur S ;
- $i \in S$ est l'état initial ;

- AP est un ensemble dénombrable d'étiquettes sur les états ;
- Act est l'ensemble des actions ;
- $Label : AP \to \Sigma$ est la fonction mesurable retournant l'ensemble des états satisfaisant une étiquette donnée ;
- Pour toute action $a \in Act$, $\mu_a : S \times \Sigma \to [0,1]$ est une fonction de transition probabiliste partielle (Définition 2.1.4).

Pour tout $x \in [c,d] \subseteq S$, $[e,b] \subseteq S$ et f une fonction définie sur $[c,d]$, nous écrivons $\mu_a(x,[e,b]) \sim f(x)U(e,b)$ pour signifier que $\mu_a(x,.)$ suit la loi uniforme $f(x) \times U(e,b)$.

Nous remarquons que dans la définition 2.2.1 la fonction *Label* est définie sur les propositions atomiques et retourne un ensemble d'états. Pour les DTMC et les MDP, la fonction était dans l'autre direction (états vers propositions atomiques). C'est plus simple de définir cette fonction ainsi quand on manipule des ensembles d'états non dénombrables. Les 2 notations sont facilement interchangeables pour les DTMC.

La définition 2.2.1 étant générale, nous utiliserons une autre version restreinte aux distributions uniformes et aux ensembles d'états que l'on peut énumérer. Voyons un exemple.

Exemple 2.2.2. La figure 2.5 illustre informellement le modèle d'un LMP. Voici sa description formelle. $S = [0,3] \cup \{4,5\}$, $Act = \{a,b\}$, supposons 1 son état initial et voici la liste des transitions :

- Si $x \in [0,1]$:

$$\mu_a(x,\{0\}) = \tfrac{x}{4}\,; \qquad \mu_a(x,\{1\}) = \tfrac{1-x}{4}\,;$$
$$\mu_a(x,]1,2]) \sim \tfrac{1}{4}U(1,2)\,; \qquad \mu_a(x,]2,3]) \sim \tfrac{x}{4}U(2,3)\,;$$
$$\mu_a(x,]0,1]) \sim \tfrac{x}{4}U(0,1)$$

- Si $x \in]1,2]$:

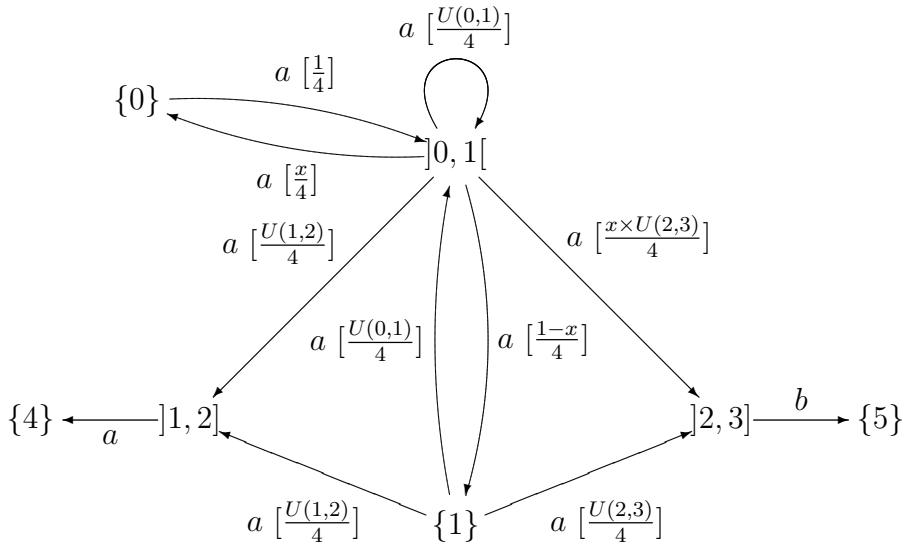
$$\mu_a(x,\{4\}) = 1$$

- Si $x \in]2,3]$:

$$\mu_b(x,\{5\}) = 1$$

- Pour toutes les autres valeurs de x, $\mu_a(x,.)$ et $\mu_b(x,.)$ valent 0.

En observant le modèle du LMP qu'illustre la figure 2.5, nous déduisons que la manière traditionnelle d'énumérer les transitions une à une, comme dans les modèles probabilistes finis, n'est pas suffisante pour décrire le LMP puisque l'ensemble de ses états est



La variable x est utilisée pour représenter l'état de départ d'une transition.

FIGURE 2.5 – Exemple d'un LMP avec des distributions de probabilités uniformes.

non-dénombrable. Même si le modèle est informel, il renseigne que les transitions sont définies sur une partition finie de l'ensemble des états. Cette information est absente de la définition 2.2.1 et pourtant, c'est la structure de système fini du modèle de LMP qu'illustre la figure 2.5 que CISMO accepte. Nous redéfinissons formellement au 2.2.2 un LMP en LMP* pour refléter cette structure de système fini dont nous nous servons fortement dans notre contribution plus loin.

Définition 2.2.2. Un LMP* est un tuple de la forme $(S, I, i, AP, Act, Label, \{\mu_a \mid a \in Act\})$ où $(S, \Sigma, i, AP, Act, Label, \{\mu_a \mid a \in Act\})$ est un LMP et :

1. S est une union finie d'intervalles de $\mathbb{R}$;
2. I est une partition finie de S en intervalles et $\{i\} \in I$;
3. Les μ_a sont des fonctions de transition probabilistes uniformes définies sur les éléments de I, c'est-à-dire pour un certain $f : S \to [0,1]$, $\mu_a(s, I_0) \sim f(x)U(I_0)$ avec $I_0 \in I$ et $s \in S$;
4. $\forall s \in S, \mid \{a \in Act : \mu_a(s, S) > 0\} \mid = 1$.

Le LMP qu'illustre la figure 2.5 est un LMP* et la partition de S considérée est $I = \{]0,1[,]1,2], \ ]2,3], [1,1], [4,4], [5,5], [0,0]\}$. Dans les lignes qui suivent, nous entendons implicitement par LMP un LMP*.

2.3 Présentation sommaire du vérificateur CISMO

Rappelons qu'un vérificateur est un outil informatique qui permet de faire de la vérification automatique de propriétés sur des modèles. Nous présentons dans cette section, à travers une synthèse, le vérificateur CISMO dédié aux LMP.

Dans l'introduction de ce mémoire, nous avons décrit ce qu'est un modèle. Dans cette section, définissons ce qu'est une logique, présentons celle implantée dans CISMO et passons en revue les structures de données qui sous-tendent CISMO.

2.3.1 Logique originale pour exprimer des propriétés sur les LMP

En évaluation de modèle, rappelons qu'une logique sert à exprimer des propriétés et un vérificateur reçoit une propriété et un modèle et retourne si oui ou non la propriété est satisfaite. Une logique est définie par une syntaxe et une sémantique. La syntaxe est un système de symboles et de règles combinables et la sémantique y associe une interprétation. La prochaine définition présente la logique originale pour exprimer des propriétés sur les LMP. Nous l'appelons $\mathtt{L}_0$ et c'est celle que reconnaît CISMO. Elle est inspirée de la logique de Larsen et Skou [31] et reste une adaptation de la logique de Hennessy et Milner [32].

Définition 2.3.1. [7] La syntaxe de $\mathtt{L}_0$ est la suivante :

$$\phi, \psi := \mathtt{T} \mid p \mid \neg\phi \mid \phi \wedge \psi \mid \langle a \rangle_q \phi$$

où a est une action, p est une proposition atomique et $q \in [0, 1]$.

Soient $L = (S, I, i, Act, Label, \{\mu_a \mid a \in Act\})$ un LMP et $a \in Act$. En évaluation de modèle, $\models \subseteq S \times \mathtt{L}_0$ est une relation de satisfaction et on écrit pour tout $s \in S$, $s \models \phi$ pour signifier que s satisfait ϕ. Nous utilisons la relation de satisfaction pour définir ci-dessous la sémantique de $\mathtt{L}_0$:

- $s \models \mathtt{T}$, $\forall s \in S$.
- $s \models p$, si et seulement si $s \in Label(p)$.
- $s \models \neg\phi$, si et seulement si $s \not\models \phi$.
- $s \models \phi \wedge \psi$, si et seulement si $s \models \phi$ et $s \models \psi$.
- $s \models \langle a \rangle_q \phi$, si et seulement si $(\exists E \in I \mid (\forall s' \in E \mid s' \models \phi) \wedge \mu_a(s, E) > q))$.

On dit que L satisfait ϕ dans $\mathtt{L}_0$, et on écrit $L \models \phi$, si l'état initial de L satisfait ϕ.

Dans l'exemple ci-dessous, nous vérifions conformément à $\mathtt{L}_0$ la propriété $\langle a \rangle_{0.5} \langle b \rangle_{0.6} \mathtt{T}$ dans le LMP qu'illustre la figure 2.5.

Exemple 2.3.1. Vérifions si le LMP qu'illustre la figure 2.5 satisfait la propriété $\langle a \rangle_{0.5} \langle b \rangle_{0.6} \mathtt{T}$. Étant donné que tous les états satisfont $\mathtt{T}$, alors cherchons les états pour lesquels il est possible de faire une transition avec l'action b et avec une probabilité de plus de 0.6. Ces états sont ceux de l'intervalle $(2, 3]$. À partir de l'état initial, il est possible de se rendre dans l'intervalle $(2, 3]$. Cependant, ces transitions ont une probabilité d'au plus 0.25. Puisque la formule à vérifier indique que les états désirés doivent pouvoir faire une telle transition avec une probabilité supérieure à 0.5, on conclut que le modèle ne satisfait pas la formule $\langle a \rangle_{0.5} \langle b \rangle_{0.6} \mathtt{T}$.

L'évaluation de modèle est une technique en grande partie automatique et pour cette raison, il faut faire attention au temps que prend la vérification d'une propriété dans un modèle par un vérificateur. Un des paramètres qui influencent le temps de vérification est la structure de données utilisée pour encoder le modèle en mémoire. CISMO propose deux structures de données que nous présentons à la prochaine section.

2.3.2 Structure de données implantée dans CISMO

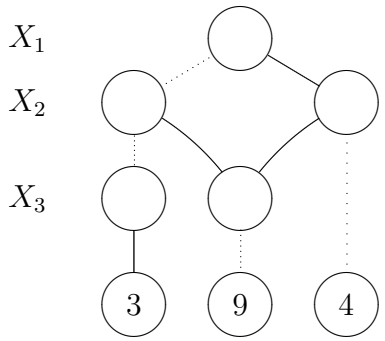
La première version de CISMO est l'œuvre de Richard [6]. Elle est basée sur les tables de hachage, une structure de données offerte en java. Cette version du logiciel est orientée sur la rapidité et plus destinée à la vérification des modèles de petite taille. Autrement exprimé, on vérifie une propriété en un temps acceptable sans prendre en compte la minimisation de l'utilisation de la mémoire.

Plusieurs problèmes sont liés à l'étude de systèmes probabilistes mais l'explosion combinatoire est l'obstacle le plus redoutable. Plusieurs stratégies de vérification visent à repousser cette limite de l'évaluation de modèle. Dans la littérature, outre les stratégies proposées dans [9] et [8] et que nous présentons plus loin, nous notons aussi la technique du *graphe quotient* et l'évaluation de modèle symbolique.

L'approche du *graphe quotient* se base sur une étude comparative de systèmes ou d'états d'un même système et établit des classes d'équivalence (regroupements d'états similaires) en vue de construire un système de transitions réduit dont les états sont les classes d'équivalence. On parle de système de transitions quotient. Pour une meilleure compréhension de la démarche à suivre pour bâtir un système de transitions quotient, le lecteur peut consulter [13].

L'évaluation de modèle symbolique [7, 33, 34, 35, 36] est l'une des percées les plus importantes en matière de vérification par évaluation de modèle. Contrairement à la visite des états individuels dans l'exploration conventionnelle de l'espace d'états d'un modèle, les vérificateurs symboliques visitent un ensemble d'états simultanément. Chaque ensemble d'états peut être représenté par des prédicats tel qu'un état appartient à l'ensemble si le prédicat est satisfait dans celui-ci. Les représentations succinctes des états jumelées aux manipulations efficaces des prédicats des modèles de grande taille font la force la force de cette technique. En évaluation de modèle symbolique on utilise des structures de données basées sur les diagrammes de décision binaires pour encoder les modèles. Parmi les variantes de diagramme de décision binaire, les MTBDD (*Multi-Terminal Binary Decision Diagrams*) [37, 12] sont les plus utilisés. Ils permettent de représenter efficacement des graphes dont les arcs sont étiquetés par un poids et des fonctions à valeurs quelconques. Les MTBDD sont implantés dans plusieurs vérificateurs dont PRISM [10] et CISMO [30, 7] et les études de cas dans [33, 35] sont des exemples de leur application aux systèmes finis comme infinis. Pour étayer les idées sur un MTBDD, nous l'illustrons ci-dessous.

Exemple 2.3.2. La figure 2.6 illustre un MTBDD (figure a) et la fonction qu'il représente (figure b). Les noeuds du graphe sont disposés par niveau et chaque niveau est désigné par une variable : X_1, X_2, et X_3. Les noeuds sont reliés entre eux par des arêtes dont chacune est orientée du haut vers le bas. Les arêtes en pointillé sortant d'un noeud étiqueté par une variable signifient que la variable prend la valeur 0 et les arêtes en trait plein signifient que la variable prend la valeur 1. Les terminaux sont représentés par des cercles étiquetés par une valeur. Pour plus de clarté, nous omettons les terminaux ayant la valeur 0 et les arêtes qui mènent directement à eux. À titre d'exemple, $f(0, 1, 0)$ vaut 9.



(a)

X_1	X_2	X_3	f
0	0	1	3
0	1	0	9
1	1	0	9
1	0	0	4
1	0	1	4
autre			0

(b)

FIGURE 2.6 – Exemple d'un MTBDD et sa fonction.

Notre contexte de travail étant bien fixé, abordons la première et récente approche de vérification qui nous a inspiré dans nos recherches.

2.4 Atteignabilité maximale dans les systèmes probabilistes infinis non réactifs

De nos jours, il existe plusieurs approches [9, 38, 39] pour évaluer dans un système probabiliste des propriétés qui expriment le fait qu'un état souhaité est toujours atteignable ou que des événements satisfaisants se produisent avec une certaine probabilité. Techniquement, on parle de propriétés d'atteignabilité. Dans cette section, nous présentons succinctement le travail publié en 2010 par Kwiatkowska, Norman et Sproston [9] et qui s'applique aux systèmes probabilistes infinis non réactifs et non déterministes ayant une distribution de probabilité discrète sur leurs états. Le fait de se limiter à l'étude de propriétés d'atteignabilité dans un système infini permet d'approximer celui-ci par des systèmes finis à partir desquels on peut inférer leur probabilité. C'est la stratégie adoptée dans l'approche que nous proposons plus loin dans le cadre de notre maîtrise. Elle contourne l'explosion combinatoire et l'approche exposée dans [9] a le mérite de proposer deux étapes pour effectuer une approximation à partir des systèmes traités. Nous nous sommes inspirés de [9] pour établir notre stratégie de vérification des propriétés d'atteignabilité dans les LMP.

Avant d'entrer dans les détails de l'approche exposée dans [9], nous allons introduire des notions indispensables à sa compréhension. Il s'agit des notions d'*ordonnaceur*, de *chemin* et de *probabilité d'ensembles de chemins*. Alors que les deux dernières notions sont requises dans l'étude de propriétés d'atteignabilité dans tout modèle probabiliste,

la première est requise seulement pour les modèles probabilistes non déterministes. La prochaine définition est celle d'un *chemin*. Nous la particularisons avec la fonction de transition d'un DTMC mais elle est toute aussi valide dans tout autre modèle probabiliste.

Définition 2.4.1. Soient $M = (S, I, Act, \mu, AP, L)$ un DTMC et $K \in \mathbb{N}$. Un chemin $\pi = s_0 a_0 s_1 a_1 s_2 a_2 s_3 ... s_K$ avec $s_i \in S$ et $a_i \in Act$ est une suite non vide finie ou infinie composée d'états et d'actions qui alternent. Il commence toujours à l'état initial de M et satisfait : $(\forall i \in K \mid (s_i \in S) \wedge (a_i \in Act) \wedge (\mu_{a_i}(s_i, s_{i+1}) > 0))$

Nous notons $|\pi| = K$ la longueur de π, $\pi(i)$ le $(i+1)^{eme}$ état de π, $\mathrm{Path}(M)$ l'ensemble des chemins dans M et $\mathrm{Path}_M(s)$ l'ensemble des chemins qui débutent par $s \in S$ dans M.

Face à l'étude de propriétés d'atteignabilité dans les systèmes, les recherches découlant de l'évaluation de modèle ont permis de vérifier plusieurs modèles [24, 40, 41, 13] et de développer diverses techniques algorithmiques qui exploitent des notions de disciplines connexes comme celles de la théorie des graphes [42, 43] exploitées dans [9]. L'évaluation de propriétés d'atteignabilité dans un modèle est en fait une instance d'un problème générique : la recherche de chemins reliant deux sommets dans un graphe orienté. Seulement, en vérification probabiliste, en plus de recouvrir un ensemble de chemins reliant des états, on évalue la probabilité de celui-ci. Calculer la probabilité d'une propriété d'atteignabilité dans un modèle revient alors à évaluer la probabilité de l'ensemble des chemins qui respectent celle-ci. Pour ce fait, nous définissons au 2.4.2 la probabilité d'un ensemble fini de chemins dans un modèle probabiliste déterministe ainsi que la probabilité d'un certains ensembles canoniques de chemins infinis.

Définition 2.4.2. Soient $M = (S, I, Act, \mu, AP, L)$ un DTMC et $\pi = s_0 a_0 s_1 a_1 s_2 ... s_K$ un chemin de longueur $K \in \mathbb{N}$ dans M. La probabilité de π, notée $Prob_M(\pi)$, se calcule comme suit : $Prob_M(s_0 a_0 s_1 a_1 ... s_K) = \mu_{a_0}(s_0, s_1) \times ....\mu_{a_{K-1}}(s_{K-1}, s_k)$.

Si $s \in S$ alors la probabilité d'un ensemble de chemins de la forme $\partial(\pi) = \{\rho \in \mathrm{Path}_M(s) : \pi$ est un préfixe de $\rho\}$ vaut : $Prob_M(\partial(\pi)) = Prob_M(\pi) = \mu_{a_0}(s_0, s_1) \times ...\mu_{a_{K-1}}(s_{K-1}, s_K)$.

Notons qu'il est possible d'engendrer une σ-algèbre sur un ensemble quelconque de chemins et pour s'en convaincre, le lecteur peut se référer à [44]. Le théorème d'extension

unique d'une mesure garantit que cette la définition 2.4.2 s'étend de façon unique à tous les ensembles mesurables de chemins et donc sur la σ-algèbre engendrée par $\partial(\pi)$.

Contrairement aux systèmes probabilistes déterministes, quand on a du *non déterminisme*, il n'y a pas une façon canonique de définir une *probabilité d'ensemble de chemins*. L'application de la définition 2.4.2 donnera naïvement une valeur supérieure à 1 comme le témoigne l'exemple ci-dessous.

Exemple 2.4.1. Considérons le MDP qu'illustre la figure 2.2 et calculons (naïvement) la probabilité P de l'ensemble des chemins qui relient les états s_1 et s_3. Soient μ^1 et μ^2 les deux distributions de probabilités associées aux transitions partant de s_1 et μ celle affectée à la transition partant de s_2. On a : $P = \mu_a^1(s_1, s_3) + (\mu_a^2(s_1, s_2) \times \mu_e(s_2, s_3))$ $= 0.5 + (1 \times 1) = 1.5$

Puisque $P > 1$, ce n'est pas la bonne façon de calculer la probabilité.

Pour pallier cette erreur, on dispose d'une stratégie qui choisit chaque fois qu'il y a indéterminisme, une distribution de probabilité pour chaque action et chaque état. Une telle stratégie s'appelle *ordonnanceur* et on peut en avoir plusieurs sur un système *non déterministe*. Ci-dessous sa définition formelle. Il s'agit d'un ordonnanceur sans mémoire.

Définition 2.4.3. Soit $M = (S, I, Act, \mu, AP, L)$ un MDP. Un ordonnanceur sur M est une fonction $O : S \times Act \to D(S)$

Nous dénotons par O_M l'ensemble des ordonnanceurs possibles sur M et si E est un ensemble de chemins obtenu sous $\sigma \in O_M$ alors $Prob_M^\sigma(E)$ désigne sa probabilité.

Étant donné que dans un MDP on doit résoudre les choix non déterministes de transition, la probabilité d'atteindre un ensemble d'états à partir d'un état initial dépend de l'ordonnanceur choisi. L'approche de vérification proposée dans [9] a pour but le calcul de la probabilité maximale qu'on peut obtenir par rapport aux ordonnanceurs possibles dans un MDP. Remarquons que l'approche ne focalise pas sur le calcul de la probabilité. Elle décrit plutôt un algorithme qui transforme un système probabiliste infini en un système probabiliste fini à partir duquel le but escompté est inféré.

En général, un système infini n'est pas équivalent à un système fini. Pour diverses raisons (explosion combinatoire, mémoire d'ordinateur limitée), on peut faire un traitement préliminaire sur un gros système pour réduire son espace d'états afin d'obtenir

un système fini susceptible de satisfaire des objectifs de vérification comportementale. Comment réduit-on alors la taille d'un système ? Les auteurs de [9] utilisent une représentation symbolique du système pour y arriver.

Les systèmes probabilistes symboliques [9] sont des systèmes de transitions disposant d'un ensemble d'états symboliques, des transitions typées (transitions étiquetées avec des variables) et d'opérateurs algébriques (intersection, différence, union, prédécesseur, etc.) sur les états symboliques. Un état symbolique est une variable utilisée pour encoder un ensemble possiblement infini d'états ordinaires. Dans [9], la façon de les générer n'est pas précisée et la nature des états classiques des systèmes traités n'est pas non plus explicitée. Mais à voir les exemples auxquels l'approche est appliquée, nous déduisons que les transitions dans les modèles concernés sont définies sur des ensembles d'états possiblement non disjoints et à l'image des transitions dans les LMP. Ainsi, les auteurs utilisent par exemple l'opérateur de différence pour décider si deux états symboliques sont disjoints ou non.

La prochaine définition tirée de [9], généralise celle dans [45] et définit un système probabiliste symbolique avec une distribution de probabilité discrète sur les transitions. Observons, d'abord, que le système probabiliste symbolique diminué des encodages symboliques est la paire (S, T_d) représentant un système probabiliste infini. Ensuite, en considérant la paire (S,δ), nous comprenons que le système probabiliste symbolique encode les transitons de (S, T_d) sous la forme d'un système de transitions et définit ses distributions de probabilités sur les étiquettes des transitions. Enfin, la fonction $\ulcorner . \urcorner$ permet d'établir une correspondance entre les états symbolique et les états de (S, T_d).

Définition 2.4.4. [9] Un système probabiliste symbolique est un tuple $M = (S, T_d, R, \ulcorner . \urcorner, T_{id}, Dis)$ où :

- S est un ensemble infini d'états ;
- $T_d : S \to 2^{D(S)}$ tient lieu d'une fonction de transition probabiliste ;
- R est un ensemble d'états symboliques qui encodent les états dans S ;
- $\ulcorner . \urcorner : R \to 2^S$ est une fonction d'extension ;
- T_{id} est un ensemble d'identificateurs de transitions tel que pour tout $a \in T_{id}$, on associe une fonction de transition $\delta_a : S \to 2^S$;
- $Dis \subseteq D(T_{id})$ est un ensemble de distributions de probabilités discrètes sur les transitions tel que $\forall s \in S$, en posant $T_{id}(s) = \{a \in T_{id} \mid \delta_a(s) \neq \emptyset\}$ on a $\forall t \in S$:
  1. Si $\mu \in T_d(s)$ alors il existe $v \in Dis$ et $t_a \in \delta_a(s)$ tel que :

$$\sum_{a \in T_{id}(s) \wedge t = t_a} v(a) = \mu(t)$$

2. Si $v \in Dis$ et $t_a \in \delta_a(s)$, alors il existe $\mu \in T_d(s)$ tel que :

$$\mu(t) \geq \sum_{a \in T_{id}(s) \wedge t = t_a} v(a).$$

Le point 1 indique que l'ensemble des distributions de probabilités traduit la façon dont les états sont reliés entre eux. Autrement dit il définit la structure du modèle. Le point 2 sous-tend qu'une probabilité de transition d'une transition dans le système probabiliste symbolique n'est jamais plus grande que la probabilité de transition de la transition qui lui correspond dans le système infini. Pour étayer la définition 2.4.4, nous l'illustrons ci-dessous.

Exemple 2.4.2. [9] Considérons un système dans lequel l'espace d'états est un ensemble de valuations (valeurs réelles) d'une variable x (ou simplement l'espace d'états est $\mathbb{R}$). Supposons que dans un état $s \in \mathbb{R}$, la variable x peut être réinitialisée en une valeur dans l'intervalle $[1, 3]$ ou $[2, 4]$ avec une probabilité de 0.5. Les deux transitions de réinitialisation possible peuvent être étiquetées par a et b, de sorte que : $\delta_a(s) = [1, 3]$ et $\delta_b(s) = [2, 4]$. Si $\upsilon \in D(\{a, b\})$ (ensemble de distributions de probabilité sur les transitions) alors nous avons : $\upsilon(a) = \upsilon(b) = 0.5$ et on peut remarquer que pour tout $s' \in [2, 3]$, il existe une distribution de probabilité $\mu_{s'} \in T_d$ qui met la probabilité de réinitialiser x en s', autrement dit, la probabilité de transiter de s à s' à 1. Aussi, pour toute $\mu_{s'}$ on a $t_a = t_b = s'$.

Comme mentionné plus haut, un système symbolique probabiliste dispose d'opérateurs algébriques sur les états. Ceux-ci permettent de manipuler aisément les états symboliques lors de la génération du système probabiliste fini. Nous définissons alors les principaux opérateurs manipulés dans l'algorithme proposé par les auteurs de [9] en considérant le système probabiliste symbolique $M = (S, T_d, R, \ulcorner . \urcorner, T_{id}, Dis)$:

- Puisqu'à chaque étiquette d'une transition on associe une fonction de transition, on dispose d'une collection de fonctions prédécesseurs $\{pre_a\}_{a \in T_{id}}$ tel que $pre_a : R \to R$ et $\forall \sigma \in R : \ulcorner pre_a(\sigma) \urcorner = \{s \in S \mid (\exists t \in \delta_a(s) \mid t \in \ulcorner \sigma \urcorner)\}$.

  Une fonction prédécesseur permet de retrouver les états à partir desquels on est en mesure d'effectuer une et une seule transition pour atteindre un état ou un ensemble d'états donné.

- L'algorithme calcule l'intersection et la différence d'ensembles d'états non symboliques dans son fonctionnement. Pour cela, pour tout $(\varrho, \sigma) \in R^2$, les auteurs uti-

lisent pour l'intersection la fonction $Et : R \times R \to R$ tel que $\ulcorner Et(\varrho, \sigma) \urcorner = \ulcorner \varrho \urcorner \cap \ulcorner \sigma \urcorner$ et pour la différence la fonction $Diff : R \times R \to R$ tel que $\ulcorner Diff(\varrho, \sigma) \urcorner = \ulcorner \varrho \urcorner \setminus \ulcorner \sigma \urcorner$. Lorsque la fonction d'extension n'est pas appliquée ni à la fonction d'intersection ni à la fonction de différence, on obtient comme résultat un état symbolique au lieu d'un ensemble d'états appartenant à S.

– Les auteurs utilisent également les fonctions $Vide : R \to \{Vrai, faux\}$, tel que $Vide(\sigma)$ retourne $Vrai$ à condition que $\ulcorner \sigma \urcorner = \emptyset$, et $Membre : S \times R \to \{Vrai, faux\}$ d'autre part, tel que $Membre(s, \sigma)$ retourne $Vrai$ si $s \in \ulcorner \sigma \urcorner$.

Maintenant que le lecteur a une meilleure connaissance d'un système probabiliste symbolique, nous formalisons ci-dessous la notion d'atteignabilité et abordons par la suite, les deux étapes de l'algorithme permettant de générer un système probabiliste fini à partir de M et $F \subseteq R$, l'ensemble cible dont on veut évaluer la probabilité d'atteignabilité dans M.

Définition 2.4.5. [9] Soient $M = (S, I, Act, \mu, AP, L)$ un MDP. On dit que $t \in S$ est accessible depuis $s \in S$ s'il existe un chemin de s à t dans M. On dit que $T \subseteq S$ est accessible depuis $U \subseteq S$ s'il existe au moins deux états $t \in T$ et $u \in U$ tel que t est accessible depuis u.

$ProbReach(t, U)$ dénote la probabilité de l'ensemble des chemins débutant par t et passant par un état dans U. Formellement,

$$ProbReach(t, U) = \max_{\sigma \in O_M}(Prob_M^{\sigma}\ (\{\omega \in \mathrm{Path}_M(t) \mid (\exists i \in \mathbb{N} \mid \omega(i) \in U)\})).$$

La première étape consiste à extraire de M un graphe orienté fini noté $graphe(T, E)$ avec $T \subseteq R$ l'ensemble des sommets et $E \subseteq T \times T_{id} \times T$ l'ensemble des arcs. Les sommets du graphe sont conséquemment les états du système probabiliste fini envisagé et les arcs non assortis de probabilités dans E aident à définir ses transitions. Dans [9], c'est l'algorithme $ProbReach$ qui génère $graphe(T, E)$. Nous l'avons modifié légèrement pour faciliter sa compréhension et sa lecture. Pour qu'il termine, il faut nécessairement que T_{id} et Dis (voir la définition 2.4.2) soient finis. Nous présentons à la figure 2.7 l'algorithme $ProbReach$ et dans celui-ci, les ensembles vides résultant de l'intersection d'états symboliques doivent être ignorés.

Dans l'algorithme de la figure 2.7, les auteurs procèdent par itération successive pour extraire un système probabiliste symbolique dans lequel F est atteignable à partir de

$ProbReach(T_{id},\ F)$
{
  $T_0 := F; E := \emptyset$;
  Pour $i := 0, 1, 2,$ ... faire
    $T_{i+1} = T_i$;
    Pour tout $a \in T_{id} \wedge \sigma \in T_i$ faire
      $T_{i+1} := pre_a(\sigma) \cup T_{i+1}$;
      $T_{i+1} := \{Et(pre_a(\sigma), \tau) \mid \tau \in T_{i+1}\} \cup T_{i+1}$;
      $E := \{(pre_a(\sigma), a, \sigma)\} \cup E$;
    Fin pour tout
  Tant que $T_{i+1} \subseteq T_i$
    Pour tout $\sigma \in T_i \wedge (\sigma^{'}, a, \tau) \in E$ faire
      Si $Vide(Diff(\sigma, \sigma^{'}))$ alors
        $E := E \cup \{(\sigma, a, \tau)\}$;
    Fin pour tout
  retourner $graphe(T_i, E)$;
}

FIGURE 2.7 – Algorithme de génération de graphe fini à partir d'un système probabiliste infini.

n'importe quel état. L'algorithme repose sur deux boucles de traitements importants. La première boucle construit progressivement T en utilisant les fonctions prédécesseurs associées aux différents types de transitions. Elle fait alors à partir de $T_0 = F$ un traitement répétitif consistant à introduire chaque fois dans T_i ($i \in \mathbb{N}$) les prédécesseurs des états qui s'y trouvent au moment courant jusqu'à ce qu'un point fixe soit atteint. Une fois le point fixe atteint, on a $T_i = T$. De même, chaque fois qu'un prédécesseur $\sigma \in R$ de τ est ajouté dans T_i, on insère dans E une transition représentée par le triplet constitué de σ, τ et du type de la transition de σ vers τ.

Rappelons que les ensembles d'états dans le modèle symbolique ne sont pas nécessairement disjoints les uns des autres comme le montre l'exemple 2.4.2. L'intervalle $[2, 3]$ dans lequel on souhaite échouer en réinitialisant x se retrouve à la fois dans $[1, 3]$ et $[2, 4]$. Avec les traitements de la première boucle, E n'est pas complet puisque les intersections entre les états symboliques ne sont pas prises en compte dans celle-ci. C'est la deuxième boucle qui traite ce cas. Dans celle-ci alors, pour tout $\sigma \in T_i$ tel que $\sigma \subseteq \tau$ et $(\tau, a, \varrho) \in E$, on complète E avec (σ, a, ϱ). Une fois $graphe(T, E)$ généré, les auteurs exploitent à l'étape 2 les distributions de probabilité dans M pour établir les probabilités de transitions du système probabiliste fini.

Techniquement, l'étape 2 consiste à manipuler T, E et une sous distribution de probabilités $SousDis \subset Dis$ de M pour bâtir le système probabiliste fini. Pour atteindre cet objectif, les auteurs procèdent de la façon suivante :

1. Assembler les transitions dans E ayant un même état de départ. Ensuite, pour chaque groupe de transitions, faire référence à la fois aux distributions de probabilités dans T_d et aux types de transition dans T_{id} pour définir la probabilité de transition de la source commune vers les différents états d'arrivée.
2. M étant avant tout un système probabiliste définissable par la paire (S, T_d), les auteurs définissent de même le système fini comme $Q = (T, T_{d_Q})$ avec $T_{d_Q} : T \to 2^{SousDis(T)}$. Ainsi, pour tout $\sigma \in T$, $\mu \in T_{d_Q}(\sigma)$ si et seulement si il existe $E_\mu \subseteq E$ maximal et $\upsilon \in Dis$ tel que :
   a) Si $(\sigma^{'}, a, \tau) \in E_\mu$, alors $\sigma^{'} = \sigma$;
   b) Si $(\sigma, a, \tau) \in E_\mu$ et $(\sigma, a^{'}, \tau^{'}) \in E_\mu$ sont distincts alors $a \neq a^{'}$;
   c) $\forall \tau \in T$, on a $\sum_{a \in T_{id} \wedge (\sigma, a, \tau) \in E_\mu} \upsilon(a) = \mu(\tau)$.

La condition (a) évoque le fait que les transitions dans E_μ ont une même source alors que la condition (b) indique qu'elles sont distinctes les unes des autres parlant de type de transition. La condition (c) quant à elle, relativise les distributions de probabilités sur E_μ avec les distributions de probabilité $\upsilon \in Dis$.

En conclusion à cette section, nous retenons qu'on peut approximer un système infini par un système fini pour évaluer des propriétés. L'approche exposée est une parmi d'autres visant à pallier l'explosion combinatoire en évaluation de modèle. Alors qu'elle trouve sa force dans l'usage d'une méthode symbolique, la seconde approche qui nous a inspiré réduit également l'espace d'états exploré en vérification probabiliste mais table sur la modulation d'un système. Nous l'abordons à la prochaine section.

2.5 Vérification compositionnelle

Nous abordons la vérification de propriétés de sûreté dans les MDP. Rappelons au lecteur que la compréhension approfondie du contenu de cette section n'est pas requise pour comprendre notre contribution à l'évaluation de modèle plus loin. Notre intention en parlant de vérification compositionnelle, est de présenter la démarche qui la sous-tend et dont nous faisons une source d'inspiration dans nos recherches. L'approche exposée est un travail publié en 2010 par Kwiatkowska, Norman, Parker, et Qu [8]. Elle propose

une méthode de vérification multi-objectifs en ce sens qu'elle permet de vérifier à la fois deux propriétés de sûreté dans un MDP. La particularité des systèmes considérés est qu'ils résultent de l'assemblage de plusieurs systèmes et on les appelle des systèmes composés. L'approche vise à réduire fortement l'espace d'états exploré dans un système composé lors de la vérification de propriétés de sûreté.

En réalité les gros systèmes informatiques ne sont pas séquentiels. Ils sont parallèles par nature et pour les modéliser on exploite leur modularité. On commence à modéliser chaque composant (module) du système par un automate par exemple pour ensuite faire coopérer (parallélisme de modules) les automates pour construire un automate global. On distingue deux types de coopération : synchrone et asynchrone. La coopération asynchrone laisse les modules fonctionner de manière indépendante tandis que celle synchrone établit une communication (envoi et réception de messages) entre les modules. L'approche exposée dan [8] table sur la communication synchrone et les modules de systèmes considérés sont des MDP. La prochaine définition décrit le mécanisme de fonctionnement d'un système composé au moyen de deux MDP.

Définition 2.5.1. [8] Si τ est une action silencieuse, $M_1 = (S_1, i_1, Act_1, \mu_1, AP_1, L_1)$ et $M_2 = (S_2, i_2, Act_2, \mu_2, AP_2, L_2)$ sont deux MDP alors M_1 mis en parallèle à M_2, noté $M_1 \mid\mid M_2$, est le MDP $M = (S_1 \times S_2, (i_1, i_2), Act_1 \cup Act_2, \mu_{M_1||M_2}, AP_1 \cup AP_2, L)$ tel que pour tout $(s_1, s_1^{'}) \in S_1^2$, $(s_2, s_2{'}) \in S_2^2$ et $a \in Act_1 \cup Act_2$ on a $L(s_1, s_2) = L(s_1) \cup L(s_2)$ et $\mu_{M_1||M_2}((s_1, s_2), a, (s_1^{'}, s_2{'})) > 0$ si l'une des conditions suivantes est vérifiée :

- $\mu_1(s_1, a, s_1^{'}) > 0$, $\mu_2(s_2, a, s_2^{'}) > 0$ et $a \in Act_1 \cap Act_2$: la probabilité de transition vaut $\mu_1(s_1, a, s_1^{'}) \times \mu_2(s_2, a, s_2^{'})$;
- $\mu_1(s_1, a, s_1^{'}) > 0$, $s_2 = s_2^{'}$ et $a \in (Act_1 \setminus Act_2) \cup \{\tau\}$: la probabilité de transition vaut $\mu_1(s_1, a, s_1^{'})$;
- $\mu_2(s_2, a, s_2^{'}) > 0$, $s_1 = s_1^{'}$ et $a \in (Act_2 \setminus Act_1) \cup \{\tau\}$: la probabilité de transition vaut $\mu_2(s_2, a, s_2^{'})$.

L'approche proposée dans [8] vérifie à la fois, dans un système composé, deux propriétés de sûreté dont l'une est une hypothèse pour l'autre et pour cette raison, nous la qualifions de vérification par hypothèse. La particularité des propriétés de sûreté ciblées est leur nature probabiliste. Ce sont des propriétés de sûreté assorties chacune d'une probabilité minimum de satisfaction souhaitée dans un MDP. À titre illustratif, «*l'événement W se produit toujours avant l'événement V*» est une propriété de sûreté et «*l'événement W se produit se produit toujours avant l'événement V avec une probabilité d'au moins* 7%» en est une de sûreté probabiliste. Dans [8], la notation $\langle A \rangle_{\geqslant PA}$ désigne

une propriété de sûreté probabiliste. Dans celle-ci, A est une propriété de sûreté et PA est sa probabilité de satisfaction souhaitée. Les propriétés de la forme $\langle A \rangle_{\geqslant PA} M \langle G \rangle_{\geqslant PG}$ où M est un MDP et $\langle A \rangle_{\geqslant PA}$ et $\langle G \rangle_{\geqslant PG}$ des propriétés de sûreté probabilistes tel que $\langle A \rangle_{\geqslant PA}$ est une hypothèse pour $\langle G \rangle_{\geqslant PG}$ sont celles traitées par la vérification par hypothèse. Cette forme de propriété exprime le fait que «*si M en tant que partie d'un système composé satisfait A avec une probabilité d'au moins PA, alors le système au complet satisfera G avec une probabilité d'au moins PG*».

La vérification par hypothèse utilise les formalisations de l'évaluation de modèle classique pour exprimer le fait qu'un modèle satisfait ou non une propriété de sûreté probabiliste. La proposition usuelle «M satisfait la propriété $\langle A \rangle_{\geqslant PA}$», notée $M \models \langle A \rangle_{\geqslant PA}$, est traduisible en une forme de propriété triviale vérifiable par l'approche de vérification par hypothèse. Elle est équivalente à $\langle \mathtt{T} \rangle M \langle A \rangle_{\geqslant PA}$ et veut dire que «*M satisfait A dans tout environnement avec une probabilité d'au moins PA*». Autrement dit il n'y a pas de facteurs qui influencent la probabilité de satisfaction de A dans M. Nous utilisons plus loin cette façon d'énoncer trivialement des propriétés de sûreté probabilistes complexes pour montrer comment la vérification par hypothèse amène à réduire la taille du modèle d'un système composé.

Pour faire de la vérification par hypothèse, au-delà de la notion de parallélisation de MDP, deux autres notions sont nécessaires. Nous aurons besoin non seulement de la notion de produit de MDP mais aussi de traduire en automate le contraire d'une propriété de sûreté. Avant de poursuivre avec les détails complexes de la vérification par hypothèse, nous abordons brièvement ces deux notions à la prochaine section pour faciliter la compréhension de la vérification par hypothèse du lecteur pour la suite.

2.5.1 Traduction d'une propriété de sûreté en automate et produit d'automates

Les propriétés comportementales intéressantes souvent vérifiées sur un système sont celles de sûreté et de vivacité, mais seules celles de sûreté sont considérées dans la vérification par hypothèse. Rappelons au lecteur qu'une propriété de sûreté dans un système stipule que «*quelque chose de mauvais n'arrive jamais*». Par exemple, nous ne pouvons jamais accéder à un compte sans avoir donné le bon mot de passe. Par contre, une propriété de vivacité stipule que «*quelque chose de bon arrivera dans le futur*». Par exemple, quand nous lançons une impression, elle finit par s'achever.

Pour exprimer des propriétés sur un modèle, on a recours à des logiques comme la

logique temporelle linéaire (LTL) [14] dont la sémantique est simple. L'éventail des logiques temporelles proposées dans la littérature est très large. De ce fait, nous n'évoquons ici que celle utile à la vérification par hypothèse et qui apporte un plus dans nos recherches. Les formules en LTL sont construites à partir de variables propositionnelles appelées propriétés atomiques, d'opérateurs booléens ($\vee, \wedge, \neg$) et d'opérateurs de modalité (X ou $\bigcirc$, $\cup$, G ou $\Box$, F ou $\Diamond$) selon la grammaire de la définition ci-dessous dans laquelle p est une proposition atomique.

Définition 2.5.2. $\phi, \varphi := \mathtt{T} \mid p \mid \phi \wedge \varphi \mid \neg\phi \mid \bigcirc\phi \mid \phi U \varphi$

Dans un système, une formule dans LTL s'interprète sur un état. Pour cela, sur un *chemin* dans celui-ci et lors d'une vérification, on s'intéresse aux propriétés que satisfont les états et on reste indifférent aux actions sur les transitions. Si π est un chemin dans un MDP dont la fonction d'étiquetage des états est L, la sémantique de LTL est alors définie inductivement par :

- $\pi \models \mathtt{T}$
- $\pi \models p$, ssi $L(\pi(0)) \models p$
- $\pi \models \phi \wedge \varphi$, ssi $\pi \models \phi$ et $\pi \models \varphi$
- $\pi \models \neg\phi$, ssi $\not\models \phi$
- $\pi \models X\phi$, ssi $L(\pi(1)) \models \phi$
- $\pi \models \Box\phi$, ssi $(\forall j \geqslant 0 \mid \pi(j) \models \phi)$
- $\pi \models \Diamond\phi$, ssi $(\exists j \geqslant 0 \mid \pi(j) \models \phi)$
- $\pi \models \phi U \varphi$, ssi $((\exists i \in \mathbb{N} \mid L(\pi(i)) \models \varphi) \wedge (\forall j \in \mathbb{N} \mid j < i : L(\pi(j)) \models \phi))$

Il est possible d'établir une équivalence entre les propriétés exprimées avec les opérateurs de modalité. En effet, $\Box\phi$ est équivalente à $\neg\Diamond(\neg\phi)$ et $\Diamond\phi$ est équivalente à $(\mathtt{T} U \phi)$. En LTL, on exprime des formules complexes en imbriquant judicieusement les opérateurs et connecteurs que nous avons définis.

En évaluation de modèle, le choix d'une logique dépend des connaissances du spécificateur de propriétés et des propriétés qu'il veut exprimer. Mais lorsqu'il utilise un vérificateur, il doit respecter incontestablement sa logique. Dans le cadre de la vérification par hypothèse, les auteurs utilisent la logique temporelle linéaire, notamment les modalités $\Box$ et $\Diamond$ qu'illustre la figure 2.8. Dans cet exemple p et q sont des propositions atomiques et les cercles représentent des états.

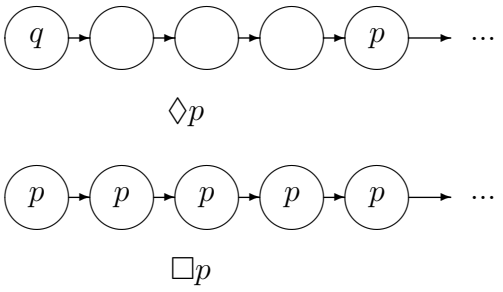


FIGURE 2.8 – Exemple de propriétés dans LTL avec les modalites □ et ◊.

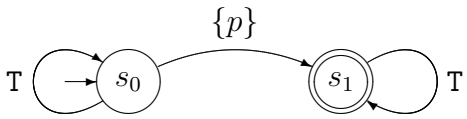


FIGURE 2.9 – Automate acceptant $\Diamond p$.

En LTL, puisqu'il faut interpréter une propriété de sûreté ϕ sur un état, nous pouvons transformer un *chemin* infini π satisfaisant ϕ dans un système en une suite infinie d'actions appelée mot. Nous y arrivons en faisant abstraction des états dans π. Ce faisant, nous pouvons également construire un automate susceptible de reconnaître le mot issu de π. Il suffit de bâtir par exemple un DTMC dans lequel π est un chemin valable. D'ailleurs, une des forces de LTL est qu'elle offre la possibilité de transformer ϕ en un DTMC acceptant les exécutions possibles qui respectent ϕ. Pour ce fait, sur un ensemble fini d'actions Act, nous définissons le langage, noté $\pounds(\phi)$, d'un automate acceptant ϕ de la façon habituelle, comme suit :

$$\pounds(\phi) = \{\pi \in (2^{Act})^w \mid \pi \models \phi\}.$$

La figure 2.9 illustre un automate acceptant $\Diamond p$. Dans celle-ci, s_1 est un *état acceptant*. Autrement exprimé l'automate reconnaît toutes les exécutions passant au moins une fois par s_1. On distingue généralement les *états acceptants* par un double cercle dans un modèle. Observons que dans un automate représentant une propriété, l'étiquetage des états n'est pas nécessaire car les propositions atomiques passent pour des actions et toutes les probabilités de transition valent 1.

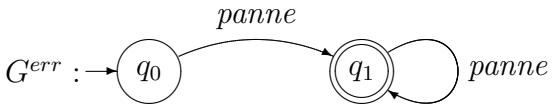
Dans [8], la notation ϕ^{err} désigne l'automate acceptant l'ensemble des comportements qui invalident la propriété de sûreté ϕ. Ainsi, si ϕ est violée dans un MDP, c'est qu'il

existe au moins un contre-exemple, c'est-à-dire une exécution finie acceptée par ϕ^{err}. C'est cette observation qu'exploite la vérification par hypothèse qui fait sa force dans la vérification comportementale de propriétés de sûreté probabilistes dans un système composé. En attendant de prouver cette force, retenons que le langage de ϕ est caractérisable par un ensemble de mauvais préfixes finis, c'est -à-dire un ensemble de mots finis dont les extensions ne se retrouvent pas dans $\pounds(\phi)$. Formellement on écrit :

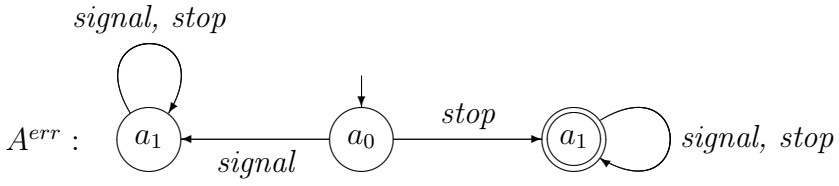
$$\pounds(\phi) = \{\pi \in (2^{Act})^w \mid \text{il n'existe pas de préfixe de } \pi \text{ dans } \pounds(\phi^{err})\}.$$

Pour mieux saisir l'idée d'un automate représentant le contraire d'une propriété, nous exposons dans l'exemple 2.5.1 deux propriétés de sûreté ainsi que les automates acceptant les préfixes qui les invalident. Il s'agit des propriétés «*un signal survient avant un arrêt*» et «*une panne ne survient jamais*». Nous nommons la première A et la deuxième G et les réexploitons dans l'exemple 2.5.3.

Exemple 2.5.1.



G : Une panne ne surgit jamais.



A : Un signal survient avant un stop.

Jusqu'ici, nous avons pris connaissance de la composition de systèmes et de la transformation d'une propriété de sûreté en automate. Avant d'entrer dans le détail de la vérification par hypothèse, nous allons définir la seule et importante notion utile à sa compréhension qui nous reste. Il s'agit du produit de systèmes, notamment celui d'un MDP et d'un automate représentant le contraire d'une propriété de sûreté. La prochaine définition est donc celle du produit de deux systèmes.

Définition 2.5.3. [8] Soient $M = (S_1, i_1, Act_1, \mu_1, AP, L_1)$ un MDP et $A^{err} = (S_2,\ i_2,\ Act_2,\ \mu_2, F)$ avec $Act_2 \in Act_1$ et F un ensemble d'états d'acceptation. Le produit de M par A^{err} est le système noté $M \otimes A^{err} = (S_1 \times S_2, (i_1, i_2), Act_1,\ \mu, AP, L)$, tel que pour tout $(s_1, s_1^{'}) \in S_1^2$, $(s_2, s_2{'}) \in S_2^2$ on a :

- $\mu((s_1, s_2), a, (s_1^{'}, s_2^{'})) > 0$ si $\mu_1(s_1, a, s_1^{'}) > 0$ et $\mu_2(s_2, a, s_2^{'}) > 0$ et $a \in Act_2$. La probabilité $\mu((s_1, s_2), a, (s_1^{'}, s_2^{'}))$ vaut celle de $\mu_1(s_1, a, s_1^{'})$;
- $\mu((s_1, s_2), a, (s_1^{'}, s_2)) > 0$ si $\mu_1(s_1, a, s_1^{'}) > 0$ et $a \notin Act_2$. La probabilité $\mu((s_1, s_2),\ a,\ (s_1^{'}, s_2))$ vaut celle de $\mu_1(s_1, a, s_1^{'})$.
- $L(s_1, s_2) = L_1(s) \cup \{err\}$ si $s_2 \in F$ sinon $L(s_1, s_2) = L_1(s)$.

Abordons maintenant le détail de la vérification par hypothèse.

2.5.2 Évaluation de propriétés de sûreté probabilistes

Rappelons que la vérification par hypothèse possède une double particularité : elle permet de vérifier à la fois deux propriétés de sûreté probabilistes dont l'une est une hypothèse pour l'autre et les propriétés en question sont vues sous forme d'automate. La propriété «*si un signal survient avant un stop dans* 99 *% des cas, alors il est probable que dans* 8 *% des cas qu'une panne ne surviennent jamais*» donne l'allure des propriétés auxquelles s'applique la vérification par hypothèse. Dans celle-ci nous notons les propriétés de sûreté A et G de l'exemple 2.5.1 et la façon dont elle est formulée permet de déduire que $\langle A \rangle_{0.99}$ est une hypothèse pour $\langle G \rangle_{0.08}$. Sa formalisation est $\langle A \rangle_{0.99} M \langle G \rangle_{0.08}$ mais comment l'évaluer dans M ? Nous exposons progressivement la démarche de son évaluation dans les lignes qui suivent. Nous abordons en premier une façon classique et simple d'évaluer une propriété de sûreté probabiliste un MDP et ensuite montrons comment appliquer la même démarche aux propriétés vérifiables par vérification par hypothèse.

Étant donné qu'une propriété de sûreté ϕ est traduisible en un ensemble de chemins dans un automate, alors pour évaluer sa probabilité de satisfaction dans un MDP M, nous devons calculer la probabilité de l'ensemble des chemins qui la respectent. Ainsi pour tout $\sigma \in O_M$ on a :

$$Prob_M^{\sigma}(\phi) = Prob_M^{\sigma}(\{\pi \in \text{Path(M)} \mid \pi \in \pounds(\phi)\}).$$

En assortissant ϕ d'une probabilité de satisfaction p pour en faire une propriété de sûreté probabiliste, on obtient les équivalences suivantes :

$$M \models \langle\phi\rangle_{\geqslant P} \Leftrightarrow (\forall \sigma \in O_M, Prob_M^{\sigma}(\phi) \geqslant P) \Leftrightarrow (Prob_M^{\min}(\phi) \geqslant P).$$

$Prob_M^{\min}(\phi)$ signifie qu'indépendamment de tout ordonnanceur sur M, p est la probabilité minimum avec laquelle on satisfait ϕ. À l'opposé, $Prob_M^{\max}(\phi)$ désigne la probabilité maximale avec laquelle on satisfait ϕ. À titre illustratif, l'exemple 2.5.2 montre comment évaluer $Prob_M^{\sigma}(G)$ (voir l'exemple 2.5.1 pour la propriété G) avec M le système composé issu de la parallélisation des systèmes M_1 et M_2 qu'illustre la figure 2.10.

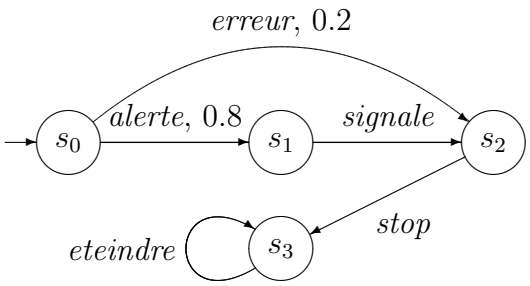
Exemple 2.5.2. [8] La figure 2.10 illustre un DTMC (M_1) et un MDP (M_2). Posons $M = M_1 \parallel M_2$ et évaluons la probabilité de satisfaire dans M la propriété G qu'illustre l'exemple 2.5.1. Il y a une seule exécution finie (préfixe) qui conduit à une panne. C'est celle qui émet une «*erreur*» avec une probabilité de 20% lorsqu'on transite de (s_0, t_0) à (s_2, t_0) et qu'on effectue par la suite une transition de (s_2, t_0) à (s_3, t_3) en exécutant l'action «*stop*» avec une probabilité de 10% pour enfin terminer par une transition de (s_3, t_3) à (s_3, t_3) en exécutant l'action «*panne*» avec une probabilité de 100%. M satisfait donc G avec 98% $(1 - 0,2 \times 0,1)$ comme probabilité. Autrement écrit $M \models \langle G\rangle_{\geqslant 0.98}$ ou $Prob_M^{\min}(G) = 0.98$.

Dans la théorie des probabilités, il est parfois plus aisé de déterminer la probabilité d'un événement à partir de son complémentaire. C'est bien ce qui nous faisons dans l'exemple 2.5.2. Dans ce sens et dans un contexte de vérification probabiliste classique, pour toute propriété de sûreté ϕ dans LTL à vérifier dans M, nous avons :

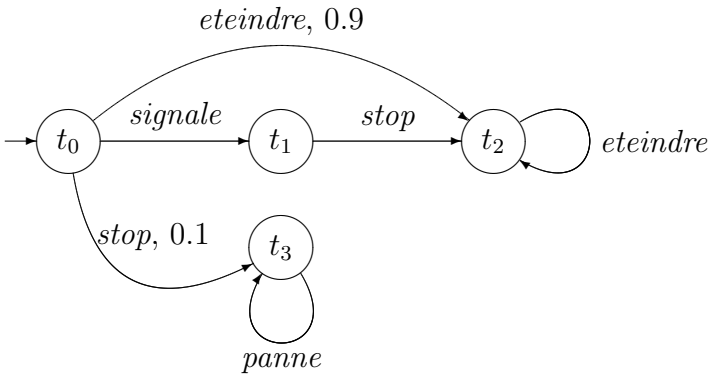
$$Prob_M^{\min}(\phi) = 1 - Prob_{M \otimes \phi^{err}}^{\max}(\Diamond err_{\phi}). \tag{2.1}$$

L'évaluation d'une propriété de sûreté probabiliste par la formule 2.1 dans un système non composé ne démontre pas le gain de performance que peut offrir cette méthode. Elle est a priori moins efficace que les techniques usuelles de l'évaluation de modèle puisqu'elle utilise le produit de systèmes, ce qui engendre un espace d'états plus vaste à explorer : la taille du produit ou de la parallélisation de $n \in \mathbb{N}$ systèmes est au pire le produit des tailles des n systèmes. Mais qu'est-ce qui explique l'efficacité de la formule 2.1 dans la vérification par hypothèse.

La taille d'un automate ϕ^{err} dépend de la propriété ϕ et contient en général moins d'états que l'automate représentant ϕ. Pour vérifier une propriété de la forme $\langle A\rangle_{\geqslant PA} M_1 \parallel M_2 \langle G\rangle_{\geqslant PG}$, on exploite la modularité de $M_1 \parallel M_2$ et on fait intervenir A^{err} dans l'évaluation pour échapper à l'exploration entière de l'espace d'états de $M_1 \parallel M_2$. Avant de détailler l'application de la formule 2.1 dans un système composé, précisons d'abord



M_1 : exemple d'un DTMC.



M_2 : exemple d'un MDP.

FIGURE 2.10 – Exemple d'un MDP et d'un DTMC.

les conditions dans lesquelles une propriété évaluable par vérification par hypothèse est vraie dans un MDP.

Définition 2.5.4. [8] Soient $M = (S, i, Act_M, \mu, AP, L)$ un MDP et $\langle A \rangle_{\geqslant PA}$ et $\langle G \rangle_{\geqslant PG}$ deux propriétés de sûreté probabilistes tel que $A = (S_A, i_A, Act_A, \mu_A, F_A)$ et $G = (S_G, i_G, Act_G, \mu_G, F_G)$. Si $Act_A \subset Act_G \cup Act_M$ alors :

$$\langle A \rangle_{\geqslant PA} M \langle G \rangle_{\geqslant PG} \Leftrightarrow (\forall \sigma \in O_M \mid (Prob^{\sigma}_{M}(A) \geqslant PA \Rightarrow Prob^{\sigma}_{M}(G) \geqslant PG))$$

Pour évaluer $\langle A \rangle_{\geqslant PA} M_1 \parallel M_2 \langle G \rangle_{\geqslant PG}$, les auteurs de l'approche que nous exposons la ramènent d'abord à la forme triviale et évaluent ensuite G dans celle-ci. Pour y arriver, ils procèdent comme suit :

1. Forme triviale : on évalue d'abord l'hypothèse $\langle A \rangle_{\geqslant PA}$ dans M_1 (M_1 est choisi arbitrairement pour expliquer l'approche) avec n'importe quelle technique de l'évaluation de modèle. Si M_1 en tant que composant de $M_1 \parallel M_2$ satisfait $\langle A \rangle_{\geqslant PA}$,

alors on peut être certain que $M_1 \parallel M_2$ va satisfaire aussi l'hypothèse. De ce fait, $\langle A\rangle_{\geqslant PA} M_1 \parallel M_2 \langle G\rangle_{\geqslant PG}$ peut se réécrire $\langle \mathtt{T}\rangle M_1 \parallel M_2 \langle G\rangle_{\geqslant PG}$. Puisque nous savons que M_1 n'influence pas la satisfaction de l'hypothèse dans $M_1 \parallel M_2$ on peut également l'ignorer en tant que compostant et transformer la propriété originale en $\langle A\rangle_{\geqslant PA} M_2 \langle G\rangle_{\geqslant PG}$.

Du moment où nous avons la possibilité de n'utiliser que M_2 pour effectuer la vérification, en tenant compte de la formule 2.1, on peut réécrire $\langle A\rangle_{\geqslant PA} M_2 \langle G\rangle_{\geqslant PG}$ en $\langle \mathtt{T}\rangle M_2 \otimes A^{err} \langle G\rangle_{\geqslant PG}$ avec $Act_A \subseteq Act_{M_1}$ et $Act_G \subseteq Act_A \cup Act_{M_2}$. Les auteurs font le produit de M_2 par A^{err} car lors de l'évaluation de G dans $M_2 \otimes A^{err}$ ils s'intéressent aux chemins passant par des états non étiquetés pas err_A afin de s'assurer que l'hypothèse est toujours vraie. Notons déjà que la taille de $M_2 \otimes A^{err}$ peut être probablement inférieure à $M_1 \parallel M_2$.

2. Évaluation de $\langle G\rangle_{\geqslant PG}$: les auteurs utilisent $\langle \mathtt{T}\rangle M_2 \otimes A^{err} \langle G\rangle_{\geqslant PG}$ à la place de la propriété originale pour terminer la vérification. En posant $M' = M_2 \otimes A^{err}$ et $M = M_2 \otimes A^{err} \otimes G^{err}$, la formule 2.1 mis en application donne :

$$Prob^{\min}_{M'}(G) = 1 - Prob^{\max}_{M}(\Diamond err_G \mid \psi), \psi = (Prob^{\sigma}_{M}(\Box \neg err_A) \geqslant PA). \tag{2.2}$$

En observant l'équation 2.2 et en utilisant le fait que $\Box\neg err \Leftrightarrow \neg\Diamond err$ et $(p \rightarrow q) \Leftrightarrow (\neg p \vee q)$, nous pouvons écrire autrement la définition 2.5.4. $\langle A\rangle_{\geqslant PA} M \langle G\rangle_{\geqslant PG}$ est alors vraie si :

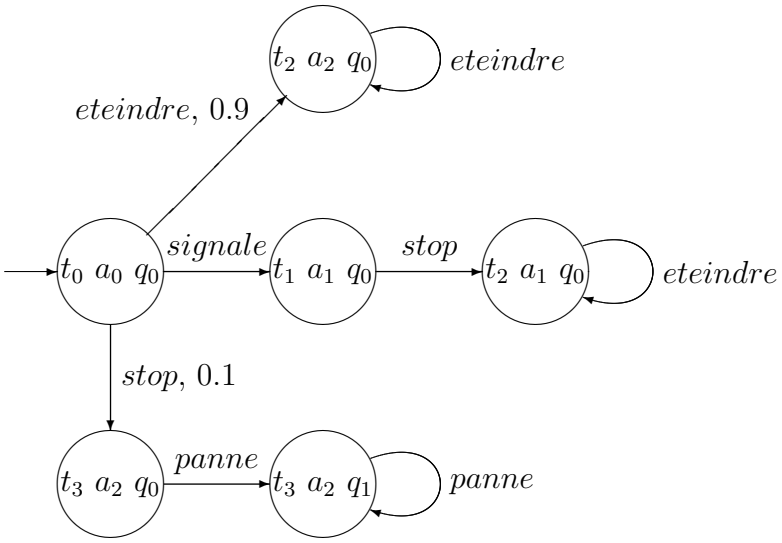
$$\neg(\exists \sigma \in O_M \mid (Prob^{\sigma}_{M}(\Box\neg err_A) \geqslant PA \wedge Prob^{\sigma}_{M}(\Diamond err_G) \geqslant 1 - PG)) \tag{2.3}$$

La formule 2.3 et signifie : $\langle A\rangle_{\geqslant PA} M \langle G\rangle_{\geqslant PG}$ est vraie à condition qu'on ne retrouve pas dans M un *chemin* sur lequel $Prob^{\sigma}_{M}(\Box\neg err_A) \geqslant PA$ et $Prob^{\sigma}_{M}(\Diamond err_G) \geqslant (1 - PG)$ à la fois. En pratique, il est plus judicieux de calculer $Prob^{\min}_{M}(G)$ et la comparer à PG pour décider si $\langle A\rangle_{\geqslant PA} M \langle G\rangle_{\geqslant PG}$ est vraie ou pas sachant qu'on souhaite avoir $Prob^{\min}_{M}(G) \geqslant PG$.

Pour mieux comprendre les deux étapes de la vérification par hypothèse ci-dessus, nous les illustrons à travers l'exemple ci-dessous.

Exemple 2.5.3. [8] Considérons les propriétés A et G illustrées par la figure 2.5.1, les systèmes M_1 et M_2 illustrés par la figure 2.10 et évaluons $\langle A\rangle_{\geqslant 0.8} M_1 \parallel M_2 \langle G\rangle_{\geqslant 0.98}$. Nous savons déjà que M_1 satisfait l'hypothèse $\langle A\rangle_{\geqslant 0.8}$. Pour compléter la vérification, nous évaluons $\langle A\rangle_{\geqslant 0.8} M_2 \langle G\rangle_{\geqslant 0.98}$ par la méthode de vérification par hypothèse en appliquant la formule 2.3 dans $M_2 \otimes A^{err} \otimes G^{err}$ dont le modèle est illustré par la figure 2.11. En observant le modèle, nous notons qu'il n'y a aucune stratégie permettant de choisir

«*signale*» avec une probabilité de 80% et «*stop*» avec une probabilité de 2% (1 − 0.98). Donc la condition $\langle A\rangle_{\geqslant 0.8} M_2 \langle G\rangle_{\geqslant 0.98}$ est vraie tout comme le montrent aussi les analyses dans l'exemple 2.5.2 ($M_1 \parallel M_2 \models \langle G\rangle_{\geqslant 0.98}$). Par contre, si l'automate A^{err} exécutait «*signale*» dans 80% des cas et «*stop*» dans 20% des cas, alors la condition $\langle A\rangle_{\geqslant 0.8} M_2 \langle G\rangle_{\geqslant 0.98}$ ne serait pas vérifiée pour tout $PG > 0.98$ $(1 - 0.02)$.



$L((t_2\ a_2\ q_0)) = L((t_2\ a_2\ q_0)) = \{err_A\}$ et $L((t_3\ a_2\ q_1)) = \{err_A, err_G\}$.

FIGURE 2.11 – Exemple de produit d'un MDP et d'automates : $M_2 \otimes A^{err} \otimes G^{err}$ (voir figure 2.5.1 pour A et G et figure 2.10 pour M_2).

L'approche de vérification multi-objectifs ne se limite pas à l'évaluation de propriétés de sûreté probabilistes dans un système disposant que de deux modules de MDP. Elle est généralisable et peut vérifier également des propriétés formulées avec plus d'une hypothèse. La démarche de vérification est la même que celle présentée et nous l'abordons brièvement sans exemple et telle que décrite par les auteurs.

Imaginons que nous disposons d'un MDP $M = M_1 \parallel ... \parallel M_{k+1}$, de la propriété $\langle G\rangle_{\geqslant PG}$ et des hypothèses $\langle A_i\rangle_{\geqslant PA_i}$ avec $i \in [1..k]$ (intervalle d'entiers). Pour prouver que $\langle A_1, ..., A_k\rangle_{\geqslant PA_1,...,PA_k} M \langle G\rangle_{\geqslant PG}$ est vraie, nous devons assurer d'abord la véracité des hypothèses en montrant que pour tout $i \in [1..k]$, $M_i \models \langle A_i\rangle_{\geqslant PA_i}$. Ensuite évaluer PG dans $\langle \mathtt{T}\rangle M \langle G\rangle_{\geqslant PG}$, autrement dit dans $M_{k+1} \otimes A_1^{err} \otimes ... \otimes A_k^{err} \otimes G^{err}$.

Il y a moyen d'évaluer aussi les propriétés de la forme $\langle A_1, ..., A_k\rangle_{\geqslant PA_1,...,PA_k} M \langle G_1 \vee ... \vee G_n\rangle_{\geqslant PG_1,...,PG_n} (n \in \mathbb{N}^*)$ mais nous n'entrons pas dans les détails de cette évaluation dans ce mémoire.

Pour en savoir davantage sur la vérification par hypothèse et surtout pendre connaissance des différentes démonstrations qui prouvent les idées avancées dans la section, le lecteur peut se référer à [8]. La prochaine section présente nos pistes de réflexion dans le cadre de notre projet de recherche.

2.6 Éléments d'inspiration pour nos recherches

Dans notre état de l'art, nous avons présenté de nouvelles techniques de vérification par évaluation de modèle. L'approche présentée à la section 2.4 part d'un système probabiliste symbolique à espace d'états infini pour extraire un système probabiliste fini utile à l'évaluation de probabilité maximale d'atteignabilité. La vérification par hypothèse, présentée à la section 2.5, quant à elle, met l'accent sur la modulation de systèmes pour effectuer une évaluation multi-objectifs de propriétés de sûreté probabilistes.

Les techniques d'évaluation de propriétés et de représentation de modèles explorées dans notre état de l'art garantissent de bonnes performances dans l'analyse de systèmes probabilistes. Comme le témoignent [8, 9, 37, 45], les théories sur la vérification par hypothèse et les méthodes symboliques sont implantées dans des vérificateurs qui se sont montrés capables d'analyser des spécifications des systèmes réels et les résultats expérimentaux enregistrés montrent clairement leur efficacité dans la complexité des systèmes analysés.

Pour atteindre les objectifs de notre mémoire, nous nous inspirons des différentes idées présentées ci-dessus pour offrir une nouvelle façon d'évaluer les propriétés d'atteignabilité dans les LMP représentables pour CISMO. Que nous inspirent donc les différentes approches de vérification présentées ?

La prise en compte de l'évaluation de propriétés d'atteignabilité dans CISMO nous impose une extension de sa logique (syntaxe et sémantique). Pour ce qui est de la vérification des LMP, nous avons exploré principalement deux approches intéressantes et sommes restés favorables à une. Les notions abordées à la section 2.4 renseignent sur la possibilité de transformer un LMP en système fini afin d'utiliser les techniques algorithmiques dédiées aux systèmes finis. Celles abordées à la section 2.5 renseignent par contre sur la possibilité d'exploiter la modularité des gros systèmes pour évaluer les propriétés d'atteignabilité.

La première approche explorée se réfère à la vérification par hypothèse. La raison qui amène à tenter de l'appliquer aux LMP table sur le fait que les LMP sont générale-

ment de gros systèmes, donc modulables possiblement en plusieurs systèmes que nous pouvons faire coopérer. À titre illustratif, imaginons la distributrice de café constituée d'un LMP gérant différentes températures de café versable (chaude, tiède, froide, etc.), d'un LMP gérant les quantités de café versable et d'un DTMC gérant le passage d'un LMP à un autre. La raison évoquée ne garantit pas encore que l'approche présentée dans [8] peut être adaptée et appliquée aux LMP pour verifier des propriétés d'atteignabilité. Il y a d'autres questions à résoudre d'abord : La synchronisation de LMP, le produit de LMP, la vérification de propriétés de sûreté dans un LMP etc. Au regard de ces questions, une solution pour évaluer les propriétés d'atteignabilité dans les LMP par vérification par hypothèse, consiste à refaire pour les LMP tout ce qui a été fait pour les systèmes finis dans d'autres outils comme PRISM [10]. Cette méthode requiert un travail énorme et pourrait à elle seule constituer un sujet de recherche. Pour cette raison, nous n'envisageons pas approfondir cette solution dans ce mémoire.

La seconde approche explorée se réfère à l'approximation d'un LMP par un système probabiliste fini et fait la vérification de propriétés d'atteignabilité sur celui-ci à l'aide d'un vérificateur dédié aux systèmes finis, comme PRISM également. Mais comment procéder à une telle approximation ? faut-il la faire de la manière présentée dans [9] ?

Dans ce mémoire, nous choisissons d'approximer un LMP par un DTMC. Contrairement à l'approche présentée à la section 2.4, dans l'approche que nous proposons, nous utilisons une distribution de probabilité continue sur les états et au lieu d'exploiter un système probabiliste symbolique pour encoder les LMP, nous utiliserons un fondement mathématique pour atteindre notre but. Vu que les états de nos LMP sont continus et possiblement discrets, une adaptation de l'algorithme d'atteignabilité maximale combinée aux outils mathématiques d'intégration numérique nous servira de tremplin pour approximer un LMP par un DTMC afin d'évaluer efficacement des propriétés d'atteignabilité aux moyens de techniques applicables aux systèmes finis.

Sans l'avoir essayé dans ce mémoire, nous pensons que toute personne capable de formuler des propriétés de d'atteignabilité probabilistes comme dans [8] sur un LMP doit pouvoir en faire l'évaluation par vérification par hypothèse et ceci à travers un système fini conséquent construit selon notre approche. Seulement, avec notre approche, on calculera des probabilités moyennes de satisfaction de propriétés d'atteignabilité.

En somme, appliquer les récentes techniques exposées à un LMP implique la transformation de celui-ci en un système probabiliste fini. Nous hissons donc au coeur de nos recherches la transformation d'un LMP en un DTMC et la spécification de pro-

priétés d'atteignabilité dans CISMO. Puisqu'une approche de vérification symbolique exploitant les MTBDD est implantée dans CISMO, nous présumons que l'évaluation de propriétés d'atteignabilité sur de gros systèmes par notre approche devrait conduire à des résultats expérimentaux performants.

Chapitre 3

Atteignabilité moyenne dans les systèmes réactifs

Nous avons abordé à la section 2.4 l'évaluation de propriétés d'atteignabilité dans les systèmes probabilistes infinis non réactifs et avons donné une définition formelle de l'atteignabilité dans la définition 2.4.5. Dans un système informatique, l'atteignabilité de certaines configurations (un état ou un ensemble d'états) est centrale pour garantir la sûreté de son fonctionnement. L'étude de propriétés de sûreté dans l'analyse d'un système réactif est donc très utile pour détecter des failles de fonctionnement surtout que nous savons qu'ils ne terminent pas.

Les solutions algorithmiques pour évaluer les propriétés d'atteignabilité dans les systèmes probabilistes finis se révèlent souvent coûteuses en temps. Ceci est en partie une conséquence de l'espace d'états exploré dans les algorithmes. Même les logiciels les plus simples ont souvent des espaces d'états gigantesques. Que dirait-on alors des systèmes infinis ? L'adaptation des algorithmes conçus uniquement pour les systèmes finis est donc un sujet récurrent de recherche.

Dans ce chapitre, nous voulons vérifier des propriétés sur les LMP (voir définition 2.2.2) et représentables pour CISMO. Malheureusement, CISMO ne gère pas les propriétés d'atteignabilité et aucun autre outil ne peut vérifier les LMP malgré les nombreux travaux dont ils font l'objet. Les différents travaux sur ceux-ci amènent à conclure que leur vérification peut être faite par des techniques d'approximation [46], par la recherche de modèles semblables (simulation ou bissimulation) [47, 48] ou par vérification directe [6]. Rappelons qu'à la section 2.6, pour améliorer CISMO nous avons exploré deux stratégies et sommes restés favorables à une. Nous avons choisi d'approximer un LMP à

vérifier par un DTMC et faire la vérification de propriétés d'atteignabilité sur celui-ci à l'aide d'un vérificateur dédié aux systèmes finis.

Notre approche permet de vérifier une certaine famille de propriétés d'atteignabilité de la forme $\mathbb{P}_q(\phi U \psi)$ avec ϕ et ψ des propriétés dans la logique $\mathtt{L}_0$ (voir définition 2.3.1) et $q \in [0, 1]$ la probabilité avec laquelle on souhaite satisfaire $\phi U \psi$. Dans la plupart des cas, notre approche permet de définir un DTMC qui, pour ce qui est de la formule étudiée, est équivalent au LMP de départ. Quand le DTMC est assorti d'erreurs, il se peut que le résultat de la vérification soit ultimement erroné. Nous démontrons que notre résultat est exact théoriquement ; bien sûr, lors de l'implémentation, il se peut que des imprécisions numériques faussent le résultat, mais ceci fait partie de toute analyse de système avec valeurs numériques. Notre approche est applicable dans un contexte de distribution de probabilités discrètes comme continues et nous nous limitons aux LMP ayant des distributions de probabilités uniformes sur leurs états. Le choix de la loi uniforme résulte de notre souci de présenter l'approche dans un contexte simple.

Nous articulons le contenu de ce chapitre autour de trois points. D'abord, nous élargissons la logique de CISMO pour qu'elle permette d'énoncer des propriétés d'atteignabilité. Pour cela, nous proposons une nouvelle logique $\mathtt{L}_1$ dans la définition 3.1.1 qui intègre à la logique $\mathtt{L}_0$ la syntaxe et la sémantique des propriétés exprimées sous la forme $\mathbb{P}_q(\phi U \psi)$.

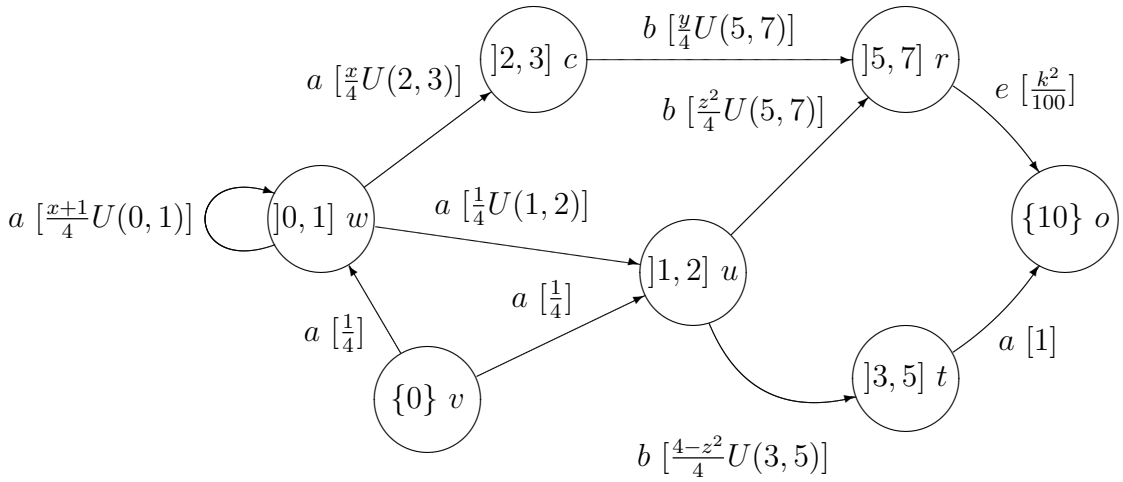
Ensuite, nous présentons théoriquement et en détail notre approche de vérification des propriétés d'atteignabilité dans les LMP. Notre stratégie globale de vérification se résume comme suit : notons que le squelette d'un LMP a une structure de système fini comme l'illustre le LMP de la figure 3.1. Si nous pouvons tout simplement remplacer ses fonctions de transitions par une valeur de probabilité bien choisie, nous obtiendrons un DTMC sur lequel nous pouvons peut-être vérifier notre propriété avec PRISM. Dans les faits, ce n'est pas si simple, mais c'est notre stratégie globale. En réalité, la valeur bien choisie est donnée par le théorème de la moyenne et nous nous basons sur $\phi U \psi$ pour construire le DTMC. Techniquement, dans un LMP nous procédons de la manière suivante :

1. Nous déterminons d'abord les intervalles d'états qui satisfont ϕ ou ψ ;
2. Nous construisons ensuite un DTMC à partir de ces états, les probabilités de transition étant définies par le *théorème de la moyenne* ;
3. Nous passons enfin le DTMC à PRISM pour qu'il dise si oui ou non $\mathbb{P}_q(\phi U \psi)$ est vérifiée.

La figure 3.2 illustre le DTMC résultant des deux premières étapes de notre approche pour la vérification de $\mathtt{T}Uo$ sur le LMP qu'illustre la figure 3.1. Dans celui-ci, par exemple, s_1 représente l'intervalle d'états $[0, 1]$ du LMP et $\frac{3}{8}$ est la moyenne de la fonction de transition $\frac{x+1}{4}$ sur $[0, 1]$. Nous donnons le détail de la construction du DTMC tout au long de ce chapitre. Étant donné que chaque probabilité de transition dans le DTMC correspond à la moyenne d'une fonction de transition dans le LMP, nous évaluons des probabilités moyennes d'atteignabilité. Pour ce fait, nous qualifions notre approche d'atteignabilité moyenne.

Enfin, nous montrons que notre approche a du sens à l'implémentation en précisant les conditions dans lesquelles les imprécisions numériques affectent le résultats ultime d'une vérification de propriété d'atteignabilité dans CISMO. Aussi, par souci de complétion, à l'aide d'outils connus pour la comparaison de modèles, nous établissons une brève étude comparative entre un DTMC obtenu théoriquement selon notre approche et un DTMC assortie d'erreurs c'est-à dire obtenu algorithmiquement selon notre approche.

Abordons la formulation de propriétés d'atteignabilité dans CISMO.



Variables : $x \in [0, 1]$, $y \in]2, 3]$, $k \in]5, 7]$ et $z \in]1, 2]$. Propositions atomiques : w, r, o, t, v et u.

FIGURE 3.1 – Exemple de LMP avec des distributions de probabilité uniforme.

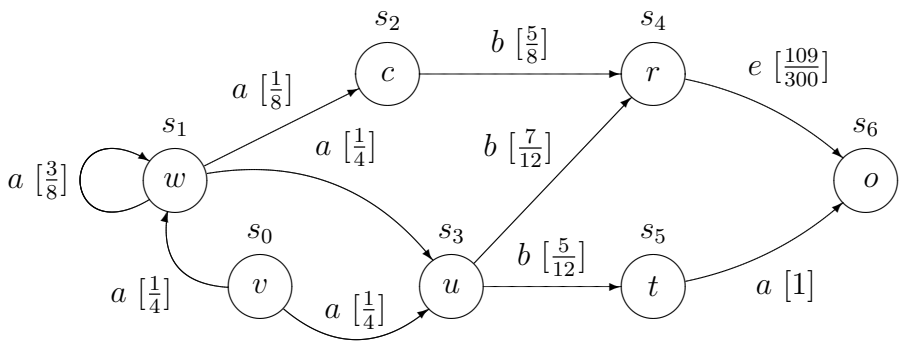


FIGURE 3.2 – DTMC équivalent au LMP illustré par la figure 3.1 pour $\mathtt{T}Uo$.

3.1 Spécification de propriétés d'atteignabilité dans CISMO

Pour exprimer une propriété à vérifier dans le modèle d'un système, on a recours à une logique 2.3.1. CISMO reconnaît à la base la logique $\mathtt{L}_0$ (voir la définition 2.3). Dans cette section, nous rendons $\mathtt{L}_0$ plus expressive en étendant sa syntaxe et sa sémantique à la prise en compte de propriétés de type $\phi U \varphi$ définies à la section 2.3.1. Pour cela, nous définissons une nouvelle logique $\mathtt{L}_1$ qui distingue les formules d'états des formules de chemins. $\mathtt{L}_1$ est presque la logique CTL (Computation Tree Logic) [13, 49] qui contrairement à LTL permet d'exprimer des propriétés sur l'ensemble de chemins qui débutent à un état donné (l'arborescence complète). On y retrouve les modalités de LTL et des opérateurs de quantification ($\forall$, $\exists$ etc.).

Tout au long de la section, nous considérons le LMP $L = (S, I, i, Act, label, \{\mu_a \mid a \in Act\})$ dans nos explications. La prochaine définition précise la logique $\mathtt{L}_1$.

Définition 3.1.1. Les formules d'états de la logique $\mathtt{L}_1$ se formulent à base de la grammaire suivante :

$$\phi, \varphi := \mathtt{T} \mid p \mid \neg\phi \mid \phi \vee \varphi \mid \mathbb{P}_q(\Psi)$$

où $p \in AP$ et $q \in [0, 1]$. Ψ est une formule de chemin telle que :

$$\Psi := \langle a \rangle \phi \mid \phi U \varphi$$

où $a \in Act$.

À part les formalismes $\mathbb{P}_q(\Psi)$, $\langle a\rangle\phi$ et $\phi U\varphi$, les autres éléments de la syntaxe de $\mathtt{L}_1$ ont exactement la même sémantique que celle de $\mathtt{L}_0$. La syntaxe $\langle a\rangle\phi$ est une modification du formalisme $\langle a\rangle_q\phi$ dans $\mathtt{L}_0$. Dans $\mathtt{L}_1$, nous en faisons une formule de *chemin* et $\mathbb{P}_q(\langle a\rangle\phi)$ tient lieu de $\langle a\rangle_q\phi$ dans $\mathtt{L}_0$. Un *chemin* $\pi = s_0a_0s_1...$ satisfait $\langle a\rangle\phi$ si et seulement si $a_0 = a \wedge s_1 \models \phi$.

En ce qui concerne la propriété $\phi U\varphi$, c'est le même formalisme que celui de LTL (voir la définition 2.3.1) que nous considérons ; seulement, notre approche ne vérifie que les propriétés ayant ϕ et φ dans $\mathtt{L}_0$.

Abordons maintenant la syntaxe grammaticale de $\mathbb{P}_q(\Psi)$. $\mathbb{P}$ est un opérateur probabiliste communément utilisé pour exprimer la probabilité de l'ensemble des chemins qui respectent une propriété. Ainsi, pour tout $s \in S$:

$$s \models \mathbb{P}_q(\Psi) \text{ si } Prob_L(\{\pi \in \text{Path}(s) \mid \pi \models \Psi\}) > q.$$

À titre illustratif, nous exprimons ci-dessous quelques propriétés avec $\mathtt{L}_1$.

Exemple 3.1.1. Soit «*panne*» une proposition atomique étiquetant un ensemble d'états dans un LMP et formulons quelques propriétés dans $\mathtt{L}_1$.

- L'action a est possible : $\mathbb{P}_0(\langle a\rangle\mathtt{T})$.
- Il est possible de faire l'action a avec une probabilité de plus de 0.01% deux fois de suite : $\mathbb{P}_{0.01}(\langle a\rangle\mathbb{P}_{0.01}(\langle a\rangle\mathtt{T}))$.
- La probabilité que le système soit hors d'usage est supérieur à 0.2% : $\mathbb{P}_{0.2}(\mathtt{T}\,U panne)$.

$\mathtt{L}_0$ étant extensionné et la famille de propriétés d'atteignabilité qui nous importe étant définie, abordons la définition du fondement mathématique qui fait l'originalité de l'approche d'atteignabilité moyenne. La prochaine section définit théoriquement et algorithmiquement le calcul de la moyenne d'une fonction.

3.2 Moyenne d'une fonction de probabilité

Dans la théorie de la dérivée et de l'intégrale, c'est le *théorème de la moyenne* qui permet d'évaluer la moyenne d'une fonction sur un intervalle. Pour une fonction f continue sur un intervalle $Y \subset \mathbb{R}$, on s'intéresse à la hauteur d'un rectangle, basé sur Y, dont l'aire est égale à l'aire sous la courbe de f. Cette hauteur est la valeur moyenne de f sur Y et

représente le quotient de l'intégrale de f et de la mesure de Y. Ci-dessous la définition formelle du théorème de la moyenne.

Définition 3.2.1. Pour toute fonction f à valeurs réelles, définie et continue sur un segment $[a, b] \subset \mathbb{R}$, il existe un réel c strictement compris entre a et b vérifiant :

$$f(c) = \frac{1}{b-a} \int_a^b f(x)dx.$$

Dans ce mémoire, nous dénotons par $M_f^{[a,b]}$ la moyenne de f sur $[a, b]$.

Nous illustrons ci-dessous le calcul de la moyenne d'une fonction.

Exemple 3.2.1. Soit la fonction $f : [0, 2] \to \mathbb{R}$ définie par $f(x) = 2x$. Puisque f est intégrable sur $[0, 2]$ et $\int_0^2 f(x) = 4$ alors la valeur moyenne de f est $M_{2x}^{[0,2]} = \frac{1}{2} \int_0^2 f(x) = 2$ et $c = 1$.

Comme le montre la définition 3.2.1, le calcul de la moyenne d'une fonction sur intervalle contraint à évaluer une intégrale. Usuellement, une intégrale s'évalue par le biais de formules mathématiques classiques de détermination de primitives. Par exemple, la primitive de la fonction $f = \frac{1}{x}$ $(x \neq 0)$ est $\ln |x|$. Puisque nous implémentons l'approche d'atteignabilité moyenne dans CISMO, des évaluations algorithmiques d'intégrale de fonction de transition dans les LMP s'imposent.

Généralement il est très difficile de construire algorithmiquement une programme qui dérive ou calcule la primitive de fonctions mathématiques. En raison de cette difficulté, nous allons nous rabattre sur l'évaluation numérique d'une intégrale. Vu que les LMP sont des systèmes infinis et que le calcul numérique d'une intégrale avec une infinité de points est impossible, nous allons approximer l'intégrale de toute fonction de transition par une somme discrète de son évaluation sur un nombre fini de points. Observons que l'approximation numérique d'une intégrale n'est pas nécessairement erronée. Elle donne un résultat exact sous certaines conditions. Observons également que si nous nous défaisons de notre objectif d'implémenter l'atteignabilité moyenne et restons dans le cadre théorique, la connaissance de notions d'intégration numérique n'est pas requise pour comprendre notre approche.

Les techniques d'intégration numérique sont nombreuses et très diverses [50, 51, 52] et deux aspects fondamentaux sont observés dans leur l'utilisation : la précision souhaitée et le temps de calcul. Disposer de plus de précision est confortable voire indispensable

lorsqu'on étudie certains problèmes. Mais lorsqu'on est trop exigeant sur cet aspect, le temps de calcul se trouve accru et parfois de manière prohibitive mais reste limité par la capacité des ordinateurs. Bref, quand les calculs sont intrinsèquement rapides et que le temps de calcul n'est pas un facteur déterminant, on aura tout loisir de, et peut-être intérêt à, choisir une méthode plus lente mais plus précise. Mais il n'est pas inutile de garder à l'esprit la notion d'incertitude des méthodes numériques. Dans les lignes qui suivent, nous explorons le fonctionnement de quelques méthodes d'intégration numérique. Étant donné qu'il ne s'agit pas d'un mémoire sur l'analyse numérique, nous ne focalisons pas sur les aspects de temps d'exécution et de précision de calcul d'intégrale. Toutefois, nous situons les sources d'erreurs possibles dans l'implémentation de notre approche dans CISMO.

En matière d'évaluation numérique d'intégrale, il existe deux principales catégories de méthodes :

- Les méthodes déterministes : Il s'agit entre autres des techniques des rectangles, des Trapèzes, de Romberg, de Simpson, de Gauss etc.
- Les méthodes probabilistes : Ce sont les techniques de type Monte-Carlo.

Nous nous appliquons à expliquer différentes techniques d'intégration numérique des deux catégories citées avant d'en choisir une pour nos besoins.

3.2.1 Méthode probabiliste : Technique de Monte-Carlo

L'approximation de l'intégrale d'une fonction f par la méthode de Monte-Carlo sur un intervalle $[a, b] \subset \mathbb{R}$ $(a < b)$ consiste à calculer la somme $I = \frac{1}{N} \sum_{i=1}^{N} f(x_i)$ avec x_i des réels aléatoirement choisis dans $[a, b]$ et $N \in \mathbb{N}^*$. Il faut donc comprendre que plus le nombre de points N augmente, plus l'approximation de l'intégrale de f sur $[a, b]$ est précise et on peut établir l'égalité suivante :

$$\int_a^b f(x)dx = \lim_{N \to +\infty} \frac{1}{N} \sum_{i=1}^{N} f(x_i). \tag{3.1}$$

Soulignons que l'équation 3.1 n'est pas toujours vraie, elle est vraie avec probabilité 1, nous avons donc ici une convergence en probabilité, non pas une convergence analytique.

Poursuivons avec le détail du principe de la théorie de Monte-Carlo. Soient I l'intégrale d'une fonction f définie sur un domaine Ω et p une fonction de densité de probabilité définie sur Ω également tel que :

$$I = \int_\Omega \frac{f(x)}{p(x)} p(x)dx = \int_\Omega g(x)p(x)dx. \tag{3.2}$$

Si X est une variable aléatoire distribuée selon p et $Y = g(X) = \frac{f(x)}{p(x)}$ est aussi une variable aléatoire, alors I est l'espérance de Y et on écrit $\int_\Omega g(x)p(x)dx = E[g(X)]$ (Espérance mathématique en statistique). Autrement exprimé $g(x)$ généralise la moyenne de f sur $[0, 1]$ avec la variable aléatoire X suivant une loi uniforme dans une dimension donnée.

Pour voir les choses de manière un peu plus concrète, approchons ça à l'intervalle $[a, b]$. En posant $I = \int_a^b f(x)dx$, la valeur moyenne de f sur $[a, b]$ est $\frac{I}{b-a}$ d'après le *théorème de la moyenne*. Afin de relativiser I avec un ensemble d'évaluations de f, considérons N points aléatoires x_1, x_2, ..., x_N dans $[a, b]$. Après les calculs de $f(x_1)$, $f(x_2)$, ..., $f(x_N)$ formons la valeur moyenne $\widehat{f_N} = \frac{1}{N}\sum_{i=1}^{N} f(x_i)$ en espérant avoir $\widehat{f_N} = \frac{I}{b-a}$. Cette égalité conduit à la formule générale suivante :

$$\int_a^b f(x)dx \approx \frac{b-a}{N}[f(x_1) + f(x_2) + ... + f(x_N)]. \tag{3.3}$$

Ces résultats sont en fait des conclusions de deux autres résultats importants. Il s'agit de la loi forte des grands nombres et du théorème de limite centrale qui peuvent être consultés en détail dans [50].

En raison de la nature aléatoire de l'échantillonnage des points, la valeur obtenue à la fin d'une simulation Monte-Carlo est la réalisation d'une variable aléatoire. Deux simulations sur un même problème, toutes choses égales par ailleurs, pourraient donc produire des valeurs différentes. Ces valeurs suivent une distribution de probabilité ayant une certaine variance. Alors qu'une méthode déterministe d'intégration numérique produit une approximation de la valeur d'une intégrale, une simulation de Monte-Carlo produit une estimation de cette dernière. Il en est de même pour l'erreur de précision. On ne peut que l'estimer et elle est de l'ordre de l'écart type de l'estimateur de l'intégrale. Sa valeur est de $\frac{1}{\sqrt{N}}\sqrt{E[X^2] - E[X]^2}$.

La méthode de Monte-Carlo est plus qualifiée pour des intégrales de plus d'une dimension qui sont généralement moins efficacement évaluées avec les méthodes déterministes. Plusieurs études comparatives des techniques d'intégration numérique sont disponibles sur le web. Le lecteur peut consulter par exemple par exemple [52].

3.2.2 Méthodes déterministes

Les méthodes numériques déterministes expriment également l'intégrale I de toute fonction f sur un intervalle $[a, b] \subset \mathbb{R}$ $(a < b)$ par une somme discrète de son évaluation avec un nombre fini $N \in \mathbb{N}^*$ de points et on a : $I = \sum_{i=1}^{N} \alpha_i f(x_i)$ avec $\alpha_i \in \mathbb{R}\backslash\{0\}$ et $x_i \in [a, b]$ des variables que nous précisons plus loin. Au-delà du souci d'avoir $\lim_{N\to+\infty} \sum_{i=1}^{N} \alpha_i f(x_i) \approx \int_a^b f(x)$, notons que la qualité d'une méthode d'intégration numérique est d'autant plus élevée dépendamment de la manière dont la convergence vers le résultat exact s'effectue.

Les méthodes d'intégration numérique les plus simples sont celles où les abscisses sont choisies de manière régulièrement espacées. Si l'on dispose de $N+1$ abscisses, on repère celles-ci sur $[a, b]$ par la relation : $x_i = x_0 + ih$ avec $x_0 = a$, $x_N = b$, i un entier naturel allant de 0 à N et $h \in \mathbb{N}$ le pas d'intégration. Pour simplifier les notations par la suite, nous écrirons f_i à la place de $f(x_i)$ pour signifier l'évaluation de f en x_i. Explorons maintenant les techniques d'intégration numérique de nature déterministe citées plus haut.

Technique des rectangles

La première méthode qui vient à l'esprit, c'est de découper la courbe entre f, l'axe des abscisses, la droite d'équation $x = a$ et la droite d'équation $x = b$ en une multitude de petits rectangles de largeur h faible. L'aire sous la courbe est obtenue en cumulant l'aire des petits rectangles du cadre circonscrit. Pour construire ces derniers, en posant $h = \frac{b-a}{N}$ où N est le nombre de rectangles avec lequel on souhaite paver l'aire sous la courbe, on a le choix entre trois techniques :

1. faire coïncider le sommet haut gauche des rectangles avec la courbe et on a :
$$\int_a^b f(x) \approx \sum_{i=1}^{N} h f_i \,;$$
2. faire coïncider le sommet haut droit des rectangles avec la courbe et on a :
$$\int_a^b f(x) \approx \sum_{i=1}^{N} h f(x_i + \tfrac{h}{2}) \,;$$
3. faire coïncider le milieu du coté haut des rectangles avec la courbe et on a :
$$\int_a^b f(x) \approx \sum_{i=1}^{N} h f_i + \tfrac{h}{2}(f_0 + f_N).$$

La technique des rectangles est très simple mais pas très précise et est utilisable si et seulement si f est dérivable sur $[a, b]$. Néanmoins, elle est facile à coder et donne des résultats acceptables pour des fonctions polynomiales, trigonométriques et expo-

nentielles. Lorsque la dérivée première de f est bornée par une constante C non nulle, l'erreur avec la méthode des rectangles est donnée par l'expression $\frac{1}{2}\frac{(b-a)^2}{N}\mid C \mid$.

Technique des trapèzes

Elle consiste à approximer toute fonction f entre deux abscisses successives $x_i \in \mathbb{R}$ et $x_{i+1} \in \mathbb{R}$ par une droite et on a :

$$\int_{x_i}^{x_{i+1}} f(x) = \frac{h}{2}(f_i + f_{i+1}) \text{ avec } h = \mid x_{i+1} - x_i \mid.$$

Pour que cette méthode converge rapidement, il est nécessaire de choisir un pas d'intégration h inférieur à la dérivée seconde de f. Elle donne une intégrale exacte quand f est un polynôme de degré 1 sur $[x_i, x_{i+1}]$. Sur $[a, b] \subset \mathbb{R}$, l'intégrale est donnée par l'équation 3.4 si et seulement si f'' existe :

$$\int_a^b f(x) = h\sum_{i=1}^{N-1} f_i + \frac{h}{2}(f(a) + f(b)) + \Theta(\frac{(b-a)^3}{N^2}f'') \tag{3.4}$$

Technique de Simpson

Dans la méthode des trapèzes, on interpole f par une droite entre les points x_i et x_{i+h}. Avec la méthode de Simpson, f est plutôt interpolée par un polynôme de degré 2 sur un intervalle constitué de 3 abscisses consécutives. Ceci améliore un peu plus la précision. Sur un intervalle, en choisissant un nombre de points pair $N+1$, c'est à dire N impair, on prouve avec le théorème de développement limité de Taylor d'ordre 2 que l'approximation de l'intégrale de f sur $[a, b] \subset \mathbb{R}$ s'évalue avec l'équation 3.5 si et seulement si f^4 existe :

$$\int_a^b f(x) = \frac{h}{3}[f_0 + f_N + 2\sum_{i=1}^{\frac{N-1}{2}}(2f_{2i-1} + f_{2i})] + \Theta(\frac{(b-a)^5}{N^4}f^4). \tag{3.5}$$

La technique de Simpson est de deux ordres de grandeur plus efficace que celle des trapèzes et elle est exacte jusqu'aux polynômes de degré 3. Sachant que cette dernière converge en $\frac{1}{N^2}$, en appliquant à la méthode des trapèzes le procédé d'accélération de Richardson et en faisant des déductions avec la formule d'Euler-Mac-Laurin, on prouve que l'approximation de $\int_a^b f(x)$ par $\frac{4T(\frac{h}{2})-T(h)}{3}$ est meilleure que celle obtenue avec $T(h)$. $T(h)$ respectivement $T(\frac{h}{2})$ désigne l'approximation de $I = \int_a^b f(x)$ par la méthode des trapèzes pour un pas $h = \frac{b-a}{N}$ respectivement $h = \frac{b-a}{2\times N}$. Notons que le

rapport $\frac{4T(\frac{h}{2})-T(h)}{3}$ vaut l'approximation de I par la méthode de Simpson et on dit que celle-ci est une première accélération de la méthode des trapèzes.

Technique de Romberg

L'idée de la technique de Romberg s'inspire directement de la combinaison $\frac{4T(\frac{h}{2})-T(h)}{3}$ faite pour accélérer la convergence de la méthode des trapèzes. Si on calcule successivement et avec la méthode des trapèzes les intégrales avec un nombre de points $\frac{N}{2^k}$, $\frac{N}{2^{k-1}}$,...., N, on obtient rapidement avoir une estimation de l'intégrale avec une erreur de l'ordre de $\frac{1}{2^k}$ où k représente le nombre de fois qu'on évalue l'intégrale. Pour atteindre cet objectif, on construit une relation de récurrence entre différentes accélérations. Soient les notations suivantes :

- $\forall k \geq 0$, $R_{0,k} = T(\frac{b-a}{2^k})$ et se calcule avec la méthode des trapèzes.
- $\forall k \geq 1$, $R_{1,k} = \frac{4R_{0,k}-R_{0,k-1}}{3}$ et représente une première accélération.

Pour 3 accélérations, les nombres obtenus peuvent être présentés dans un tableau comme ci-dessous :

subdivision $2^0 = 1$	subdivision $2^1 = 2$	subdivision $2^2 = 4$	subdivision $2^3 = 8$
$R_{0,0}$	$R_{0,1}$	$R_{0,2}$	$R_{0,3}$
	$R_{1,1} = \frac{4R_{0,1}-R_{0,0}}{3}$	$R_{1,2} = \frac{4R_{0,2}-R_{0,1}}{3}$	$R_{1,3} = \frac{4R_{0,3}-R_{0,2}}{3}$
		$R_{2,2} = \frac{16R_{1,2}-R_{1,1}}{15}$	$R_{2,3} = \frac{4R_{1,3}-R_{1,2}}{15}$
			$R_{3,3} = \frac{64R_{2,3}-R_{2,2}}{63}$

En observant le tableau, on peut déduire la formule générale de la récurrence pour $n \in \mathbb{N}^*$ accélérations et $k \in \mathbb{N}$ subdivisions : $R_{n,k} = \frac{4^k R_{n,k-1}-R_{n-1,k-1}}{4^k-1}$. Lorsque n tend vers l'infini, l'erreur avec la méthode de Romberg est de l'ordre de $\Theta(4^{-n(k+1)})$ avec $k \geq n+1$. Cette technique donne des résultats satisfaisants pour les besoins courants et est programmable. On peut stocker les différents résultats dans une matrice ou simplement écrire une fonction récursive qui prendra en paramètre le nombre d'accélérations et le nombre de subdivisions et fournira la valeur de l'intégrale.

Pour le besoin d'intégration numérique dans CISMO, nous retenons la méthode des trapèzes. La façon d'évaluer théoriquement et algorithmiquement la moyenne d'une fonction étant précisée, abordons le détail des différentes étapes de l'approche d'atteignabilité moyenne dans les LMP.

3.3 Du LMP au DTMC

Nous voulons vérifier des propriétés d'atteignabilité de la forme $\mathbb{P}_q(\phi U \psi)$ dans les LMP. Rappelons au lecteur que nous entendons par LMP un LMP* (voir définition 2.2.2) et que le squelette d'un LMP a une structure de système fini. C'est-à-dire que l'ensemble des états est une union d'intervalles de $\mathbb{R}$ et les transitions sont définies par des fonctions de répartition d'un intervalle à l'autre.

Tout au long de la section, nous considérons la propriété d'atteignabilité $\beta = \mathbb{P}_q(\phi U \psi)$ et le LMP $L = (S, I, i, Act, Label, \{\mu_a \mid a \in Act\})$ dans nos explications théoriques. Par contre, pour étayer les explications théoriques, nous utilisons le LMP qu'illustre la figure 3.1 comme exemple de base et évaluons $\mathbb{P}_{0.5}(\mathtt{T} U o)$ dans celui-ci. Autrement dit nous détaillons tout au long de la section non seulement la construction du DTMC que présente la figure 3.2 mais aussi l'évaluation de $\mathbb{P}_{0.5}(\mathtt{T} U o)$ dans celui-ci.

Pour vérifier β dans L, nous l'approximons par un DTMC sur lequel nous évaluons $\mathbb{P}_q(\mathtt{T} U \psi)$. Le DTMC est construit de telle sorte que nous inférons si L satisfait β, à partir du fait qu'il satisfait β. Notre stratégie de vérification trouve sa force dans l'usage du théorème de la moyenne défini au 3.2.1 et s'apparente à la méthode classique [13] pour évaluer une propriété $\phi U \psi$ dans LTL. Nous l'articulons, comme indiqué plus haut, autour des trois étapes suivantes :

1. Déterminer les intervalles d'états qui satisfont ϕ ou ψ ;

2. Construire un DTMC à partir de ces états, les probabilités de transition étant définies par le *théorème de la moyenne* ;

3. Passer le DTMC résultant à PRISM pour qu'il dise si oui ou non $\phi U \psi$ est vérifiée.

Pour construire un DTMC à partir de L et de β, nous manipulons beaucoup les intervalles de I dans nos algorithmes. Pour cette raison, avant d'aborder les différentes étapes de notre approche, définissons les notions que nous utilisons.

3.3.1 Définitions et exemples

Dans notre approche, nous nous restreignons aux états de L satisfaisant ϕ ou ψ pour construire le DTMC qui approxime L pour β. Il se peut donc que lors de la construction de celui-ci, à la première étape de notre approche, que des intervalles dans le modèle de L, c'est-à-dire appartenant à I, connaissent une réduction. Nous étayons ce cas de figure par l'exemple 3.3.1 pour convaincre le lecteur.

Exemple 3.3.1. Supposons qu'on veuille vérifier $\mathbb{P}_{0.2}((\mathbb{P}_{0.6}(\langle b\rangle \mathtt{T}))\mathtt{U}\mathtt{T})$ dans le LMP qu'illustre la figure 3.1 à la page 53. Si nous voulons les états dans $]2,3]$ qui satisfont $\mathbb{P}_{0.6}(\langle b\rangle \mathtt{T})$, il faut résoudre l'inéquation $\frac{y}{4} > 0.6$ sur $]2,3]$. La solution est $]2.4,3]$ et on note bien $]2.4,3] \subset]2,3]$.

Pour faciliter la lecture et manipuler aisément les intervalles dans le modèle de $L = (S, I, i, Act, Label, \{\mu_a \mid a \in Act\})$, nous avons défini des notations que nous adoptons dans notre approche :

- $I^{\subseteq}(S) = \{J : \exists i \in I \mid J \subseteq i\}$: Cet ensemble d'ensembles est infini. Il contient les intervalles utilisés dans le modèle de L et les intervalles qui sont inclus dans un intervalle de I.
- $I^{\cup}(S) = \{\bigcup_{j=0}^{n\in\mathbb{N}^*} I_j : I_j \in I^{\subseteq}(S)\}$: C'est l'ensemble des unions possibles d'intervalles appartenant à $I^{\subseteq}(S)$. En théorie, $I^{\cup}(S)$ est infini mais à l'implémentation nous ne retenons qu'un nombre fini d'intervalles dans nos analyses. Observons que $I^{\subseteq}(S) \subseteq I^{\cup}(S)$.
- Nous définissons une fonction d'éclatement d'intervalles $Partition : I^{\cup}(S) \to P(I^{\subseteq}(S))$ similaire à la fonction $\lceil . \rceil$ présentée dans la définition 2.4.4. Pour tout $X \in I^{\cup}(S)$, la fonction *Partition* dispose des propriétés suivantes :
  1. $Partition(X)$ est une partition de X et ses éléments appartiennent à $I^{\subseteq}(S)$.
  2. Les éléments de $Partition(X)$ sont maximaux par rapport à I, autrement dit : $(\forall I_1, I_2 \in Partition(X) \mid (I_1 \neq I_2) \Rightarrow \neg(\exists i \in I \mid I_1 \cup I_2 \subseteq i))$
- Au-delà du partionnement d'intervalles, nous avons besoin, en construisant la structure d'un DTMC, de subdiviser par rapport à une partition des ensembles. Pour cela, nous définissons la fonction $Restrict : P(I^{\subseteq}(S)) \times I^{\cup}(S) \to P(I^{\subseteq}(S))$ tel que :
$$\forall Y \in P(I^{\subseteq}(S)),\ X \in I^{\cup}(S),\ Restrict(Y, X) = \{Z \cap X : Z \in Y\}.$$

– Dans nos algorithmes, nous désignons par F l'ensemble des fonctions qui définissent des probabilités de transition dans L et pour tout $f \in F$, nous notons D_f son domaine de définition et I_f son image. Formellement,

$$F = \{f : (\exists a \in Act, I_f, D_f \in I \mid \mu_a(D_f, I_f) \sim f\}.$$

Nous illustrons ci-dessous les notations ci-dessus.

Exemple 3.3.2. Considérons le LMP qu'illustre la figure 3.1 à la page 53. Nous avons $S = [0,7] \cup [10,10]$, $I = \{[0,1]\ ,\ ]1,2],\ ]2,3],\ ]3,5],]5,7],\ [10,10]\}$ et :

– $Partition([0,2]) = \{]0,1[,\ ]1,2]\}$

– $Restrict(\{]0,\frac{1}{2}[,\ ]\frac{3}{2},2]\},[\frac{1}{4},3]) = \{]\frac{1}{4},\frac{1}{2}[,\ ]\frac{3}{2},2]\}$.

– $I^{\subseteq}(S) = \{]0,1[,\ ]1,2[,\ [2,3],\ [0,0],\ ]3,5],\ [0,\frac{1}{2}],\ [\frac{3}{2},2[,\ ...\}$

– $I^{\cup}(S) = \{]0,1[\cup]1,2[,\ ]0,1[\cup[2,3[,\ ]1,2[\cup[0,0],\ ...\}$

Tout comme dans l'approche présentée à la section 2.4, nous calculons des prédécesseurs d'états dans les LMP et pour une raison algorithmique, nous adaptons la fonction prédécesseur de la définition 2.4.4 à nos besoins comme suit : $\wp_{re} : I^{\subseteq}(S) \to P(I^{\subseteq}(S))$ tel que $\forall X \in I^{\subseteq}(S)$,

$$\wp_{re}(X) = \{\gamma \in I^{\subseteq}(S) \mid (\exists a \in Act \mid (\forall x \in \gamma \mid \mu_a(x, X) > 0)) \text{ et } \gamma \text{ est maximal}\}$$

La traduction textuelle de $\wp_{re}$ est la suivante : étant donné un intervalle $X \in I^{\subseteq}(S)$ en paramètre, elle fournit un ensemble d'intervalles à partir desquels il est possible de se rendre dans X en une transition. L'exemple ci-dessous illustre le calcul d'états prédécesseurs dans un LMP.

Exemple 3.3.3. Considérons le LMP qu'illustre la figure 3.1 à la page 53. Nous avons :

– $F = \{\frac{x+1}{4}U(0,1),\ \frac{x}{4}U(2,3),\ \frac{1}{4}U(1,2),\ \frac{y}{4}U(5,7),\ \frac{z^2}{4}U(5,7),\ \frac{4-z^2}{4}U(3,5),\ \frac{p^2}{100}\}$.

– $\wp_{re}([\frac{11}{2},6]) = \{]1,2],\ ]2,3]\}$ car les deux fonctions qui définissent des probabilités d'atteindre $[\frac{11}{2},6]$ sont $\frac{z^2}{4}U(5,7)$ et $\frac{y}{4}U(5,7)$.

Une autre notion indispensable pour comprendre la première étape de notre approche est la notion de *fonction à branches* que nous avons tirée des travaux de Richard [6]. Pour déterminer les états de $L = (S, I, i, Act, Label, \{\mu_a \mid a \in Act\})$ qui satisfont ϕ ou ψ, nous faisons des calculs de probabilités intermédiaires, notamment celui de calculer $\mu_a(x, E)$ pour tout $x \in S$, $a \in Act$ et $E \subseteq S$. Les transitions dans L étant définies sur

I, nous n'avons pas d'autre choix que de considérer les intervalles dans I pour calculer $\mu_a(x, E)$ et garder le résultat, pour une raison algorithmique, sous forme d'ensemble de paires *intervalle-fonction* appelées *fonction à branches*. Une paire *intervalle-fonction* est une branche dont le domaine est l'intervalle et la fonction [1] la valeur sur celui-ci.

Une *fonction à branches* retourne pour chaque intervalle une fonction de cet intervalle dans [0,1]. Pour l'évaluer en $x \in S$, nous recherchons si un intervalle contient x parmi les paires *intervalle-fonction*. Si oui, nous évaluons la fonction associée en x. Dans le cas contraire, la *fonction à branche* vaut 0 en x. L'exemple ci-dessous éclaircit les idées sur la *fonction à branches*.

Exemple 3.3.4. Considérons le LMP qu'illustre la figure 2.5 à la page 26 et déterminons la *fonction à branche* f pour $x \in]0, 1]$. $S = [0, 3] \cup [4, 4] \cup [5, 5]$ et nous avons :

- Si $x \in]0, 1[$, $\mu_a(x, S) = \frac{x+1+1+x+(1-x)}{4} = \frac{x+3}{4}$
- Si $x \in [1, 1]$, $\mu_a(x, S) = \frac{1+1+1}{4} = \frac{3}{4}$

on a donc $f(x) = \{(]0, 1[, \frac{x+3}{4}), ([1, 1], \frac{3}{4})\}$.

Les différentes notations pour manipuler les intervalles dans le modèle d'un LMP étant fixées, abordons la première étape de l'approche d'atteignabilité moyenne.

3.3.2 Déterminer l'ensemble d'états satisfaisants une propriété dans la logique de base de CISMO

Nous détaillons la première étape de l'approche d'atteignabilité moyenne en considérant toujours la propriété d'atteignabilité $\beta = \mathbb{P}_q(\phi U \psi)$ et le LMP $L = (S, I, i, Act, Label, \{\mu_a \mid a \in Act\})$ dans nos explications théoriques.

La première étape de notre approche consiste à déterminer séparément les états qui satisfont ϕ et ceux qui satisfont ψ. Cette étape a sa raison d'être puisqu'une fois effectuée, dans la deuxième étape de notre approche, nous évaluons différentes valeurs moyennes de fonctions de transition dans L uniquement sur ces états. Ce faisant, nous approximons mieux la probabilité réelle de β dans L, car une réduction d'intervalle dans le modèle du LMP influence positivement la probabilité avec laquelle le DTMC envisagé pourrait satisfaire β. Autrement dit la moyenne d'une fonction de transition f sur $[a, b] \subset [c, d] \subset \mathbb{R}$ est plus intéressante que la moyenne de f sur $[c, d]$ si c'est seulement à partir de l'intervalle $[a, b]$ que les états cibles peuvent être atteints. Étant donné que

1. La fonction n'est pas nécessairement une constante.

les propriétés ϕ et ψ sont dans la logique $\mathtt{L}_0$ définie au 2.3.1, pour déterminer l'ensemble d'états satisfaisant l'une ou l'autre des propriétés, nous utilisons des algorithmes tirés des travaux de Richard [6]. Ses travaux exposent trois algorithmes principaux qui se complètent pour atteindre le but de la première étape de notre approche.

Le premier, *Sat*(L, ϕ) tiré de [6], effectue une vérification directe de ϕ dans L. Il explore donc judicieusement l'espace d'états basé sur la sémantique des opérateurs logiques formant ϕ et retourne les états de L qui satisfont ϕ sous forme d'un intervalle. C'est l'algorithme moteur qui assume la première étape de l'approche d'atteignabilité moyenne dans LMP. Le deuxième, *CalculerProbabilite*(L, E, a) tiré de [6], retourne une fonction à branche (voir la section 3.3.1) correspondant à $\mu_a(x, E)$ pour tout $x \in S$, $a \in Act$ et $E \subseteq S$. Le troisième, *Separation* ($f(x)$, q) tiré de [6], détermine pour quelles valeurs de $x \in S$ la fonction $f(x)$ est strictement supérieure à la probabilité q. Nous avons modifié la structure des algorithmes *CalculerProbabilite*(L, E, a) et *Separation*($f(x)$, q), q) pour faciliter leur lecture et compréhension dans le contexte de nos travaux. La première fonction fait appel aux deux dernières dans son exécution. Pour mieux comprendre son fonctionnement, nous présentons avant l'algorithme des deux dernières fonctions. Abordons donc la description de l'algorithme *CalculerProbabilite*(L, E, a).

L'algorithme *CalculerProbabilite*(L, E, a) qu'illustre la figure 3.3 calcule la fonction qui associe à $x \in S$ la probabilité d'aller vers un état dans $E \subseteq S$ par l'action a. Dans celui-ci, comme dans les autres, L.*Fonctions*() et L.*State*() renvoient l'ensemble des fonctions de probabilité de transitions dans L et S, respectivement. Dans L'algorithme, on parcourt itérativement la partition I pour construire une fonction à branche *Branches* qu'on retourne. Ainsi, pour chaque intervalle $J \in I$, on détermine la fonction de probabilité f qui définit la probabilité d'atteindre E à partir de J et on forme le couple (J, f) qu'on ajoute à *Branches*. Voici en détail l'algorithme *CalculerProbabilite*(L, E, a).

Abordons maintenant la description de l'algorithme *Separation*($f(x)$, q) qu'illustre la figure 3.4. Il prend en entrée une *fonction à branches* f et une probabilité q. On combine *Separation*($f(x)$, q) et *CalculerProbabilite*(L, E, a) pour déterminer les états dans L qui satisfont les propriétés de la forme $\mathbb{P}_q(\langle a \rangle \phi)$ dans la logique $\mathtt{L}_1$. Dans l'algorithme, b.*Domaine*() et b.*Fonction*() renvoient le domaine d'une branche et la fonction de probabilité associée au domaine, respectivement. Aussi, pour tout $X \in I^{\subseteq}(S)$, $B_{inf}(X)$ désigne la borne inférieure de X et $B_{sup}(X)$ la borne supérieure de X.

Separation($f(x)$, q) détermine pour quelles valeurs de $x \in D_f$, f est strictement supérieures à q, c'est-à-dire qu'il retourne l'ensemble $\{x \in D_f \mid f(x) > q\}$. Si E désigne

```
CalculerProbabilite(LMP L, Ensemble E, Action a)
{
  Branches := ∅ ; F := L.Fonction();
  Pour chaque I ∈ Partition(L.State())
      SommeFonction := 0 ;
      Pour chaque E_i ∈ Partition(E)
          Si (∃f ∈ F | I_f ∩ E_i ≠ ∅ : μ_a(I, I_f) > 0)
            SommeFonction := SommeFonction + μ_a(I, E_i) ;
      Fin pour chaque E_i
      Branches := Branches ∪ {(I, SommeFonction)} ;
  Fin Pour chaque I
  Retourner Branches ;
}
```

FIGURE 3.3 – Algorithme : déterminer dans un modèle de LMP la probabilité de transiter d'un état vers un ensemble d'états.

l'ensemble d'états qui satisfont ϕ dans L et f la fonction à branche retournée par *CalculerProbabilite*(L, E, a), alors *Separation*(f, q) permet de ne conserver que les états de L qui satisfont $\langle a \rangle_q \phi$. Pour trouver l'ensemble $\{x \in D_f \mid f(x) > q\}$, on doit déterminer d'abord les points d'intersection entre chacune des branches de f et la droite d'équation $d(x) = q$. Pour cela, l'algorithme itère sur chacune des branches de $f(x)$ et raisonne sur différents cas de figure.

Si la branche courante $g(x)$ est une fonction constante, alors il pourrait y avoir une infinité de points d'intersection. Dans le cas où $g(x)$ est une droite strictement supérieure à q, tout le domaine de $g(x)$ est ajouté à E, l'ensemble des états retourné. Si $g(x)$ n'est pas constante, alors il faut déterminer les valeurs qui annulent la fonction $g(x) - q$. Pour cela, on évoque *TrouverZeros*($g(x) - q$, X) [2] qui effectue une recherche numérique et retourne les zéros de $g(x) - q$ appartenant à la fermeture du domaine X de $g(x)$. Nous donnerons plus loin plus de détails sur l'algorithme *TrouverZeros*($g(x) - q$, X). Une fois *TrouverZeros*($g(x) - q$, X) exécutée, on se base sur le fait que les points d'intersection retournés sont ordonnés (ordre croissant) pour déduire pour quels $x \in X$ de $g(x) > q$. En considérant les bornes inférieure et supérieure de X comme des points d'intersection, on vérifie si $g(x) > q$ pour un x choisit entre deux points d'intersection successifs, $liste[i-1]$ et $liste[i]$. Si c'est le cas, on conserve l'intervalle définie par $liste[i-1]$ et $liste[i]$. Les algorithmes *Separation*($f(x)$, q) et *CalculerProbabilite*(L, E,

2. On suppose que la fonction retourne tous les zéros de $g(x) - q$ appartenant à X.

a) étant présentés, abordons la description de $Sat(L, \phi)$.

Separation(Fonction $f(x)$, $\mathbb{R}$ q)
{
  $E := \emptyset$;
  Pour chaque branche b dans f faire
    $X := b.Domaine()$; $g(x) := b.Fonction()$;
    Si $g(x)$ est constante sur X et $g(x) > q$
      $E := E \cup X$;
    Sinon
      Liste $liste := TrouverZeros(g(x) - q, X)$;
      $n := |liste|$;
      Si $n == 0$
        Si $g(B_{inf}(X)) > q$
          $E := E \cup X$;
      Sinon
        Si $g(B_{inf}(X)) > q$
          Si $B_{inf}(X) \in X$
            $E := E \cup \{[B_{inf}(X), liste[0][\}$;
          Sinon
            $E := E \cup \{]B_{inf}(X), liste[0][\}$;
        Pour i de 1 à $n - 1$
          Si $g(\frac{liste[i-1]+liste[i]}{2}) > q$
            $E := E \cup \{]liste[i-1], liste[i][\}$;
        Si $g(B_{sup}(X)) > q$
          Si $B_{sup}(X) \in X$
            $E := E \cup \{[liste[n-1], B_{sup}(X)[\}$;
          Sinon
            $E := E \cup \{]liste[n-1], B_{sup}(X)]\}$;
          Fin si
    Fin Sinon
  Fin pour chaque branche
  Retourner E ;
}

FIGURE 3.4 – Algorithme : déterminer pour quelles valeurs une fonction à valeurs réelles est supérieure à un réel.

En effet, il ne sert à rien de parcourir tous les états d'un modèle pour ne retenir ceux qui vérifient une propriété. Par exemple, supposons qu'un DTMC possède les propositions atomiques a et b sur ses états et qu'on veuille vérifier la propriété $a \vee b$ dans celui-ci. Une première solution est d'explorer tous les états pour vérifier leurs étiquettes et déduire

par la suite ceux qui satisfont $a \vee b$. Cette méthode est coûteuse en temps et ce serait pire avec un ensemble d'états infini comme pour les LMP. Par contre, il est plus intéressant de trouver les états qui satisfont a et b via deux appels de la fonction d'étiquetage du DTMC et quelques opérations ensemblistes (union, intersection et différence). Cette méthode de vérification se base sur la sémantique des opérateurs logiques formant une propriété pour retrouver les états qui la satisfont dans un modèle. $Sat(L, \phi)$ implémente une telle approche. Il procède de manière récursive pour déterminer l'ensemble d'états satisfaisant ϕ et à la manière de l'algorithme d'évaluation de modèle dans la logique de CTL [13, 49]. L'algorithme de la figure 3.5 illustre $Sat(L, \phi)$ et l'exemple 3.3.5 illustre son exécution avec L le LMP illustré par la figure 3.1 et ϕ la propriété ($\mathtt{T} \vee o$).

Sat(LMP L, Formule ϕ)
{
  Selon ϕ choisir :
    Cas T
      retourner $L.State()$;
    Cas p
      retourner $L.Label(p)$;
    Cas $\neg\psi$
      retourner $L.State() \setminus Sat(L, \psi)$;
    Cas $\psi_1 \wedge \psi_2$
      retourner $Sat(L, \psi_1) \cap Sat(L, \psi_2)$;
    Cas $\psi_1 \vee \psi_2$
      retourner $Sat(L, \psi_1) \cup Sat(L, \psi_2)$;
    Cas $\mathbb{P}_q(\langle a \rangle \psi)$
      $Branches := CalculerProbabilite(S, Sat(L, \psi), a)$;
      retourner $Separation(Branches, q)$;
  Fin choix
}

FIGURE 3.5 – Algorithme : déterminer l'ensemble d'états satisfaisant une propriété dans un LMP.

Exemple 3.3.5. Soit L le LMP que présente la figure 3.1 et déterminons les états qui satisfont $\mathtt{T} \vee o$. Puisque nous sommes en présence d'une propriété avec un $\vee$, nous avons :

- Propriété $\mathtt{T}$: tous les états d'un modèle satisfont toujours $\mathtt{T}$. $Sat(L, \mathtt{T})$ retourne donc $[0, 7] \cup [10, 10]$.
- Propriété o : seul l'état 10 satisfait l'étiquette o. $Sat(L, o)$ retourne donc $[10, 10]$.

– Propriété $\mathtt{T} \vee o$: $Sat(L, \mathtt{T} \vee o)$ retourne $[0, 7] \cup [10, 10]$.

Rappelons que L et $\mathbb{P}_{0.5}(\mathtt{T}\mathtt{U}o)$ sont respectivement le LMP de base et la propriété de base choisis pour étayer nos explications théorique de notre approche. Rappelons également que dans le contexte de notre approche, il faut se contenter des appels $Sat(L, \mathtt{T})$ et $Sat(L\ ,\ o)$ et ne pas unifier leur résultat.

Pour approfondir la description de l'algorithme *Separation*$(f(x),\ q)$ illustré à la figure 3.4, revenons à l'algorithme *TrouverZeros*$(h(x),\ X)$ tiré de [6] et présenté à la figure 3.7 qui effectue une recherche numérique et retourne les zéros de $h(x)$ appartenant à la fermeture de l'intervalle X.

Il existe plusieurs méthodes pour effectuer la recherche numérique des valeurs qui annulent une fonction. La plupart sont des techniques itératives nécessitant une valeur proche d'un zéro de la fonction qu'on étudie pour débuter une recherche. La technique la plus usuelle est celle de Newton [53] mais elle demande le calcul de fonction dérivée. Comme mentionné à la section 3.2, l'implémentation d'une fonction de dérivation est très complexe. Pour contourner une telle difficulté dans CISMO, c'est la méthode de la sécante *MethodeSecante*$(h(x),\ x_0,\ x_1)$ 3.6 dans laquelle on approxime la dérivée numériquement qui est implémentée et $TrouverZeros(h(x),\ X)$ l'utilise pour constituer un ensemble ordonné (croissant) de zéros. La méthode de la sécante est plus lente en convergence que la méthode de Newton mais tout de même meilleure que d'autres méthodes dont la convergence est linéaire. Son taux de convergence est de $\frac{1+\sqrt{5}}{2}$. Pour le fait que *TrouverZeros*$(h(x),\ X)$ appel *MethodeSecante*$(h(x),\ x_0,\ x_1)$ dans son exécution, décrivons d'abord l'algorithme *MethodeSecante*$(h(x),\ x_0,\ x_1)$ tirée de [6].

L'algorithme *MethodeSecante*$(h(x),\ x_0,\ x_1)$ est une adaptation de la méthode de la sécante dans [53] et les problèmes de précision liée aux calculs numériques sont bien abordés dans le mémoire de Richard [6]. Son algorithme présenté à la figure 3.6 prend en entrée la fonction dont on veut trouver les zéros et les bornes x_0 (borne inférieure) et x_1 (borne supérieure) de l'intervalle dans lequel il faut rechercher les zéros. Dans l'algorithme, $\epsilon \in \mathbb{R}$ et $N \in \mathbb{N}$ sont des constantes fixées [3]. ϵ est la précision à observer dans la recherche des zéros et N est le nombre maximale d'itérations de recherches dichotomiques à faire sur un $[x_0, x_1]$. Pour revenir à *TrouverZeros*$(h(x),\ X)$, elle subdivise X en une multitude de segments suffisamment petits sur lesquels elle itère en appelant *MethodeSecante*$(h(x),\ x_0,\ x_1)$ pour trouver tous les zéros de h sur X. Nous

3. Un utilisateur de CISMO peut faire varier ϵ et N via son interface.

utilisons la fonction *Segmenter*(I) pour matérialiser la subdivision d'un intervalle dans son algorithme illustré par la figure 3.7.

MethodeSecante(Fonction $h(x)$, $\mathbb{R}$ x_0, $\mathbb{R}$ x_1)
{
  $n := 2$;
  Tant que $(n-1) < N$ Faire
    $x_{n+1} := x_n - \frac{h(x_n)(x_n - x_{n-1})}{h(x_n) - h(x_{n-1})}$;
    Si $x_{n+1} = 0$ et $\mid x_n \mid < \epsilon$;
      Retourner $\{x_{n+1}\}$;
    Sinon
      Si $\mid \frac{x_{n+1} - x_n}{x_{n+1}} < \epsilon \mid$
        Retourner $\{x_{n+1}\}$;
  Fin Tant que
  Retourner $\emptyset$;
}

FIGURE 3.6 – Algorithme : trouver un zéro d'une fonction sur un intervalle.

TrouverZeros(Fonction $h(x)$, Intervalle I)
{
  $E := Segmenter(I)$; $Zeros := \emptyset$; $i := 0$
  Tant que $i <\mid E \mid$ Faire
    $\lambda = E[i]$;
    $E = E \setminus E[i]$;
    $i = i + 1$;
    $Zeros := Zeros \cup MethodeSecante(h(x), B_{inf}(\lambda), B_{sup}(\lambda))$;
  Fin Tant que
  Retourner $Zeros$;
}

FIGURE 3.7 – Algorithme : déterminer les zéros d'une fonction sur un intervalle

De tout ce qui précède, il est à retenir que nous avons décrit la première étape de notre approche. Elle consiste à retrouver les états de L qui satisfont β. Nous avons présenté les algorithmes de Richard [6] qui se complètent pour assumer algorithmiquement de cette première étape. L'algorithme moteur est $Sat(L, \phi)$. Pour les détails d'implémentation, d'exemples de trace et les problèmes rencontrés suite à l'implémentation des algorithmes présentés dans cette section, le lecteur peut se référer à [6, 53]. La première étape

de l'approche d'atteignabilité moyenne étant présentée, nous abordons à la prochaine section la deuxième étape.

3.3.3 Construction de DTMC moyen associé à un LMP

Nous considérons toujours la propriété d'atteignabilité $\beta = \mathbb{P}_q(\phi U \psi)$ et le LMP $L = (S, I, i, Act, Label, \{\mu_a \mid a \in Act\})$ dans nos explications théoriques et consacrons la section à la construction d'un DTMC moyen à partir des états issus de l'application de la première étape de notre approche.

Avant d'aborder les détails de la deuxième étape, rappelons au lecteur que, lorsqu'on s'intéresse à une propriété d'atteignabilité dans un modèle, on focalise sur les propriétés que satisfont les états et on reste indifférent aux actions sur les transitions car elles peuvent être traduites par des propositions atomiques. Observons également que dans le contexte de l'approche d'atteignabilité moyenne, les propositions atomiques qui étiquettes les états de L peuvent être ignorées dans un DTMC moyen qui l'approxime pour β. Ceci se justifie par le fait que nous sommes certains que les états qui participent à la construction du DTMC respectent ϕ ou ψ. Toutefois, il est nécessaire d'identifier l'état initial du LMP et les états cibles dans le DTMC pour pourvoir formuler la propriété d'atteignabilité à vérifier lorsque nous passons le DTMC moyen à un vérificateur de système fini à la troisième étape de notre approche. Eu égard aux remarques évoquées, nous ignorons les actions et les propositions atomiques dans nos algorithmes dans cette section.

Puisque seuls les états satisfaisant ϕ ou ψ nous intéressent par la suite, alors nous pouvons nous limiter à ceux-ci et extraire de L un sous-LMP respectant la définition 3.3.1 et qui est en réalité le DTMC moyen que nous envisageons. Dans la définition 3.3.1 la fonction $EtatsDTMC(L, \phi, \psi)$ dénote le partionnement de l'ensemble des états satisfaisant ϕ ou ψ et nous précisons son algorithme plus loin. Rappelons que les fonctions de transition de L sont de la forme $f(x)U(a, b)$: la fonction $f(x)$ représente un facteur, et celui-ci multiplie la distribution uniforme sur $[a, b]$ (de probabilité totale 1).

Définition 3.3.1. Soit $L = (S, I, i, Act, Label, \{\mu_a \mid a \in Act\})$ un LMP, ϕ et ψ des propriétés de $\mathtt{L}_0$. Le DTMC moyen associé $L_{\phi U \psi}$ est défini comme suit :

$$L_{\phi U \psi} = (EtatsDTMC(L, \phi, \psi), i, \mu', AP, Label)$$

où, pour $I_1, I_2 \in EtatsDTMC(S, \phi, \psi)$ ayant comme fonction de transition $f(x)U(J_2)$

de I_1 vers $I_2 \subseteq J_2$,

$$\begin{aligned}\mu'(I_1, I_2) &= \frac{|I_2|}{|J_2|}\frac{1}{|I_1|}\int_{I_1} f(x)dx. \\ &= \frac{|I_2|}{|J_2|}M_f^{I_1}.\end{aligned}$$

Le facteur $\frac{|I_2|}{|J_2|}$ normalise la valeur de la distribution uniforme : celle-ci est définie sur J_2 mais est appliquée sur l'intervalle I_2. Dans cette définition, les états sont des intervalles, mais ces intervalles ne sont qu'une étiquette pour les états, leur structure d'intervalle est inutilisée.

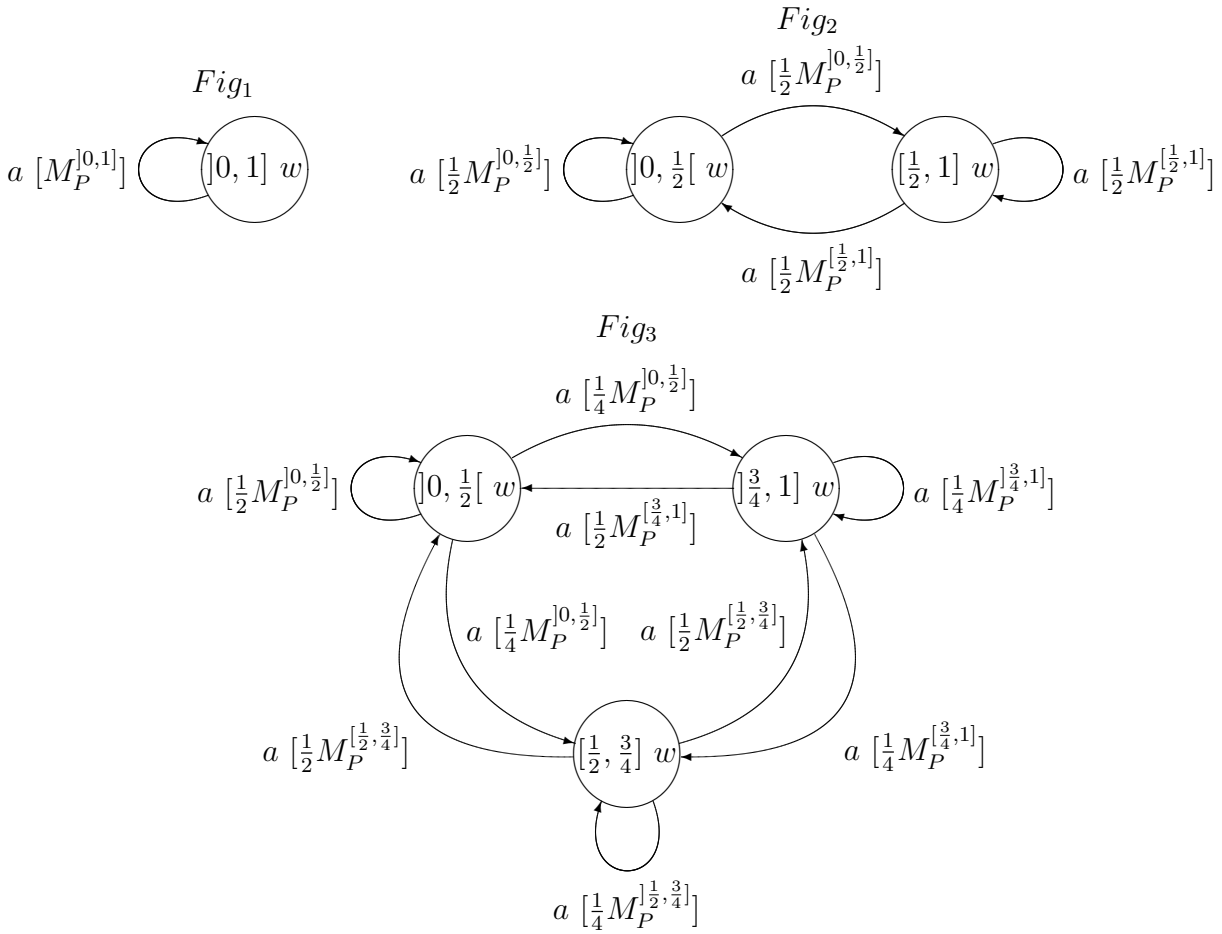
Remarquons que du fait que les LMP sont définis avec des fonctions de transitions probabilistes partielles (voir la définition 2.1.4), la probabilité de transiter d'un état du DTMC moyen vers les autres états peut ne pas donner 1. Nous présentons dans cette section un algorithme qui génère le DTMC moyen après avoir illustré la définition 3.3.1.

Pour donner une idée concrète d'un DTMC moyen qui respecte la définition 3.3.1 et convaincre de ce qu'il est possible d'en construire une multitude dans certains cas à partir d'états satisfaisants ϕ ou ψ, nous illustrons d'abord des exemples de DTMC moyen. Ensuite, nous précisons notre façon déterministe d'en générer un dans l'algorithme que nous proposons.

Soit L le LMP illustré par la figure 3.1 de la page 53 et 0 son état initial. Supposons $Sat(L,\, \phi) =]0, \frac{3}{4}]$ et $Sat(L,\, \psi) = [\frac{1}{2}, 1]$ et présentons trois façons de construire $L_{\phi U \psi}$ à partir de $]0, \frac{3}{4}]$ et $[\frac{1}{2}, 1]$:

- Première solution : en choisissant de construire $L_{\phi U \psi}$ avec $Partition(Sat(\phi) \cup Sat(\psi)) =]0, 1]$, nous obtenons le DTMC Fig_1 qu'illustre la figure 3.8.
- Deuxième solution : notons que $Sat(L,\, \phi)$ et $Sat(L,\, \psi)$ ne sont pas disjoints. En choisissant de construire $L_{\phi U \psi}$ avec $Partition([\frac{1}{2}, 1]) = [\frac{1}{2}, 1]$ et de $Partition([0, \frac{1}{2}[) = [0, \frac{1}{2}[$, nous obtenons le DTMC Fig_2 qu'illustre la figure 3.8.
- Troisième solution : Étant donné que $Sat(\phi)$ et $Sat(\phi)$ ne sont pas disjoints, nous essayons de les découper en des intervalles disjoints. Il existe une infinité de possibilités de découpage mais nous en proposons une. En choisissant de construire $L_{\phi U \psi}$ avec $Partition(Sat(\phi) \backslash (Sat(\psi) \cap Sat(\phi)) = [0, \frac{1}{2}[$, $Partition(Sat(\psi) \backslash (Sat(\psi) \cap Sat(\phi)) =]\frac{3}{4}, 1[$ et $Partition(Sat(\psi) \cap Sat(\phi)) = [\frac{1}{2}, \frac{3}{4}]$, nous obtenons le DTMC Fig_3 qu'illustre la figure 3.8.

Rappelons qu'à l'implémentation nous devons approximer numériquement la moyenne de chaque fonction de probabilité de transition dans $L_{\phi U \psi}$ par un ensemble fini de



La variable x représente l'état de départ d'une transition et $P(x) \sim \frac{x+1}{4}U(0,1)$.

FIGURE 3.8 – Exemples de DTMC moyen issu du LMP illustré par la figure 3.1.

points choisis dans son domaine de définition. À cet effet, plus un intervalle dans un DTMC moyen est réduit, mieux la moyenne d'une fonction de transition définie sur celui-ci est précise. Comme nous pouvons le noter sur les DTMC moyens qu'illustre la figure 3.8, contrairement à Fig_1, Fig_2 distingue bien les états à atteindre en partant de 0. Par contre dans Fig_3 l'intervalle $[0,1]$ est plus morcelé. La troisième solution est donc meilleure que les 2 autres. Dans un grand modèle, elle favorisera plus l'obtention d'intervalles petits et donc des résultats plus précis lors des vérifications.

Des exemples de DTMC moyen, nous retenons que mieux on choisit le découpage de $Sat(L,\ \psi)$ et $Sat(L,\ \phi)$ plus la probabilité d'atteignabilité de $\phi U \psi$ dans le DTMC moyen se rapprochera de la probabilité réelle dans L. Dans ce mémoire, notre façon

déterministe de découper $Sat(L,\, \psi)$ et $Sat(L,\, \phi)$ est à l'image de la stratégie utilisée pour obtenir Fig_3 et que nous traduisons dans la fonction $EtatsDTMC(L,\, \phi,\, \psi)$ dont l'algorithme est illustré par la figure 3.3.6.

$EtatsDTMC$(LMP L, Formule ϕ, Formule ψ)
{
  $I_1 := Partition((Sat(L,\, \phi) \backslash (Sat(L,\, \psi) \cap Sat(L,\, \phi))))$;
  $I_2 := Partition((Sat(L,\, \psi) \backslash (Sat(L,\, \psi) \cap Sat(L,\, \phi))))$;
  $I_3 := Partition((Sat(L,\, \psi) \cap Sat(L,\, \phi)))$;
  Retourner $I_1 \cup I_2 \cup I_3$;
}

FIGURE 3.9 – Algorithme : partitionner un ensembles d'états pour construire un DTMC.

Le prochain exemple montre une trace de $EtatsDTMC(L, \psi, \phi)$. L'algorithme s'applique au LMP illustré par la figure 3.1 avec la propriété $\mathtt{TU}o$ et nous exploitions le résultat des différentes partitions de l'exemple par la suite.

Exemple 3.3.6. Soit L le LMP illustré par la figure 3.1 et calculons $EtatsDTMC(L,\mathtt{T},o)$. Puisque $I = \{[0,0],\,]0,1],\,]2,3],\,]1,2],\,]5,7],\,]3,5],\, [10,10]\}$, $Sat(L,\, \mathtt{T}) = [0,7] \cup [10,10]$ et $Sat(L,\, o) = [10,10]$ on a donc :

- $I_1 = Partition(([0,7] \cup [10,10]) \backslash [10,10]) = \{[0,0],\,]0,1],\,]2,3],\,]1,2],\,]5,7],\,]3,5]\}$
- $I_2 = Partition([10,10] \backslash [10,10]) = \{\}$
- $I_3 = Partition([10,10]) = \{[10,10]\}$
- Retourner : $\{[0,0],\,]0,1],\,]2,3],\,]1,2],\,]5,7],\,]3,5],\, [10,10]\}$

Le résultat donne exactement les intervalles sur lesquels les transitions de L sont définies. Il faut juste alors remplacer ses fonctions de transitions par leur moyenne pour obtenir le DTMC moyen qui l'approxime.

Le but poursuivi est de définir à partir d'un LMP un DTMC moyen qui respecte la définition 3.3.1. La façon de restructurer les états issus de la première étape de notre approche étant fixée, abordons de façon pratique le calcul de la moyenne d'une fonction de transition (fonction de répartition) non constante.

Soient L le LMP illustré par la figure 3.1 et $\mathtt{TU}o$ la propriété à évaluer dans L. En nous référant à l'exemple 3.3.6, le sous-LMP à partir duquel nous pouvons définir le

DTMC moyen est L. Remplaçons alors ses fonctions de transition non constantes par leur valeur moyenne.

Notons que dans L la probabilité P de transiter de $x \in [0,1[$ vers $[0,1[$ suit l'uniforme $\frac{x+1}{4}U(0,1)$. Pour calculer $M_P^{[0,1[} = M^{[0,1[}_{\frac{x+1}{4}U(0,1)}$, c'est-à dire la moyenne de P sur $[0,1]$, nous procédons comme suit :

$M_P^{[0,1[}$

$\langle$*Application du théorème de la moyenne*$\rangle$

$= \frac{1}{1-0}\int_0^1 \frac{x+1}{4}U(0,1)$

$\langle$*Application de la loi uniforme U(0,1)*$\rangle$

$= \frac{1}{1-0}\int_0^1 (\frac{x+1}{4} \times \frac{1-0}{1-0})$

$\langle$*Simplification*$\rangle$

$= \frac{1}{4}\int_0^1 (x+1)$

$\langle$*Résultat*$\rangle$

$= \frac{3}{8}$

La moyenne de $\frac{x+1}{4}U(0,1)$ sur $[0,1]$ est donc $M^{[0,1[}_{\frac{x+1}{4}U(0,1)} = \frac{3}{8}$. En procédant comme ci-dessus, nous obtenons les autres probabilités moyennes de transition que nous énumérons ci- dessous :

$$M^{(0,1]}_{\frac{1}{4}U(1,2)} = \tfrac{1}{4}\,; \qquad M^{(0,1]}_{\frac{x}{4}U(2,3)} = \tfrac{1}{8}\,; \qquad M^{(1,2]}_{\frac{z^2}{4}U(5,7)} = \tfrac{7}{12}\,; \qquad M^{\{0\}}_{\frac{1}{4}} = \tfrac{1}{4}\,;$$

$$M^{(1,2]}_{\frac{4-z^2}{4}U(3,5)} = \tfrac{5}{12}\,; \qquad M^{(2,3]}_{\frac{y}{4}U(5,7)} = \tfrac{5}{8}\,; \qquad M^{(5,7]}_{\frac{p^2}{100}} = \tfrac{109}{150}$$

Dans un DTMC, comme souligné plus haut, il n'y a aucun intérêt à définir un état sous forme d'intervalle. Ainsi, en remplaçant les intervalles dans le LMP illustré par la figure 3.1 par des noms d'états (s_1 pour $[0,1]$ par exemple) à l'image de ce qui se fait en évaluation de modèle classique et leur associant adéquatement les propositions atomiques (w étiquette s_1 par exemple) nous obtenons le DTMC moyen illustré par la figure 3.2.

Nous avons maintenant tous les moyens pour comprendre l'algorithme $\widehat{L_\beta}(L,\ \phi U \psi)$ illustré par la figure 3.10 qui génère le DTMC moyen à partir de L et $\phi U \psi$. Abordons

donc son détail.

Dans l'algorithme 3.10, $Moyenne(I_1, f, I_2)$ matérialise la moyenne d'une fonction f sur l'intervalle I_1 qui définit la probabilité de transiter dans l'intervalle I_2 et le DTMC S_{fini} retourné est encodé sous forme d'ensemble de transitions. Chaque transition est un triplet (I_1, P, I_2) où l'intervalle I_1 représente l'état de départ de la transition, P est la probabilité moyenne de transition et l'intervalle I_2 représente l'état d'arrivée. Pour chaque intervalle d'arrivée I_2, l'algorithme recherche les intervalles prédécesseurs dans L et ajoute pour chacun une transition dont la probabilité est donnée par la moyenne sur l'intervalle I_1. L'algorithme implémente la définition 3.3.1 sans avoir construit séparément de sous-LMP. Pour cela, il itère sur $EtatsDTMC(L,\ \psi,\ \phi)$ et calcule les probabilités de transition en même temps qu'il construit le sous-LMP. L'usage de la fonction prédécesseur définie au 3.3.1 nous aide à définir la structure du modèle de S_{fini}. $\wp_{re}(X)$ se calcule dans L et seuls les intervalles dans $S_{\phi U\psi}$ participent à la construction de S_{fini}.

$\widehat{L_\beta}$(LMP L, Formule $\phi U\psi$)
{
  $S_{\phi U\psi} := EtatsDTMC(L,\ \psi,\ \phi)$;
  $S_{fini} := \emptyset$; $E := S_{\phi U\psi}$;
  $F := L.Fonctions()$; i := 0 ;
  Tant que $E \neq \emptyset$
    $I_2 = E[i]$; $E := E \backslash E[i]$
    $\Upsilon := Restrict(\wp_{re}(I_2), S_{\phi U\psi})$;
    Pour chaque $I_1 \in \Upsilon$
      Si $(\exists f \in F \mid I_2 \subseteq I_f \wedge I_1 \subseteq D_f)$
        $S_{fini} := S_{fini} \cup \{(I_1, Moyenne(I_1, f, I_2), I_2)\}$;
    Fin pour chaque
    i := i + 1 ;
  Fin Tant que
  Retourner S_{fini} ;
}

FIGURE 3.10 – Algorithme de génération de DTMC à partir d'un LMP et une formule d'atteignabilité

Pour renforcer la compréhension du lecteur de l'algorithme 3.10, nous l'appliquons au LMP illustré par la figure 3.12 à la page 92 pour montrer comment le DTMC moyen L_{bUc} de la même figure est obtenu à partir de la propriété d'atteignabilité bUc. Ci-dessous donc la procédure :

1. Trouvons les états qui satisfont les propositions atomiques b ou c : $Sat(L,\, b) = \{[0,2] \cup [4,4]\}$ et $Sat(L,\, c) = \{[\frac{5}{2}, 3] \cup [4,4]\}$.
2. Étant donné que $([\frac{5}{2}, 3] \cup [4,4]) \cap ([0,2] \cup [4,4]) = [4,4]$, alors l'ensemble $E = S_{bUc} = \{[0,2], [\frac{5}{2}, 3], [4,4]\}$.
3. À cette étape, nous pouvons établir l'ensemble S_{fini} des transitions de L_{bUc} à partir de S_{bUc} et L en itérant sur chaque $I_2 \in S_{bUc}$:
   a) $I_2 = [0,2]$: l'ensemble des prédécesseurs de I_2 dans L est $\wp_{re}(I_2) = \{[0,3], [4,4]\}$. Lorsque nous le raffinons par rapport à S_{bUc} avec la fonction *Restrict* nous obtenons $\Upsilon = \{[\frac{5}{2}, 3], [0,2], [4,4]\}$ et $S_{fini} = \{([0,2], \frac{2}{9},\ [0,2]),\ ([\frac{5}{2}, 3], \frac{11}{18}, [0,2]),\ ([4,4], 1, [0,2])\}$. À cette étape, $E = \{[\frac{5}{2}, 3], [4,4]\}$ et $i = 1$.
   b) $I_2 = [\frac{5}{2}, 3]$: en procédant comme ci-dessus, nous obtenons $\Upsilon = \{[\frac{5}{2}, 3], [0,2]\}$ car de $[4,4]$ on ne peut qu'atteindre $[0,2]$. À ce niveau, $S_{fini} = \{([0,2], \frac{2}{9}, [0,2]),\ ([\frac{5}{2}, 3], \frac{11}{18}, [0,2]),\ ([\frac{5}{2}, 3], \frac{11}{72}, [\frac{5}{2}, 3]),\ ([0,2], \frac{1}{18}, [\frac{5}{2}, 3]),\ ([4,4], 1, [0,2])\}$, $E = \{[4,4]\}$ et $i = 2$.
   c) $I_2 = [4,4]$: à ce stade, $E = \emptyset$ et $[4,4]$ ne possède pas de prédécesseur. Nous devons donc arrêter l'itération.
4. Nous retournons S_{fini} qui représente L_{bUc}.

Nous sommes à la fin de la deuxième étape de l'approche d'atteignabilité moyenne. L'objectif visé étant de vérifier β dans L via un DTMC moyen, nous devons calculer la probabilité de l'ensemble des chemins dans S_{fini} qui vérifient $\phi U \psi$ et la comparer à q pour décider si L satisfait β ou pas. Autrement dit notre approche est exacte théoriquement si le théorème 3.3.1 est vrai.

Théorème 3.3.1. Soit L un LMP et $L_{\phi U \psi}$ un DTMC moyen associé. Alors $Prob_L(\{\pi \in \text{Path}(L) \mid \pi \models \phi U \psi\} = Prob_{L_{\phi U \psi}}(\{\pi \in \text{Path}(L_{\phi U \psi}) \mid \pi \models \phi U \psi\})$.

Pour démontrer le théorème 3.3.1, nous aurons besoin de deux notations qui étendent celles que nous avons définies. Nous avons vu la notion de chemins à la définition 2.4.1. Nous généralisons cette définition aux LMP pour représenter des ensembles de chemins qui passent par les mêmes intervalles de I. Soient un LMP L et $I_1, I_2, \ldots, I_n$ une suite d'intervalles de L, on définit l'ensemble des chemins suivant :

$$i\, I_1 I_2 I_3 \ldots I_n := \{i\, a_1\, i_1\, a_2\, i_2\, \ldots a_n\, i_n \in \text{Path}_L(i) \mid i_j \in I_j\}.$$

On dit qu'un ensemble de chemins X satisfait une propriété φ, et on écrit $X \models \varphi$ si tous les chemins de l'ensemble la satisfont.

Démonstration : Soient $I_1, I_2, \ldots, I_n$ des états de $L_{\phi U \psi}$, donc des intervalles dans $I^{\subseteq}(S)$. Les fonctions de transitions entre ces intervalles sont héritées de L comme suit. Soient $J_1, J_2, \ldots, J_n \in I$ tels que $I_i \subseteq J_i$, et $f, f_1, f_2, f_3 \ldots, f_{n-1}$ les fonctions de L telles que $f_i U(J_{i+1})$ est la fonction de transition de J_i à J_{i+1} dans L, pour $i = 1, \ldots, n$ alors que $fU(J_1)$ est la fonction de transition de i vers I_1.

La première d'égalité ci-dessous repose sur le fait que la probabilité des chemins qui commencent en i est égale à la probabilité des chemins qui commencent sur x_1 dans I_1 pondérée par la probabilité de transiter de i vers ces x_1 (i.e., $f(i)dU(J_1)(x_1)$), ceux-ci étant en nombre infini, non dénombrables : il faut donc prendre l'intégrale ; la probabilité uniforme $dU(J_1)(x_1)$ doit tenir compte de la taille de J_1, d'où le facteur $\frac{1}{|J_1|}$ à la deuxième égalité, suivi de la mesure le Lebesgue habituelle : dx_1.

$$
\begin{aligned}
&Prob_L(iI_1I_2\dots I_n)\\
&= \int_{I_1} Prob_L(x_1I_2\dots I_n)f(i)dU(J_1)(x_1)\\
&= \int_{I_1} Prob_L(x_1I_2\dots I_n)f(i)\frac{1}{|J_1|}dx_1\\
&= f(i)\frac{1}{|J_1|}\int_{I_1} Prob_L(x_1I_2\dots I_n)\ dx_1\\
&= f(i)\frac{1}{|J_1|}\int_{I_1}(\int_{I_2} Prob_L(x_2I_3\dots I_n)f_1(x_1)dU(J_2)(x_2))\ dx_1\\
&= f(i)\frac{1}{|J_1|}\int_{I_1}(\int_{I_2} Prob_L(x_2I_3\dots I_n)f_1(x_1)\frac{1}{|J_2|}dx_2)\ dx_1\\
&= f(i)(\frac{1}{|J_1|}\int_{I_1} f_1(x_1)dx_1)(\frac{1}{|J_2|}\int_{I_2} Prob_L(x_2I_3\dots I_n)dx_2)\\
&\vdots\\
&= f(i)(\frac{1}{|J_1|}\int_{I_1} f_1(x_1)dx_1)(\frac{1}{|J_2|}\int_{I_2} f_2(x_2)dx_2)\dots(\frac{1}{|J_{n-2}|}\int_{I_{n-2}} f_{n-2}(x_{n-2})dx_{n-2})\\
&\qquad\cdot\ (\int_{I_{n-1}} Prob_L(x_{n-1}I_n)dU(J_{n-1})(x_{n-1})\\
&= f(i)(\frac{1}{|J_1|}\int_{I_1} f_1(x_1)dx_1)(\frac{1}{|J_2|}\int_{I_2} f_2(x_2)dx_2)\dots(\frac{1}{|J_{n-2}|}\int_{I_{n-2}} f_{n-2}(x_{n-2})dx_{n-2})\\
&\qquad\cdot\ (\int_{I_{n-1}} Prob_L(x_{n-1}I_n)\frac{1}{|J_{n-1}|}dx_{n-1})\\
&= f(i)(\frac{1}{|J_1|}\int_{I_1} f_1(x_1)dx_1)(\frac{1}{|J_2|}\int_{I_2} f_2(x_2)dx_2)\dots(\frac{1}{|J_{n-2}|}\int_{I_{n-2}} f_{n-2}(x_{n-2})dx_{n-2})\\
&\qquad\cdot\ (\frac{1}{|J_{n-1}|}\int_{I_{n-1}} f_{n-1}(x_{n-1})\frac{|I_n|}{|J_n|}dx_{n-1})\\
&= f(i)(\frac{1}{|J_1|}\ |I_1|\ M_{f_1}^{I_1})\ (\frac{1}{|J_2|}\ |I_2|\ M_{f_2}^{I_2})\ \dots(\frac{1}{|J_{n-2}|}\ |I_{n-2}|\ M_{f_{n-2}}^{I_{n-2}})\ (\frac{1}{|J_{n-1}|}\ |I_{n-1}|\ M_{f_{n-1}}^{I_{n-1}}\frac{|I_n|}{|J_n|})\\
&= \frac{|I_1|}{|J_1|}f(i)\ \frac{|I_2|}{|J_2|}M_f^{I_1}\ \cdots\ \frac{|I_n|}{|J_n|}M_{f_{n-1}}^{I_{n-1}}\\
&= \mu'(i,I_1)\ \mu'(I_1,I_2)\ \cdots\ \mu'(I_{n-1},I_n)\\
&= Prob_{L_{\phi U\psi}}(iI_1I_2\dots I_n).
\end{aligned}
$$

Notons que $Prob_L(x_{n-1}I_n) = \mu(x_{n-1},I_n) = f_{n-1}(x_{n-1})\frac{|I_n|}{|J_n|}$ car la fonction de I_{n-1} vers I_n est $f_{n-1}U(J_n)$.

Nous pouvons maintenant terminer la démonstration. La première égalité ci-dessous vient du fait que les chemins peuvent être regroupés selon les intervalles qu'ils traversent

dans L. Ces ensembles sont tous disjoints, d'où la deuxième égalité.

$$
\begin{aligned}
& Prob_L(\{\pi \in Paths(L) \mid \pi \models \phi U \psi\} \\
&= Prob_L(\bigcup_{iI_1I_2\ldots I_n \models \phi U \psi} iI_1I_2\ldots I_n) \\
&= \sum_{iI_1I_2\ldots I_n \models \phi U \psi} Prob_L(iI_1I_2\ldots I_n) \\
&= \sum_{iI_1I_2\ldots I_n \models \phi U \psi} Prob_{L_{\phi U \psi}}(iI_1I_2\ldots I_n) \\
&= Prob_{L_{\phi U \psi}}(\{\pi \in \mathrm{Path}(L_{\phi U \psi}) \mid \pi \models \phi U \psi\}.
\end{aligned}
$$

Étant donné que nous avons prouvé le théorème 3.3.1 nous sommes convaincus que tout DTMC moyen obtenu théoriquement d'un LMP par les démarches présentées jusqu'ici est équivalent à celui-ci de point de vue atteignabilité moyenne pour la propriété étudiée.

La prochaine section présente la dernière étape de notre approche, c'est-à-dire comment nous évaluons la probabilité de l'ensemble des chemins qui satisfont $\phi U \psi$ dans $L_{\phi U \psi}$.

3.3.4 Évaluation de propriétés d'atteignabilité dans un DTMC

Le but poursuivi est de décider de la satisfiabilité d'une propriété d'atteignabilité $\beta = \mathbb{P}_q(\phi U \psi)$ dans un DTMC L_β construit à partir d'un LMP L.

Les deux premières étapes de l'approche d'atteignabilité moyenne permettent d'approximer L par un DTMC moyen L_β bien choisi pour évaluer β. L_β est construit de telle sorte qu'il est possible d'inférer si une formule est satisfaite par L, à partir du fait qu'elle est satisfaite par le DTMC. Il existe aujourd'hui des vérificateurs [25, 10] pour faire la vérification formelle de β dans de systèmes probabilistes finis, nous évitant ainsi d'implémenter des algorithmes de vérification de systèmes finis. La dernière étape de notre approche exploite cet acquis.

Nous articulons cette section autour de deux points. Le premier point traite de la dernière étape de notre approche, c'est-à dire l'évaluation de β dans L_β dans le vérificateur PRISM [10]. Par souci de complétion, dans le deuxième point, nous donnons les moyens aux lecteurs qui le désirent de vérifier manuellement si L_β satisfait β.

Abordons le premier point à travers un exemple. Soit L le LMP de la figure 3.1 à la page 53 et vérifions si L satisfait $\mathbb{P}_{0.5}(\mathtt{T}\,U\,o)$. Le DTMC illustré par la figure 3.2 étant

celui qui approxime L (voir exemple 3.3.6), nous allons donc vérifier avec PRISM s'il satisfait $\mathbb{P}_{0.5}(\mathtt{T}Uo)$ et décider de la satisfiabilité de $\mathbb{P}_{0.5}(\mathtt{T}Uo)$ par L.

Rappelons au lecteur que dans les DTMC que nous construisons, la probabilité de transiter d'un état vers les autres états peut ne pas donner 1 (voir la définition 2.1.4). Pour cela, il est important de prendre en compte cette contrainte sur leur modèle lors d'une vérification. On y arrive dans PRISM en décochant *Do probabilityrate check* dans la liste des contraintes [4].

Dans PRISM, l'ensemble des états d'un DTMC est défini usuelement par une liste d'entiers à laquelle on associe une variable dont la valeur initiale est précisée. La notation adoptée est $x : [v_1..v_2]$ *init* v avec $(v_1, v_2) \in \mathbb{N}^2$ et $v \in [v_1..v_2]$. Par exemple, $x : [1..6]$ *init 1* signifie que nous déclarons 6 états à manipuler avec la variable x que nous initialisons à 1, l'état initial. Une transition se définit à partir de x et des probabilités de transitions. On écrit $[a]\ x = z_1 -> p_1 : x' = z_2 +...+ p_2 : x' = z_3$ pour signifier qu'à partir de l'état z_1 on effectue l'action a et on peut transiter vers z_2 avec p_1 ou vers z_3 avec p_2, etc. x dénote l'état courant du DTMC et x' l'état ultime d'une transition. Dans PRISM, on termine chaque instruction par un point virgule (;), on utilise *label* pour étiqueter les états par le biais de la variable x, on délimite le code des transitions par *module* et *endmodule* et on fait suivre le délimiteur *module* d'un identificateur représentant le nom du DTMC. Ce nom est très utile pour synchroniser des modules de DTMC.

Le code dans PRISM illustré par la figure 3.11 est celui du DTMC illustré par la figure 3.2. Pour vérifier la propriété $\mathbb{P}_{0.5}(\mathtt{T}Uo)$ sur celui-ci, PRISM propose deux alternatives :

1. P>0.5 [true U "o"] : Formulée de cette façon, PRISM répond oui si la propriété est vérifiée et non dans le cas contraire. Pour notre exemple on obtient oui.

2. Pmin=? [true U "o"] : Formulée de cette façon, PRISM donne la probabilité minimale avec laquelle la propriété est satisfaite. Dans notre cas on obtient 1. Ce qui est bien supérieure à 0.5 et confirme la réponse obtenue avec P>0.5 [true U "o"].

Nous sommes à la fin de la description de l'approche d'atteignabilité moyenne. Par souci de complétion, avant de clore la section, nous décrirons une approche adoptée par les concepteurs de PRISM [10] pour calculer des probabilités de propriétés d'atteignabilité dans un DTMC. Le lecteur qui désire s'exercer pour comprendre le travail que fait PRISM pour décider de la satisfiabilité d'une propriété d'atteignabilité par un DTMC peut lire l'approche.

4. Voir la rubrique options au niveau du logiciel PRISM.

label "v" = $x = 0$;
label "w" = $x = 1$;
label "c" = $x = 2$;
label "u" = $x = 3$;
label "r" = $x = 4$;
label "t" = $x = 5$;
label "o" = $x = 6$;
module MonDTMC
$x : [0..6]$ init 0;
$[a]\ x = 0 -> 0.25 : (x' = 1) + 0.25 : (x' = 3)$;
$[a]\ x = 1 -> 0.375 : (x' = 1) + 0.125 : (x' = 2) + 0.25 : (x' = 3)$;
$[b]\ x = 2 -> 0.625 : (x' = 4)$;
$[b]\ x = 3 -> 0.583 : (x' = 4) + 0.416 : (x' = 5)$;
$[e]\ x = 4 -> 0.363 : (x' = 6)$;
$[a]\ x = 5 -> 1 : (x' = 6)$;
endmodule

FIGURE 3.11 – Code du DTMC illustré par la figure 3.2 dans PRISM.

Il existe plusieurs techniques [54, 55, 44, 56] pour calculer la probabilité d'une propriété d'atteignbilité dans un MDP. Mais dans ce mémoire, nous présentons sur deux étapes et dans un contexte de calcul de probabilités maximales dans un DTMC $M = (S, i, \mu, AP, Act, L)$, la méthode d'approximation successive par itération de valeurs [56].

La première étape consiste à séparer l'ensemble d'états $S^0_{\max}$ qui atteignent un ensemble d'états cible E avec une probabilité de 0% de l'ensemble d'états $S^1_{\max}$ qui atteignent E avec une probabilité de 100%. De nos jours, il existe plusieurs ouvrages qui présentent des algorithmes permettant d'atteindre ce but. Nous n'en exposons pas mais le lecteur peut se référer à [13, 55, 57].

La deuxième étape traite de la probabilité d'atteignabilité de E par les états qui ne sont ni dans $S^0_{\max}$ ni dans $S^1_{\max}$. La résolution du système d'équations ci-dessous donne ces probabilités.

Définition 3.3.2. [56] Soient $M = (S, i, \mu, AP, Act, L)$ un DTMC et $T \subseteq S$ une cible à atteindre. Pour tout $s \in S$, nommons x_s la probabilité d'atteindre T à partir de s s'obtient en résolvant le système d'équation suivant :

$$x_s = \begin{cases} 1 & si \quad s \in S^1_{\max} \\ 0 & si \quad s \in S^0_{\max} \\ \sum\limits_{s' \in S, a \in Act} \mu_a(s, s') \times x_{s'} & si \quad s \notin S^1_{\max} \cup S^0_{\max} \end{cases}$$

Pour éclaircir les idées, nous illustrons la définition ci-dessus avec le DTMC que présente la figure 3.2.

Exemple 3.3.7. Considérons le DTMC que présente la figure 3.2 et soit P_s la probabilité d'atteindre l'état s_6 à partir d'un état $s \in \{s_1, s_2, s_3, s_4, s_5, s_6\}$. En appliquant la définition 3.3.2 nous avons :

$$\begin{cases} P_{s_0} = \frac{1}{4}P_{s_1} + \frac{1}{4}P_{s_3} \\ P_{s_1} = \frac{3}{8}P_{s_1} + \frac{1}{8}P_{s_2} + \frac{1}{4}P_{s_3} \\ P_{s_3} = \frac{7}{12}P_{s_4} + \frac{5}{12}P_{s_5} \\ P_{s_2} = \frac{5}{8}P_{s_4} \\ P_{s_4} = \frac{109}{300}P_{s_6} \\ P_{s_5} = P_{s_6} = 1 \end{cases}$$

Le système d'équations défini au 3.3.2 à une solution unique d'après les résultats de [58]. Contrairement aux autres méthodes, l'approximation de probabilités par itération de valeurs offre une meilleure appréciation de l'évolutivité des probabilités d'atteignabilité de chacun des états d'un système mais au détriment d'une grande précision de point de vue implémentation. Au lieu de calculer une solution de l'ensemble du système d'équations de la définition 3.3.2, on le résout en $n \in \mathbb{N}$ étapes. En pratique, pour n assez grand, on obtient une bonne approximation. La définition 3.3.3 reformule celle du 3.3.2 et intègre la notion d'itération dans le calcul des probabilités. Nous écrivons donc x^n_s au lieu de x_s pour désigner la probabilité de x_s à l'étape n.

Définition 3.3.3. Soient $M = (S, i, \mu, AP, Act, L)$ un DTMC et $T \subseteq S$ une cible à atteindre. Pour tout $s \in S$, nommons x^n_s la probabilité d'atteindre T à partir de s en

$n \in \mathbb{N}$ itérations. Le système d'équations de Bellman est la suivante :

$$x_s^n = \begin{cases} 1 & si \quad s \in S^1_{\max} \\ 0 & si \quad s \in S^0_{\max} \\ 0 & si \quad s \notin S^1_{\max} \cup S^0_{\max} \; etn = 0 \\ \sum\limits_{s' \in S, a \in Act} \mu_a(s, s') \times x_{s'}^{n-1} & si \quad s \notin S^1_{\max} \cup S^0_{\max} \end{cases}$$

À titre illustratif, nous solutionnons dans l'exemple ci-dessous le système d'équations de l'exemple 3.3.7 avec la méthode de la définition 3.3.3.

Exemple 3.3.8. Soient $n \in \mathbb{N}$ et $V^n = [P^n_{s_0}, P^n_{s_1}, P^n_{s_2}, P^n_{s_3}, P^n_{s_4}, P^n_{s_5}, P^n_{s_6}]$ le vecteur solution à l'itération n qui contient les probabilités d'atteindre s_6 à partir des différents états du DTMC illustré par la figure 3.2. Nous avons $S^0_{\max} = \emptyset$, $S^1_{\max} = \{s_6\}$ et on calcule V^n comme suit :
$V^0 = [0\ 0,\ 0,\ 0,\ 0,\ 0,\ 1]$

$V^1 = [0,\ 0,\ 0,\ 0,\ 0.3633 \times 1,\ 1 \times 1,\ 1] = [0,\ 0,\ 0,\ 0.3633,\ 1,\ 1]$

$V^2 = [0,\ 0,\ 0.625 \times 0.3633,\ 0.4167 \times 1 + 0.5833 \times 0.3633,\ 0.3633 \times 1,\ 1 \times 1,\ 1] = [0,\ 0.2271,\ 0.6286,\ 0.3633,\ 1,\ 1]$

$V^3 = [0.25 \times 0.625,\ 0.25 \times 0.6286 + 0.125 \times 0.2271,\ 0.625 \times 0.3633,\ 0.4167 \times 1 + 0.5833 \times 0.3633,\ 0.3633 \times 1,\ 1 \times 1,\ 1] = [0.1855,\ 0.2271,\ 0.6286,\ 0.3633,\ 1,\ 1]$

...

Notons qu'à la quatrième itération, P_{s_2} est restée inchangée tout comme P_{s_5} et P_{s_6} depuis la deuxième itération et P_{s_3} à depuis la troisième itération. Un point fixe est donc atteint pour ces probabilités. À l'opposé, vue la boucle sur l'état s_1, nous devons poursuivre les itérations pour atteindre un vecteur fixe. Nous ne pouvons pas montrer tous les calculus ici car il faut un nombre très élevé d'itérations. Mais vue la boucle sur s_1, nous extrapolons que pour un n très grand le vecteur solution sera $V = [1, 1, 0.2271, 0.6286, 0.3633, 1, 1]$.

À ce stade, nous avons décrit entièrement l'approche d'atteignabilité moyenne et au regard de tout ce qui précède, elle permet à partir du LMP L et de la propriété d'atteignabilité β de définir un DTMC moyen L_β avec des probabilités de transitions théoriquement exactes. Notre stratégie de vérification s'apparente à la méthode classique

d'évaluation de propriétés d'atteignabilité dans CTL [49] dans un DTMC et permet d'inférer si β est vraie dans L à partir du fait qu'elle est satisfaite par L_β.

À première vue, L_β est lié à L par β et on pourrait penser qu'il ne servirait que pour l'évaluation de celle-ci. En réalité non, car il est possible de caractériser un ensemble de propriétés pour lequel il peut être réutilisé à des fins de vérification. Nous apportons les preuves de cet avantage de notre approche à la prochaine section après avoir analysé et précisé l'exactitude de notre approche de point de vue implémentation. Nous montrons que notre approche a du sens lorsque le DTMC moyen est assorti d'erreurs et quantifions l'erreur près à laquelle L satisfait β si le DTMC moyen la satisfait. Nous récupérons par la suite les conclusions issues de nos analyses sur la réutilisabilité de L_β pour formaliser, à travers une brève étude comparative, les relations entre L_β construit théoriquement et L_β généré algorithmiquement avec des imprécisions numériques et L.

3.4 Exactitude de l'atteignabilité moyenne à l'implémentation et réutilisabilité de DTMC moyen

À la section 1.1, nous avons précisé les objectifs de notre mémoire. Si nous devons nous défaire de la préoccupation d'implémenter notre approche dans CISMO voire dans un autre vérificateur, les notions d'intégration numériques présentées à la section 3.2 ne seraient pas évoquées. La moyenne d'une fonctions de transition peut être calculée sans erreurs dans un cadre théorique et en aucun cas un DTMC moyen construit selon notre approche ne saurait être une approximation erronnée. Vu que nous faisons de l'évaluation de modèle et qu'elle se veut automatique, nous allons devoir situer les erreurs numériques dans notre approche pour mieux appréhender les limites de notre approche de vérification de point de vue implémentation. Tout au long de la section, nous considérons le LMP $L = (S, I, i, AP, Act, Label, \{\mu_a | a \in Act\})$ et la propriété d'atteignabilité $\beta = \mathbb{P}_q(\phi U \psi)$ dans $\mathtt{L}_1$. Pour faciliter nos explications théoriques, nous désignons par L_β le DTMC moyen équivalent théoriquement à L pour évaluer β et $\widehat{L_\beta}$ celui assorti éventuellement d'erreur parlant des probabilités de transition.

Dans CISMO il existe plusieurs sources potentielles d'erreurs qui influencent le résultat ultime d'une vérification selon notre approche. Après une brève élucidation des sources d'erreurs numériques relevant des travaux antérieurs sur CISMO, nous focalisons d'abord sur les sources d'erreurs inhérentes à notre approche, spécifiquement

celles provenant de l'évaluation d'intégrale. Nous les quantifions et prouvons qu'indépendamment de tout vérificateur, de L à $\widehat{L_\beta}$ il y a conservation d'erreurs numériques, c'est-à-dire la somme des erreurs commises en construisant $\widehat{L_\beta}$ est fonction du nombre de fonctions de transition dans L dont les moyennes ne peuvent être obtenues de façon exacte. Cette conservation numérique sera la preuve que L satisfait β à une certaine erreur près si et seulement si $\widehat{L_\beta}$ la satisfait. Ensuite, nous soulignons un avantage de notre approche en montrant que malgré que L_β est extrait de L à partir de β, il peut servir à évaluer des propriétés autres que β. Enfin, par souci de complétion des analyses d'erreurs numériques et de réutilisabilité de L_β, nous utilisons les conclusions de celles-ci avec des notions connues pour la comparaison de modèles probabilistes pour formuler les relations de ressemblance entre L_β, $\widehat{L_\beta}$ et L.

Dans les travaux antérieurs sur CISMO, d'une part, c'est l'évaluation des expressions arithmétiques qui engendre l'imprécision dans les réponses. On utilise la librairie JCM (Java Components for Mathematics) [59] pour effectuer les opérations arithmétiques et on manipule précisément la classe *ExpressionProgram* qui implémente des fonctions utilisant le type primitif double de la norme IEEE 754. Le lecteur peut se référer à [6, 60, 61] pour connaître les limites de cette norme. D'autre part, à la section 3.3.2 nous avons présenté la fonction *TrouverZeros*($h(x)$, X) qui utilise le partionnement en segment et la méthode de la sécante pour trouver les zéros de $h(x)$ sur X. Lorsqu'elle trouve un zéro, nous savons que la valeur est précise à une valeur de précision près. Par contre, il se peut que la fonction ne découvre pas de zéro pour deux raisons. La première est que le nombre d'itérations fait par la méthode de la sécante n'est pas suffisant et la deuxième est qu'aucun changement de signe n'est détecté sur le segment où est situé un zéro. Un exemple dans [6] présente ce cas de figure. Rappelons que CISMO propose deux modes de vérification : la vérification symbolique et la vérification basée sur l'utilisation de tables de hachage. Ses deux modes de vérification exploitent *TrouverZeros*($h(x)$, X) et sont victimes des imprécisions évoquées ci-dessus. L'approche d'atteignabilité moyenne fonctionne également avec ces deux modes de vérification et nous ne jugulons pas les erreurs numériques relevant des travaux antérieurs sur CISMO dans la mesure où nous réutilisons les algorithmes de Richard [6]. Les anciennes erreurs numériques viennent donc compléter celles émanantes de nos travaux et que nous clarifions par la suite.

Dans l'implémentation de notre approche, nous utilisons également la librairie JCM [59] pour calculer la moyenne d'une fonction de transition. La classe *RiemannSumRects* du package *edu.hws.jcm.functions* implémente différentes méthodes déterministes d'inté-

gration numérique dont celle que nous avons choisi : la méthode des trapèzes présentée (voir la section 3.2). Les différentes méthodes de cette classe fonctionnent avec un nombre de subdivisions d'intervalles qu'on peut faire varier de 1 à 5000.

Les premières erreurs numériques dans notre approche dépendent de ϕ et ψ. Si leur vérification (cas de `T` ou de propositions atomiques) ne nécessite aucune opération arithmétique, alors les imprécisions débutent avec le calcul des intégrales à condition de manipuler des fonctions de degré supérieur à 1. Les fonctions de transition dans L sont à priori des fonctions de répartition exprimées avec une loi uniforme sur des états et prennent la forme d'un polynôme dans le cadre de nos travaux. Pour avoir choisi la méthode des trapèzes comme technique d'intégration numérique, lors du calcul des moyennes des fonctions de transition de degré d'au plus 1, nos calculs d'intégrale sont exacts. Au-delà du degré 1, nous commettons pour le calcul d'intégrale d'une fonction de transition f définie sur $[a, b] \subset I^{\subseteq}(S)$ dans L une erreur de l'ordre de $\Theta(\frac{(b-a)^3}{N^2} f'')$ avec N le nombre de subdivisions de $[a, b]$. Pour être plus précis, il y a en réalité deux niveaux d'erreurs dans le calcul de la moyenne de f : nous avons les erreurs relatives à la méthode des trapèzes et celles liées aux calculs arithmétiques. Dans les démonstrations qui suivent, nous supposons les calculs arithmétiques exacts et entendons par erreur les erreurs inhérentes à notre méthode d'intégration. Les sources d'erreurs dans l'implémentation de notre approche étant fixées, nous allons prouver que l'erreur totale commise dans l'approximation d'un LMP par un DTMC selon notre approche est toujours bornée supérieurement.

Lors de la génération de $\widehat{L_\beta}$, nous supposons qu'il existe toujours au moins une fonction dont le calcul de la moyenne engendre la plus grande valeur d'erreurs numériques. Nous désignons par ε cette valeur dans nos démonstrations et observons qu'elle peut être nulle. Soit $K \in \mathbb{N}$ la cardinalité de l'ensemble des fonctions de transition dans L dont le calcul de leur moyenne engendre une erreur d'intégration. Si nous choisissons d'approximer directement L par un DTMC sans aucune considération de propriété d'atteignabilité, sans aucune modification préalable de son modèle et en supposant exacts les calculs arithmétiques, alors nous commettrons de toute évidence $K \times \varepsilon$ erreurs au total. L'exemple ci-dessous éclaircit ce raisonnement.

Exemple 3.4.1. Soit L le LMP illustré par la figure 3.1 et évaluons l'erreur totale commise en le transformant directement en DTMC avec le théorème de la moyenne. Pour y arriver, nous devons remplacer les fonctions de transitions non constantes par leur moyenne. Puisque deux (2) d'entre elles sont de degré 2 et ε désigne la valeur maximale possible de l'erreur commise à chaque calcul de moyenne, alors l'erreur total

voulue s'estime à $2 \times \varepsilon$.

Rappelons au lecteur que d'après la stratégie de construction de sous-LMP présentée plus haut, il se peut que $\widehat{L_\beta}$ soit de taille différente de L. Le sous-LMP Fig_3 de la figure 3.8 discuté à la section précédente illustre le cas. Pour mieux comprendre à quel point $\widehat{L_\beta}$ et L_β sont proches, il est important de s'interroger sur comment une erreur issue du calcul de la moyenne d'une fonction de transition $f(x)U(c,d)$ définie sur $[a,b] \in I$ avec $[c,d] \in I$ est reporté dans $\widehat{L_\beta}$ lors d'éventuels éclatements ou réductions d'intervalle dans I. Abordons alors l'analyse du report d'erreur en supposant que l'erreur commise en évaluant la moyenne de $f(x)U(c,d)$ sur $[a,b]$ vaut ε.

Supposons $i \in \mathbb{N}$ et $[c,d]$ éclaté en $n \in \mathbb{N}$ intervalles $[c_i,d_i] \subset [c,d]$ disjoints qui participent à la construction de $\widehat{L_\beta}$. Si $[a,b]$ participe également à la construction de $\widehat{L_\beta}$, alors le sous-LMP qui permettra de bâtir $\widehat{L_\beta}$ renfermera des transitions de $[a,b]$ vers chaque $[c_i,d_i]$ avec $f(x)U(c,d)$ comme fonction de de transition. Dans un schéma pareil, vu que nous intégrerons n fois $f(x)U(c,d)$ sur $[a,b]$ en calculant les probabilités moyennes de transitions, alors on penserait qu'à chaque intégration nous commettons une erreur ε_i valant ε. Dans les faits non car, la somme des erreurs commises sur l'ensemble des transitions de $[a,b]$ vers $[c_i,d_i]$ est bornée supérieurement par ε et on nous le justifions en deux temps.

Si P_i désigne la probabilité moyenne de transiter de $x \in [a,b]$ vers $[c_i,d_i]$, D_i la distance $d_i - c_i$ et ε_f l'erreur commise en évaluant $\int_a^b f(x)dx$, alors nous avons :

P_i

$\langle$ Application du théorème de la moyenne. $\rangle$

$= \frac{1}{b-a} \int_a^b (f(x)\frac{c_i-d_i}{c-d})dx$

$= \frac{d_i-c_i}{(b-a)(d-c)} \int_a^b f(x)dx$

$\langle$ Soit I la valeur numérique approchée de $\int_a^b f(x)$. $\rangle$

$= \frac{d_i-c_i}{(b-a)(d-c)}(I + \varepsilon_f)$

$= \frac{d_i-c_i}{(b-a)(d-c)}I + \frac{d_i-c_i}{(b-a)(d-c)}\varepsilon_f$

$\langle$ Supposons qu'on peut obtenir la valeur exacte de $\frac{d_i-c_i}{(b-a)(d-c)}$.$\rangle$

$= V_{approche} + \varepsilon_i$

où $V_{approche}$ est la valeur approchée de la probabilité moyenne de transition.

Avec la décomposition effectuée pour calculée P_i, chaque calcul de probabilité moyenne implique une erreur $\varepsilon_i = \frac{d_i - c_i}{(b-a)(d-c)}\varepsilon_f$. Puisque nous avons N intervalles issus de $[c, d]$, une estimation de l'erreur commise sur l'ensemble des transitions donne :

$$\sum_{i=1}^{n} \varepsilon_i$$
$$= \tfrac{d_1-c_1}{(b-a)(d-c)}\varepsilon_f + \tfrac{d_2-c_2}{(b-a)(d-c)}\varepsilon_f + ...+ \tfrac{d_n-c_n}{(b-a)(d-c)}\varepsilon_f$$
$$= \tfrac{1}{b-a}(\tfrac{D_1}{d-c} + \tfrac{D_2}{d-c} + \tfrac{D_3}{d-c} +....+\tfrac{D_n}{d-c})\varepsilon_f$$
$$\langle \text{ Or } \sum_{i=1}^{N} D_i \leq d - c \text{ et } \varepsilon = \tfrac{1}{b-a}\varepsilon_f. \rangle$$
$$\leq \varepsilon$$

Au regard de la conservation d'erreurs numériques lors de la construction d'un DTMC par notre approche, nous déduisons le théorème 3.4.1 précisant à quelles erreurs près un LMP satisfait une propriété d'atteignabilité à l'implémentation.

Théorème 3.4.1. Soient L un LMP, $K \in \mathbb{N}$ la cardinalité de l'ensemble de ses fonctions de transition dont le calcul de la moyenne engendre une erreur et $\widehat{L_\beta}$ un DTMC généré algorithmiquement selon notre approche à partir de L et de la propriété d'atteignabilité β. Si ε est la valeur maximale possible de l'erreur commise à chaque intégration numérique, alors les calculs arithmétiques étant exacts nous avons : $Y \models \phi \Leftrightarrow L \models_{K\epsilon} \phi$ où $\models_{K\varepsilon}$ désigne la satisfaction approchée à $K \times \varepsilon$ erreurs près.

Les limites de notre approche à l'implémentation étant fixées, abordons la question de réutilisabilité de DTMC dans notre approche.

Nous savons déjà qu'il est possible d'approximer un LMP par un DTMC pour vérifier les propriétés d'atteignabilité dans la logique $\mathtt{L}_1$ de CISMO. Étant donné que nous pouvons formuler une multitude de propriétés d'atteignabilité sur un LMP, alors il est évident que plusieurs DTMC peuvent lui correspondre moyennement et notre approche permet de n'en construire qu'un à la fois. Mais, est-ce possible d'utiliser un DTMC pour évaluer une propriété autre que celle ayant servi à sa génération ?

Tel qu'annoncé plus haut, nous pouvons caractériser quelques propriétés pour lesquelles un DTMC généré selon notre approche peut être réutilisé. Pour cela, considérons toujours le LMP L et la propriété d'atteignabilité $\beta = \mathbb{P}_{q'}(\phi U \psi)$. Si $\mathbb{P}_{q'}(\phi')$ est une propriété d'atteignabilité dans $\mathtt{L}_1$:

1. Si $\beta \Leftrightarrow \phi'$, alors L_β peut servir à évaluer $\mathbb{P}_{q'}(\phi')$ même si $q \neq q'$. Dans le cas où q et q' sont égaux, alors Y satisfait nécessairement $\mathbb{P}_{q'}(\phi')$.

2. Si $\beta \Rightarrow \phi'$, il n'est pas garanti que L_β puisse servir à verifier ϕ'. Afin d'être convaincu de ce fait, nous l'illustrons à travers les DTMC et le LMP L qu'illustre la figure 3.12. En posant $\phi = (a \wedge b)\,Uc$ et $\phi^{'} = bUc$, nous notons bien que $\phi \Rightarrow \phi'$ et pourtant $L_{(a\wedge b)Uc}$ ne peut être utilisé pour évaluer ϕ'. Par contre, si $\phi^{'} = aUc$ on a $L_{(a\wedge b)Uc}$ et L_{aUc} identiques.

Avant de clore l'analyse sur la réutilisabilité de DTMC, nous allons évoquer une remarque sur l'étiquetage des états d'un DTMC construit selon notre approche. En considérant le DTMC L_{bUc} de la figure 3.12 à la page 92, l'état s_4 représente l'intervalle $[0, 2]$ étiqueté par b dans L et selon le LMP de la même figure, l'intervalle $[0, 1]$ est étiqueté par a. Du fait que $[0, 1] \subset [0, 2]$, peut-on dire que s_4 satisfait $a \wedge b$ sachant que les états dans $]1, 2]$ satisfont que b ? Peut-on utiliser L_{bUc} pour évaluer la probabilité de aUc ? Les réponses à ces questions sont relatives et peuvent dépendre des objectifs de vérification. Pour cela, dans de circonstance pareille dans CISMO, nous précisons à l'utilisateur que a étiquette s_4 et le détail sur les intervalles associés à a et à b lui est aussi fourni. Ainsi,la décision d'évaluer ou pas aUc avec L_{bUc} au lieu de L_{aUc} lui revient.

Dans notre analyse sur la réutilisabilité de DTMC, nous avons focalisé sur les spécifications d'atteignabilité. Mais un DTMC construit selon notre approche peut servir à vérifier d'autres propriétés que celles d'atteignabilité. En témoigne le DTMC L_{aUc} de la figure 3.12 qui satisfait la proposition atomique a. Nous remarquons cette possibilité pour attirer l'attention du lecteur sur le fait que la réutilisabilisation de DTMC dans notre approche peut s'étendre au-delà de celle que nous avons justifiée dans ce mémoire.

En conclusions à tout ce qui précède, nous retenons que $\widehat{L_\beta}$ ou L_β peuvent servir à évaluer plusieurs propriétés et que les imprécisions numériques dans $\widehat{L_\beta}$ qui nuisent aux résultats ultimes de vérifications sont bornées supérieurement. Nous allons maintenant utiliser ces certitudes pour présenter une comparaison quantitative entre les modèles $\widehat{L_\beta}$, L_β et L. Dans la théorie classique d'analyse et de vérification de systèmes, les relations de simulation [62, 63, 64, 65] et de bisimulation [66, 64, 67] sont utilisées pour les comparaisons de systèmes. Dans le cadre de notre mémoire, nous nous intéressons à la bisimulation probabiliste.

Informellement, deux systèmes probabilistes ou non sont bisimulaires s'ils peuvent produire les mêmes exécutions. La recherche de bisimularité entre deux systèmes vise donc à

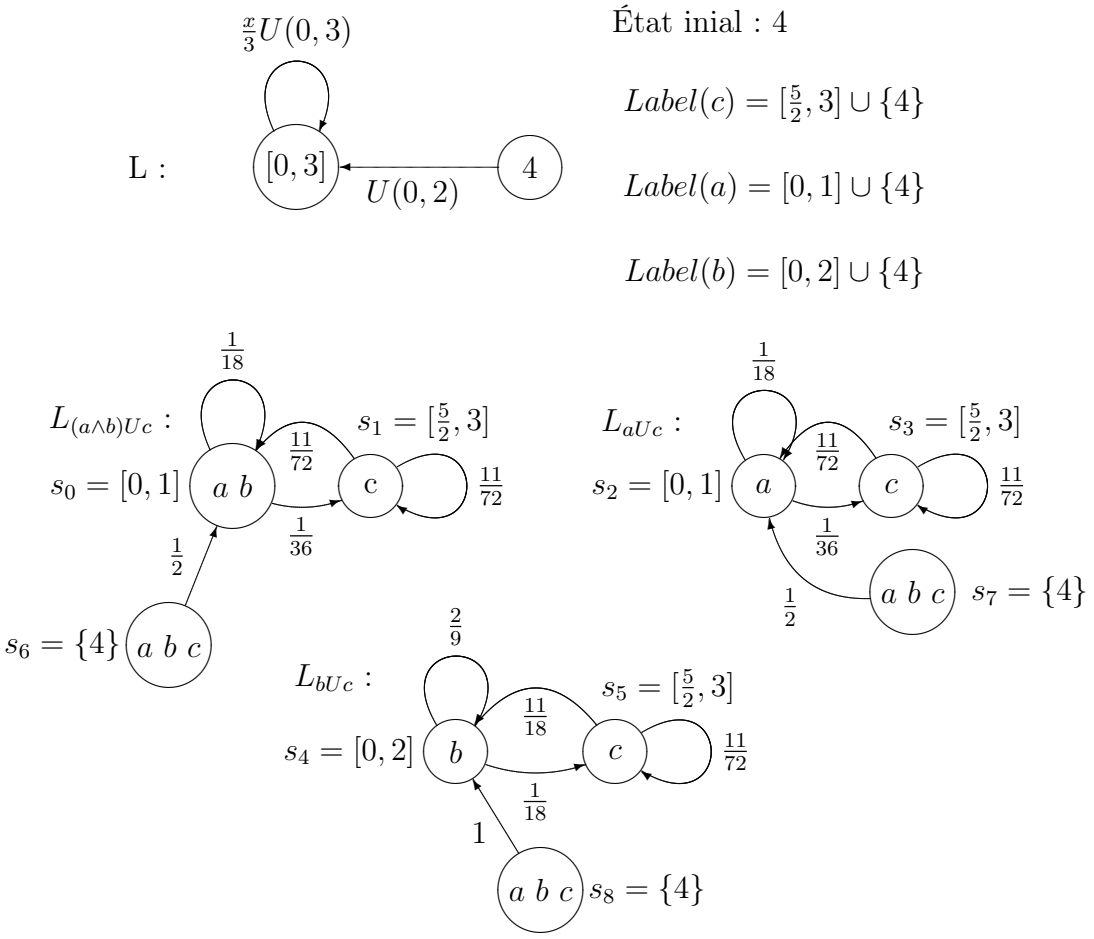


FIGURE 3.12 – Exemple de LMP transformé en DTMC selon l'approche d'atteignabilité moyenne.

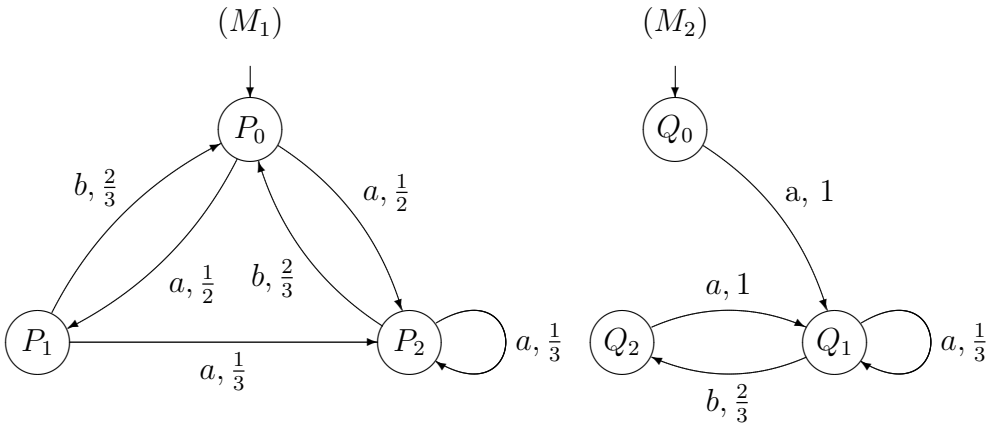
savoir si l'un peut imiter l'autre en effectuant les mêmes transitions avec les mêmes probabilités que son correspondant. Pour définir la bisimulation probabiliste, on table sur la notion d'équivalence de distributions de probabilités que nous définissons ci-dessous.

Définition 3.4.1. Soient R une relation d'équivalence sur un ensemble d'états S et μ_1 et μ_2 deux distributions de probabilités sur S. On dit que μ_1 et μ_2 sont équivalentes et on écrit $\mu_1 \equiv_R \mu_2$ si pour toute classe d'équivalence C de S on a $\mu_1(C) = \mu_2(C)$.

La notion de bisimulation se voit sous deux angles : la bisimulation forte qui fait abstraction des actions silencieuses qu'effectue un système et la bisimulation faible qui fait état de celles-ci. La définition formelle ci-dessous de la bisimulation est celle d'une bisimulation probabiliste forte sur deux DTMC et la figure 3.13 en présente un exemple.

Définition 3.4.2. Soient $M_1 = (S_1, i, \mu_1, Act, AP, L)$ et $M_2 = (S_2, i, \mu_2, Act, AP, L)$ deux MDP. Une relation d'équivalence R sur $S_1 \cup S_2$ (union disjointe) est une bisimulation forte si pour tout $q \in S_1$ et $r \in S_2$ tel que $q\ R\ r$ on a :

$$L(q) = L(r) \text{ et } \forall a \in Act,\ \mu_1(a, .) \equiv_R \mu_2(a, .)$$



$R = \{(P_0, Q_0), (P_0, Q_2), (P_1, Q_1), (P_2, Q_1)\}$ est une bisimulation forte.

FIGURE 3.13 – Exemple de DTMC bisimulaires.

La bisimulation probabiliste telle que définie ci-dessus exige une égalité des probabilités de transition dans les deux systèmes. Dans certains cas, les systèmes peuvent disposer des mêmes transitions avec des probabilités différentes, et donc être différents tout en ayant des comportements très proches. C'est le cas de L_β et $\widehat{L_\beta}$ dans l'approche d'atteignabilité moyenne. Dans ce cas de figure, quand on est en mesure de relativiser les différences entre les probabilités de transition, on relaxe l'exigence d'égalité de probabilités de transition de la definition 3.4.2 et on parle *ϵ-bisimulation* [64]. C'est l'idée de tolérance d'erreurs sur les probabilités de transition qui nous permet de parler de *bisimulation* entre L_β et $\widehat{L_\beta}$ et de la caractériser par les erreurs numériques dont le théorème 3.4.1 garantit une borne supérieure. D'après les théories de *ϵ-bisimulation*, le théorème 3.4.2 est donc vrai malgré que nous ne nous y accordons pas une démonstration particulière dans ce mémoire. La comparaison quantitative de systèmes probabilistes a fait l'objet de plusieurs travaux et [68, 64], par exemple, apportent des preuves suffisantes qui éclaircissent sur le théorème 3.4.2.

Théorème 3.4.2. Soient L un LMP, $K \in \mathbb{N}$ la cardinalité de l'ensemble de ses fonctions de transition dont le calcul de la moyenne engendre une erreur et β une propriété d'atteignabilité. Si ε est la valeur maximale possible de l'erreur commise à chaque intégration numérique alors, L_β est $K\varepsilon$-bisimulaire à $\widehat{L_\beta}$ et on écrit $L_\beta \sim_{K\varepsilon} \widehat{L_\beta}$. Bien entendu, le cas $K\varepsilon = 0$ ramène aux relations de bisimulation classique.

Au regard du théorème 3.4.2 qui tolère l'inexactitude des probabilités de transition dans la comparaison de systèmes, deux systèmes *ϵ-bisimulaires* produisent des exécutions comparables. Vu que lorsqu'on évalue une propriété sur un système on s'intéresse aux exécutions qui la respectent, on peut choisir bâtir une comparaison de systèmes sur la base d'ensembles F de propriétés que respectent leurs exécutions. On parle alors de *F-bisimulation* [67] et elle se note $\approx_F$. Du moment où nous avons prouvé dans notre contexte que L_β peut servir à évaluer et satisfaire par ricochet plusieurs propriétés de L, nous retenons $L_\beta \approx_F L$. Mais peut-on conclure également $L \approx_F \widehat{L_\beta}$ ou $L_\beta \approx_F \widehat{L_\beta}$?

En fait, il n'est pas précis de conclure $L_\beta \approx_F \widehat{L_\beta}$ ou $L \approx_F \widehat{L_\beta}$ car nous savons que L_β et $\widehat{L_\beta}$ sont *ϵ-bisimulaires*. Pour cela, nous introduisons la notation $\approx_\epsilon^F$ pour signifier une *F- bisimulation* à ϵ erreur près et les différentes idées développées dans cette section se résument par les propositions équivalentes ci-dessous :

$$\begin{gathered} L \approx_F L_\beta \Leftrightarrow L \approx_{K\varepsilon}^F \widehat{L_\beta} \Leftrightarrow \widehat{L_\beta} \approx_{K\varepsilon}^F \widehat{L_\beta} \\ \Leftrightarrow \\ L_\beta \sim_{K\varepsilon} \widehat{L_\beta} \\ \Leftrightarrow \\ L \models \beta \Leftrightarrow L_\beta \models \beta \Leftrightarrow \widehat{L_\beta} \models_{K\varepsilon} \beta. \end{gathered}$$

De tout ce qui précède, retenons que le domaine des systèmes probabilistes repose fondamentalement sur des représentations approchées de systèmes réels : les valeurs des probabilités de transition d'un système, souvent inférées par expérimentation, ne peuvent qu'être approximées. Nous nous sommes reposés sur l'intuition qu'il était nécessaire de posséder des outils de comparaison quantitative entre systèmes prenant en compte cet aspect, pour établir une relation de bisimulation entre un LMP et un DTMC qui l'approxime selon l'approche d'atteignabilité moyenne. Au regard des différentes analyses et conclusions présentées, l'approche d'atteignabilité moyenne vient offrir incontestablement un moyen efficace pour comparer des systèmes probabilistes infinis via leur équivalent en système fini.

Chapitre 4

Implémentation

Nous abordons les détails de l'implémentation de l'approche d'atteignabilité moyenne dans CISMO et articulons le chapitre autour de trois points :

1. nous précisons les paramètres fixes dans notre implémentation et présentons sommairement les classes introduites dans l'architecture du programme de CISMO ;
2. nous présentons les formats à respecter pour formuler les propriétés d'atteignabilité et décrire les LMP ;
3. nous présentons un exemple d'évaluation de propriété d'atteignabilité résolu par CISMO et nous en profiterons pour spécifier les formats possibles de DTMC qu'il génère.

Dans notre implémentation, nous avons fortement privilégié la réutilisation du code existant pour préserver les deux modes de vérification qu'offre CISMO : la vérification symbolique qui exploite les MTBDD et celle qui utilise les tables de hachage.

Le besoin d'intégration numérique lors de la construction de DTMC nous contraint à fixer des paramètres dans notre code. Rappelons que dans notre approche, nous ne nous sommes pas souciés de l'optimisation des erreurs numériques. Pour cela, nous n'avons pas jugé utile de proposer dans les options (au niveau interface) de préférences de CISMO une gamme variée de méthodes d'intégration qu'offre la classe *RiemannSumRects* du package *edu.hws.jcm.functions* du projet JCM [59]. Nous avons donc fixé la méthode des trapèzes comme technique d'intégration numérique. Aussi, la classe permet de varier de 1 à 5000 le nombre de subdivisions d'intervalles sur lequel on évalue une intégrale mais nous l'avons fixé à 5000. Pour ce qui est d'un nombre décimal, nous limitons le nombre de chiffres après une virgule à 7 dans nos calculs.

Dans l'architecture du programme de CISMO, nous avons introduit trois nouvelles classes au-delà d'avoir retouché principalement :

- Les classes *Interval*, *IntervalList*, *ProbSystemHash*, *Transition* et *ProbSystemMtbdd* du package Data.
- La classe *Engin* du package Engin.
- Les classes *MainFrame*, *FormulaDialog* et *EnginTraceDialog* du package Ui.

Dans ces différentes classes, nous avons essentiellement réajusté différentes fonctions pour supporter le traitement du format que nous adoptons pour traduire une loi uniforme dans la description d'un LMP.

Dans la version de CISMO que nous rendons disponible, les classes *ProbSystemMDP* et *StandardUniform* sont celles introduites dans le package Data. La première dispose des fonctions majeures pour générer un DTMC selon notre approche. Elle implémente l'algorithme 3.10 auquel nous assortissons le traitement des propositions atomiques qui étiquettent les états du DTMC et les actions qu'effectue celui-ci. La classe *StandardUniform* quant à elle, implémente la loi uniforme telle que décrite dans ce mémoire et dispose des fonctions utiles à l'évaluation d'une uniforme dans une fonction de transition. Dans une fonction de transition, nous traduisons la loi uniforme sur un ensemble d'états cibles par la lettre U qui a valeur de variable. Au-delà d'avoir permis l'utilisation de loi uniforme dans les fonctions de transition, nous avons préservé la façon d'exprimer les fonctions de transitions dans les versions antérieures de CISMO.

La classe *LogicalUNTIL* ajoutée au package Logic implémente les fonctions utiles à la reconnaissance d'une propriété d'atteignabilité par CISMO. Pour une propriété $\mathbb{P}_q(\phi U \psi)$ (voir la section 3.1) à évaluer, vu que nous ne calculons pas dans CISMO la probabilité avec laquelle elle peut être satisfaite, nous utilisons $\phi U \psi$ pour générer le DTMC équivalent. Sur ce, CISMO accepte la propriétés sous le format (ϕ)UNTIL(ψ). Remarquons que ϕ et ψ sont entourées de parenthèses. Pour ce qui est des formules de type $\mathbb{P}_q(\langle a \rangle \phi)$ (voir la définition 2.3.1), c'est le format $< a > [q]\phi$ que nous conservons.

À présent, abordons la description du format sous lequel un LMP est donné en paramètre à CISMO et les formats de DTMC qu'il génère. La figure 4.2 montre le contenu du fichier donné à CISMO pour charger le LMP qu'illustre la figure 2.5. Par contre, la figure 4.3 montre dans CISMO le LMP qu'illustre la figure 4.1 et sur lequel nous voulons vérifier (b)U(c). En appuyant sur le bouton "get finite system", CISMO propose deux modes d'affichage du DTMC moyen qu'il génère.

Dans le premier mode, mode d'affichage par défaut, comme le montre la figure 4.4, les intervalles ayant servi à construire le DTMC moyen sont considérés comme les états de celui-ci. Dans le second mode, nous remplaçons chaque intervalle ayant servi à bâtir le DTMC moyen par un entier positif tel que le montre la figure 4.5. C'est la représentation classique d'un système fini et remarquons que CISMO informe sur une proposition atomique qui n'étiquette pas entièrement l'état auquel elle est associée.

En conclusion, retenons un vérificateur est un programme complexe pouvant rassembler plusieurs fonctionnalités. Dans ce chapitre, nous avons décrit l'implémentation de l'approche d'atteignabilité moyenne dans CISMO. Dans notre implémentation, nous avons choisi d'utiliser le projet JCM [59] non seulement par souci de réutilisation de code mais aussi parce qu'il dispose de la classe *RiemannSumRects* qui implémente la méthode des trapèzes. Pour finir, nous avons montré des exemples de propriétés d'atteignabilité évaluées sur des LMP et les résultats obtenus sont conformes à nos attentes.

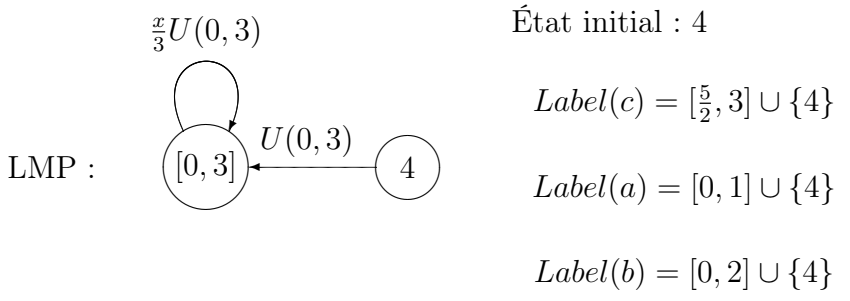


FIGURE 4.1 – Exemple de LMP avec distributions de probabilités uniformes.

```
states: [0,3]U{4}U{5};

init:  1;

NbAction : 2;

X : (0,1) , (1,2] ,(2,3], {0}, {1};
Y : (0,1) , {0}, {1}, (1,2], (2,3],{4}, {5};

label
        panne: {4}U{5};
endlabel

transition
        (0,1)  -[a]->   (1/4)*U   :   (0,1);
        (0,1)  -[a]->   x/4       :   {0};
        (0,1)  -[a]->   (1-x)/4   :   {1};
        (0,1)  -[a]->   U/4       :   (1,2];
        (0,1)  -[a]->   x*U/4     :   (2,3];
        {0}    -[a]->    1/4      :   (0,1);
        {1}    -[a]->    U/4      :   (0,1);
        {1}    -[a]->    U/4      :   (2,3];
        {1}    -[a]->    U/4      :   (1,2];
        (1,2]  -[a]->    1        :   {4};
        (2,3]  -[b]->    1        :   {5};
endtransition
```

FIGURE 4.2 – Exemple de fichier décrivant le LMP de la figure 2.5 à passer à CISMO.

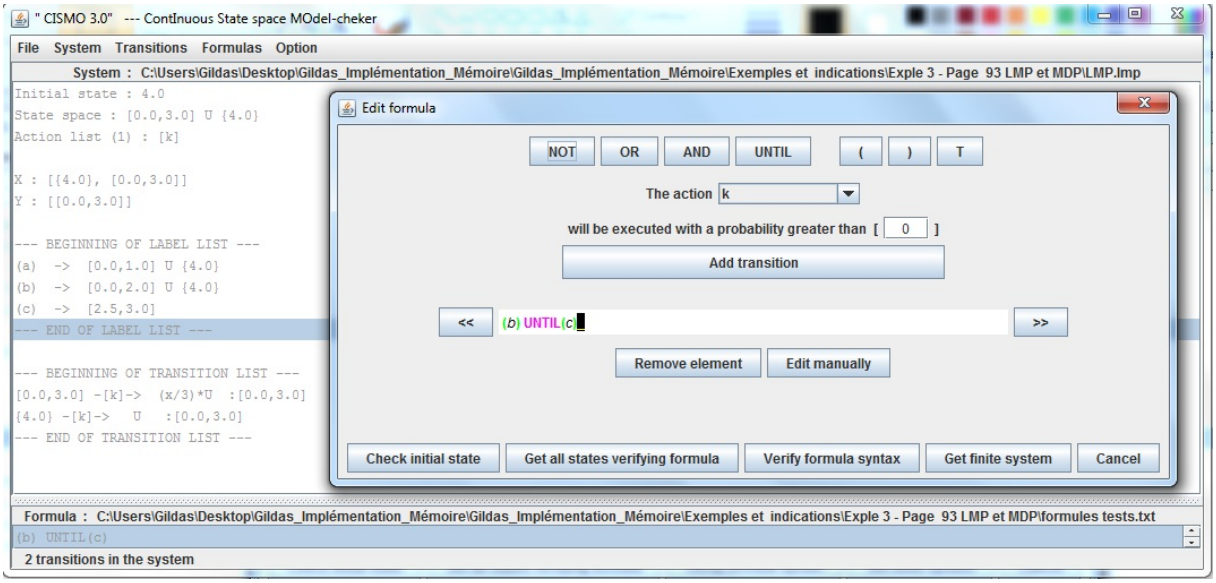


FIGURE 4.3 – LMP illustré par la figure 4.1 chargé dans CISMO.

```
Formula :  ( (b) ) UNTIL ( (c) )
State space :  [0.0,2.0] U [2.5,3.0] U {4.0}

Initial state :  4.0

Final state :  [2.5,3.0]

---   BEGINNING OF LABEL LIST   ---

(a)  ->  [0.0,1.0] U {4.0}

(c)  ->  [2.5,3.0]

(b)  ->  [0.0,2.0] U {4.0}

---   BEGINNING OF TRANSITION LIST   ---

{4.0}    -  [k]  ->    0.1666667   :   [2.5,3.0]

{4.0}    -  [k]  ->    0.6666667   :   [0.0,2.0]

[2.5,3.0]    -  [k]  ->    0.1527778   :   [2.5,3.0]

[2.5,3.0]    -  [k]  ->    0.6111111   :   [0.0,2.0]

[0.0,2.0]    -  [k]  ->    0.0555556   :   [2.5,3.0]

[0.0,2.0]    -  [k]  ->    0.2222222   :   [0.0,2.0]
```

FIGURE 4.4 – Forme détaillée de DTMC moyen généré à partir du LMP illustré par la figure 4.1 et de la formule $b\mathrm{U}c$.

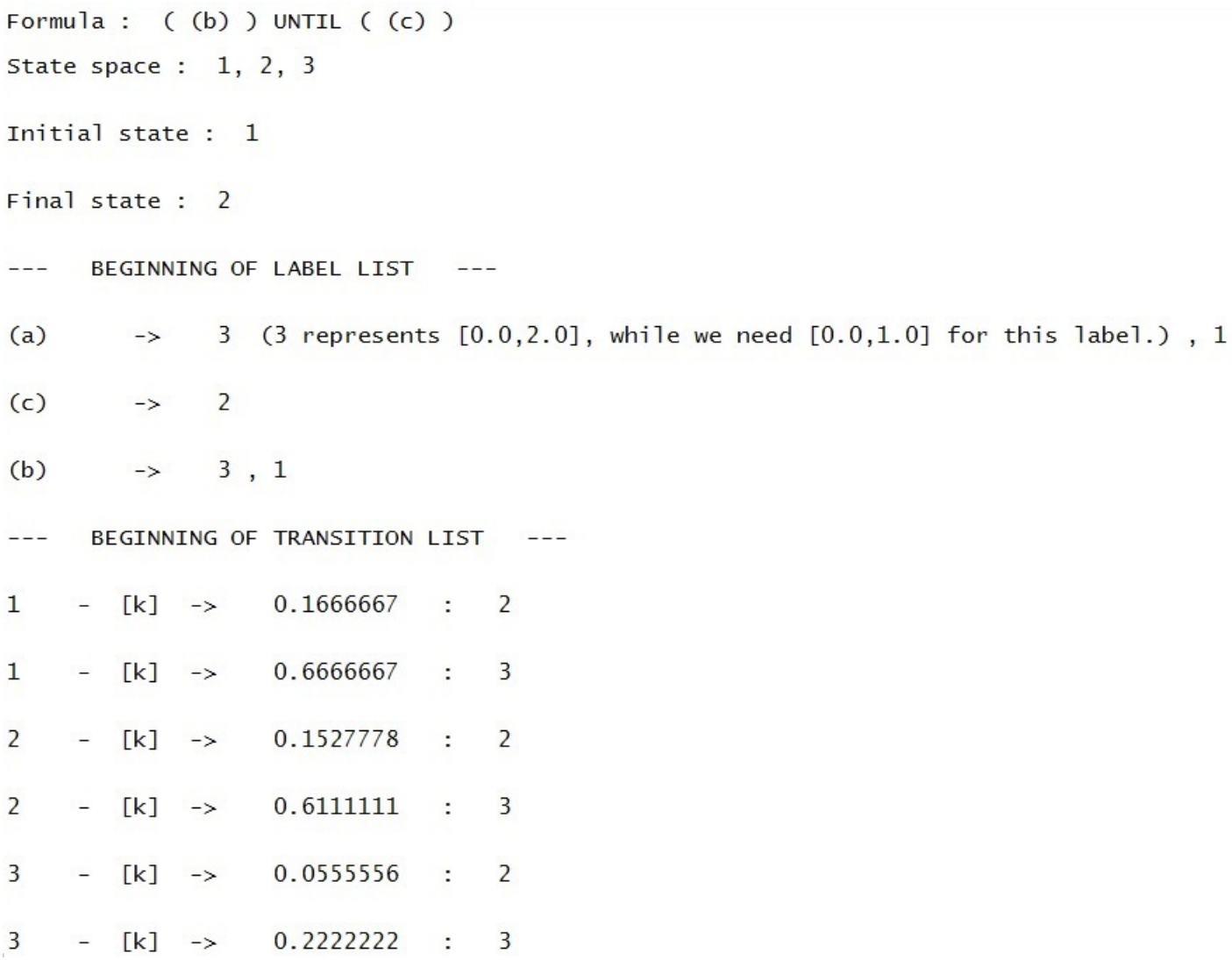


FIGURE 4.5 – Forme classique du DTMC moyen généré à partir du LMP illustré par la figure 4.1 et de la formule $\mathtt{T}Uc$

Chapitre 5

Conclusion

Quelques échecs retentissants comme le bogue du Pentium de Intel en 1994, la destruction du premier exemplaire de la fusée Ariane 5 en 1996 etc. ont achevé de convaincre de l'impérieuse nécessité de vérifier les logiciels ou systèmes informatiques avant leur mise en service. Il existe plusieurs méthodes pour effectuer de telles vérifications, les principales étant le *test*, la *démonstration de théorèmes* et l'évaluation de modèle. Dans ce mémoire, nous nous sommes intéressés à l'évaluation de modèle.

Pour vérifier automatiquement un système par évaluation de modèle, il est nécessaire d'en construire d'abord une modélisation formelle sous la forme d'un système de transitions appelé modèle. Pour cela, on utilise un langage de spécification de systèmes. Il faut ensuite énoncer formellement les propriétés à vérifier à l'aide d'un langage de spécification de propriétés, par exemple la logique temporelle. Enfin, il faut disposer d'algorithmes capables de dire si le système vérifie ou non les propriétés énoncées. Les algorithmes sont incarnés dans un vérificateur : un outil informatique pour l'évaluation de modèle. Par le biais d'un modèle, il est donc possible de s'assurer qu'un système respecte une propriété.

Il existe plusieurs façons de modéliser un système et plusieurs types de modèles existent dans la littérature. Dans ce mémoire, nous avons fait de la vérification probabiliste, c'est-à-dire le modèle des systèmes traités intègre les probabilités pour appréhender les incertitudes de la modélisation. Le modèle probabiliste le plus usuel et célèbre est la chaîne de Markov. La plupart du temps, on l'utilise pour modéliser les systèmes dont l'évolution dépend toujours et uniquement de l'état courant. Dans ce mémoire, nous avons hissé au coeur des recherches menées une variante de la chaîne de Markov : la chaîne de Markov étiquetée (en anglais, LMP pour *Labelled Markov process*). Elle

possède généralement un espace d'états infini et sert à modéliser les systèmes qui ont un comportement aléatoire ou qui approximent des systèmes complexes. Elle permet aussi l'analyse de la composition de plusieurs systèmes puisqu'on étudie leurs interactions avec leurs environnements.

Le but d'une vérification est certes de se convaincre qu'un système fonctionne sans failles, mais généralement, on s'assure seulement de déterminer avec certitude si un certain type d'erreur peut ou non survenir. Les propriétés les plus vérifiées sont celles de sûreté, d'atteignabilité et de vivacité.

Dans ce mémoire, nous avons focalisé sur la vérification de propriétés d'atteignabilité dans les LMP. Elles sont fort utiles pour garantir la sûreté d'un système, car elles expriment le fait que des états recherchés dans un système peuvent être atteints. Jusque là, le vérificateur CISMO dédié aux modèles à espace d'états infini, plus spécifiquement aux LMP, ne gérait pas les propriétés d'atteignabilité et aucun autre outil ne peut vérifier les LMP. Pour cela, nous avons choisi d'améliorer CISMO dans nos travaux.

Pour ce fait, nous avons élargi sa logique $\mathtt{L}_0$ pour qu'elle permette d'énoncer des propriétés d'atteignabilité sous la forme $\mathbb{P}_q(\phi U \psi)$ avec ϕ et ψ dans $\mathtt{L}_0$ et $q \in [0, 1]$. $\phi U \psi$ admet la sémantique de la logique temporelle linéaire et q est la probabilité avec laquelle on souhaite la satisfaire dans un LMP. Les recherches menées nous ont permis de prouver qu'il est possible d'évaluer une propriété d'atteignabilité dans un LMP par le biais d'une autre variante de chaîne de Markov qui l'approxime : la chaîne de Markov à temps discret (en anglais, DTMC pour *Discrete time Markov chain*).

Pour réaliser cet exploit, nous avons exploré deux stratégies de vérification après s'être inspiré de deux approches de vérifications publiées en 2010 : la première encode un système infini qui n'est pas un LMP sous la forme d'un système probabiliste symbolique et y extrait un système fini en vue d'évaluer une probabilité d'atteignabilité dans celui-ci. La seconde est une méthode multi-objectifs que nous avons qualifiée de vérification par hypothèse. Elle vérifie à la fois deux propriétés de sûreté assorties de probabilités dont l'une est une hypothèse pour l'autre. Malgré que les deux approches présentent des résultats expérimentaux performants, nous sommes restés favorables à la première stratégie, car la seconde implique d'implémenter plusieurs algorithmes connus aux systèmes finis pour les LMP. Ce qui est requiert un travail énorme.

Au lieu d'utiliser les systèmes probabilistes symboliques dans notre approche, nous avons utilisé le *théorème de la moyenne*, et notre stratégie de vérification est la sui-

vante : à partir de la structure de système fini qu'a un LMP L de CISMO et d'une propriété $\mathbb{P}_q(\phi U \psi)$, d'abord, nous extrayons de L un sous-LMP disposant uniquement des états satisfaisants ϕ ou ψ que nous transformons en un DTMC moyen $L_{\phi U \psi}$ en remplaçant ses fonctions de transitions non constantes par leur moyenne. Ensuite, nous passons $L_{\phi U \psi}$ à un vérificateur de systèmes finis qui évalue $\phi U \psi$ dans $L_{\phi U \psi}$. Si $L_{\phi U \psi}$ satisfait la propriété, alors nous inférons que L la satisfait aussi. Nous avons implémenté dans CISMO l'algorithme qui génère $L_{\phi U \psi}$ et nous réutilisons fortement d'anciens algorithmes du vérificateur.

Pour convaincre le lecteur de la véracité de notre approche, nous avons prouvé son exactitude. Tout comme dans l'analyse de tout système avec valeurs numériques, notre approche donne théoriquement des résultats exacts. Mais à l'implémentation, lorsque les calculs numériques de moyenne de fonctions de transitions engendrent des erreurs, le DTMC qu'on obtient est une approximation assortie d'erreurs et c'est normal. Ceci fait que le résultat ultime de la vérification peut être faux. Toutefois, les imprécisions numériques n'invalident pas la véracité de notre approche. Nous avons prouvé qu'elle à du sens à l'implémentation en démontrant, d'une part, que les erreurs numériques sont toujours bornées supérieurement et en établissant, d'autre part, des relations de bisimulation entre le DTMC moyen construit théoriquement et le DTMC moyen généré algorithmiquement avec des erreurs.

Au-delà de l'amélioration de CISMO, c'est la vérification probabiliste qui connaît aussi une évolution. Les bénéfices de notre approche sont de trois ordres. D'abord, nous profitons des acquis connus aux systèmes finis en faisant la vérification ultime dans un vérificateur dédié aux systèmes finis. Contrairement à la première approche explorée, notre approche est moins compliquée car elle utilise un fondement mathématique simple pour transformer un problème difficile en un problème simple à résoudre et notre stratégie de vérification s'apparente à la méthode classique de vérification de propriétés d'atteignabilité dans LTL sur un système fini. Ensuite, elle est aussi utile que les approches qui nous ont inspiré car, d'une part, on peut l'intégrer aux vérificateurs dédiés aux systèmes finis pour élargir le champ des systèmes qu'ils traitent. D'autre part, elle participe à éviter un des principaux problèmes en vérification : l'explosion combinatoire en terme de nombre d'états ou de chemins. Elle peut entraîner un manque de mémoire lors de la vérification, surtout de systèmes infinis comme les nôtres. Enfin, parlant de distribution de probabilités sur les états, notre approche est plus avancée par rapport la première approche exposée. Elle s'applique tant dans un contexte de distributions de probabilités continues que discrètes.

Nous avons mené nos recherches avec une distribution de probabilité continue, notamment la loi uniforme qui est simple, dans le but de présenter notre approche dans un contexte simple. Une prochaine étude pourrait consister à la généraliser en l'étendant à d'autres lois continues et pourquoi pas à d'autres types de formules puisque nous n'avons étudié dans ce mémoire qu'une famille de propriétés d'atteignabilité. Une autre piste de recherche intéressante serait de voir comment prendre un DTMC généré selon notre approche et l'augmenter ou le diminuer de quelques états pour satisfaire d'autres formules. Donnons un exemple pour étayer cette idée.

Considérons le LMP illustré par la figure 3.12 et supposons que nous voulons vérifier $a\,U\,c$ et $b\,U\,c$ dans L. Nous pouvons construire d'abord L_{bUc} et procéder à la vérification de $b\,U\,c$. Ensuite, pour construire L_{aUc}, nous pouvons utiliser L_{bUc} et les états issus de l'application de la première étape de notre approche à L avec $a\,U\,c$. Dans le cas d'espèce, nous notons que $[\frac{5}{2}, 3]$ $(Sat(L,\ c))$ et $[0, 1]$ $((Sat(L,\ a)))$ figurent déjà dans L_{bUc} seulement, $[0, 1] \subset [0, 2]$. Pour avoir donc L_{aUc}, il nous faut juste restreindre $[0, 2]$ (s_4) à $[0, 1]$ et mettre à jour les probabilités des transitions sortantes et entrantes dans s_4. Cette façon de bâtir L_{aUc} nous évite d'exécuter au complet l'algorithme 3.10 de la page 77 et donc nous gagnons du temps.

À la lumière des travaux futurs possibles évoqués, avec leur réalisation, nous serons plus en mesure de promouvoir et pousser plus loin la vérification de modèles probabilistes à espace d'états continus et par le fait même CISMO.

Bibliographie

[1] Institut für Informatik Prof. Thomas Huckle. Collection of software bugs. http ://www5.in.tum.de/ huckle/bugse.html, la dernière mis à jour date du 7 Novembre 2011. Site consulté le 01 janvier 2014.

[2] Vibhav Gogate and Pedro Domingos. Probabilistic theorem proving. In *Proceedings of the Twenty-Seventh Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-11)*, pages 256–265, Corvallis, Oregon, 2011. AUAI Press.

[3] Olaf Beyersdorff. The deduction theorem for strong propositional proof systems. In *Proceedings of the 27th international conference on Foundations of software Technology and Theoretical Computer Sciences*, FSTTCS'07, pages 241–252, Berlin, Heidelberg, 2007. Springer-Verlag.

[4] Philippe Schnoebelen and Pierre McKenzie. *Systems and software verification : model-checking techniques and tools*. Springer, Berlin, New York, Paris, 2001. Traduit de : Vérification de logiciels : techniques et outils du model-checking/coordonné par Philippe Schnoebelen.

[5] Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, 24(3) :293–318, Septembre 1992.

[6] Nicolard Richard. La vérification formelle de systèmes probabilistes continus. Faculté des études supérieures de l'Université Laval, 2003.

[7] Simon Paquette. Évaluation symbolique de systèmes probabilistes à espace d'états continus. Faculté des études supérieures de l'Université Laval, 2005.

[8] Marta Kwiatkowska, Gethin Norman, David Parker, and Qu Hongyang. Assume-Guarantee Verification for Probabilistic Systems. In *TACAS 2010 : 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Paphos, Chypre, 2010. Springer.

[9] Marta Kwiatkowska, Gethin Norman, and Jeremy Sproston. Symbolic computation of maximal probabilistic reachability. In K. Larsen and M. Nielsen, editors, *Proc. 13th International Conference on Concurrency Theory (CONCUR'01)*, volume 2154 of *LNCS*, pages 169–183. Springer, 2001.

[10] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0 : Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[11] Emmanuelle Anceaume, François Castella, Romaric Ludinard, and Bruno Sericola. Markov Chains Competing for Transitions : Application to Large-Scale Distributed Systems. Juin 2011.

[12] David Anthony Parker. *Implementation of symbolic model checking for probabilistic systems*. PhD thesis, University of Birmingham, 2003.

[13] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[14] Pierre Bremaud. *Markov Chains : Gibbs Fields, Monte Carlo Simulation, and Queues*. Springer-Verlag New York Inc., corrected edition, Février 2001.

[15] Marc Troyanov. Mesures et intégration. fr.scribd.com/doc/73764590/Me-Sure-Integration, Consulté le 30 avril 2013.

[16] Erhan Çinlar. *Introduction to Stochastic Processes*. Prentice-Hall, Englewood Cliffs, N. J., 1975.

[17] Stewart. N. Ethier and Thomas G. Kurtz. *Markov Processes : Characterization and Convergence*. Wiley, New York, NY, 1986.

[18] Daniel J. Lehmann and Saharon Shelah. Reasoning with time and chance (extended abstract). In Josep Díaz, editor, *Automata, Languages and Programming, 10th Colloquium, Barcelona, Spain, Juillet 18-22, 1983, Proceedings*, volume 154 of *Lecture Notes in Computer Science*, pages 445–457. Springer, 1983.

[19] Roberto Segala. *Modeling and verification of randomized distributed real-time systems*. PhD thesis, Cambridge, MA, USA, 1995.

[20] Henk C. Tijms. *A First Course in Stochastic Models*. Wiley, 2003.

[21] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. *Model-checking algorithms for continuoustime Markov chains.* IEEE Trans, Software Eng, 2003.

[22] Kumud Sanwal Vigyan Singhal Adnan, Aziz and Robert Brayton Brayton. *Model-Checking continous-time Markov chains.* ACM Trans, Comput. Log., 2000.

[23] Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In Mike Paterson, editor, *ICALP*, volume 443, pages 322–335. Springer, 1990.

[24] Rajeev Alur and David Dill. A theory of timed automata. *Theoretical Computer Science*, 126 :183–235, 1994.

[25] Gerard Holzmann. *SPIN model checker, the : primer and reference manual.* Addison-Wesley Professional, first edition, 2003.

[26] Gerd Berhmann, Alexandre David, John Håkansson, Martijn Hendriks, Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. Uppaal 4.0. In *3rd International Conference on Quantitative Evaluation of Systems (QEST'06)*, pages 125–126. IEEE Computer Society, Février 2006.

[27] Josée Desharnais, Edalat Abbas, and Prakash Panangaden. Bisimulation for Labelled Markov Processes. *Inf. Comput.*, 179(2) :163–193, Décembre 2002.

[28] Vincent Danos, Josée Desharnais, and Prakash Panangaden. Labelled Markov Processes : Stronger and faster approximations. *Electron. Notes Theor. Comput. Sci.*, 87 :157–203, Novembre 2004.

[29] Josée Desharnais, Gupta Vineet, Radha Jagadeesan, and Prakash Panangaden. Metrics for Labelled Markov Processes. *Theor. Comput. Sci.*, 318(3) :323–354, Juin 2004.

[30] CISMO. *http ://www.ift.ulaval.ca/ jodesharnais/cismo/*, Consulté le 09/01/2013.

[31] Kim G. Larsen. and Arne Skou. Bisimulation through probabilistic testing (preliminary report). In *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '89, pages 344–352, New York, NY, USA, 1989. ACM.

[32] Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32(1) :137–161, Janvier 1985.

[33] T. Kropf and J. Ruf. Using MTBDDs for discrete timed symbolic model-checking. In *Proceedings of the 1997 European conference on Design and Test*, EDTC '97, pages 182–, Washington, DC, USA, 1997. IEEE Computer Society.

[34] Fuzhi Wang and Marta Z. Kwiatkowska. An MTBDD-based implementation of forward reachability for probabilistic timed automata. In Doron Peled and Yih-Kuen Tsay, editors, *ATVA*, volume 3707 of *Lecture Notes in Computer Science*, pages 385–399. Springer, 2005.

[35] Holger Hermanns, Marta Kwiatkowska, Gethin Norman, David Parker, and Markus Siegle. On the use of MTBDDs for performability analysis and verification of stochastic systems. *Journal of Logic and Algebraic Programming*, 56(1-2) :23–67, 2003.

[36] Sheldon B. Akers. Binary decision diagrams. *IEEE Trans. Comput.*, 27(6) :509–516, Juin 1978.

[37] Marta Kwiatkowska, Gethin Norman, and David Parker. Advances and challenges of probabilistic model checking. In *Proc. 48th Annual Allerton Conference on Communication, Control and Computing*, pages 1691–1698. IEEE Press, 2010.

[38] Alexander Bell and Boudewijn R.Haverkort. Distributed disk-based algorithms for model checking very large markov chains. *Form. Methods Syst. Des.*, 29(2) :177–196, Septembre 2006.

[39] Avi Yadgar, Orna Grumberg, and Assaf Schuster. Languages : From formal to natural. chapter Hybrid BDD and All-SAT Method for Model Checking, pages 228–244. Springer-Verlag, Berlin, Heidelberg, 2009.

[40] Farn Wang. Symbolic parametric safety analysis of linear hybrid systems with BDD-like data-structures. *IEEE Transactions on Software Engineering*, 31(1) :38–51, 2005.

[41] P. A. Abdulla, P. Bjesse, and N. Eén. Symbolic Reachability Analysis based on SAT-Solvers. In S. Graf and M. Schwartzbach, editors, *Proc. Tools and Algorithms for the Construction and Analysis of Systems TACAS, Berlin, Germany*, LNCS 1785. Springer-Verlag, 2000.

[42] Andrew V. Goldberg and Chris Harrelson. Computing the shortest path : A search meets graph theory. In *Proceedings of the sixteenth annual ACM-SIAM symposium*

on Discrete algorithms, SODA '05, pages 156–165, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[43] Robert W. Floyd. Algorithm 97 : Shortest path. *Commun. ACM*, 5(6) :345–, Juin 1962.

[44] Christel Baier. *On the Algorithmic Verification of Probabilistic Systems*. Habilitation, Universität Mannheim, 1998.

[45] Thomas A. Henzinger, Rupak Majumdar, and Jean-François Raskin. A classification of symbolic transition systems. *ACM Trans. Comput. Logic*, 6(1) :1–32, Janvier 2005.

[46] Josée Desharnais, Prakash Panangaden, Radha Jagadeesan, and Vineet Gupta. Approximating Labeled Markov Processes. In *Proceedings of the 15th Annual IEEE Symposium on Logic in Computer Science*, LICS '00, pages 95–, Washington, DC, USA, 2000. IEEE Computer Society.

[47] Josée Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for Labelled Markov processes. *Theor. Comput. Sci.*, 318(3) :323–354, 2004.

[48] Josée Desharnais, Abbas Edalat, and Prakash Panangaden. Bisimulation for labelled markov processes. *Inf. Comput.*, 179(2) :163–193, Décembre 2002.

[49] Otaniemi Otakaari and Keijo Heljanko. Model Checking the Branching Time Temporal Logic CTL. Technical report, 1997.

[50] Jean-Pierre Demailly. *Analyse numérique et équations différentielles*. Grenoble sciences. Presses universitaires de Grenoble, 1991, Grenoble.

[51] Francinou. Serge, Gianella. Hervé, and Nicolas. Serge. *Exercices de mathématiques : Oraux x-ens, algèbre 2*. Enseignement des mathématiques. Cassini, 2009.

[52] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition : The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 3 edition, 2007.

[53] André Fortin. *Analyse numérique pour ingénieurs*. Presses internationales polytechniques edition, Mars 2009 (3ème édition).

[54] Danièle Beauquier. Markov Decision Processes and deterministic Büchi automata. *Fundam. Inf.*, 50(1) :1–13, Février 2002.

[55] Luca De Alfaro. *Formal verification of probabilistic systems*. PhD thesis, Stanford, CA, USA, 1998. AAI9837082.

[56] Vojtech Forejt, Marta Kwiatkowska, Gethin Norman, and David Parker. Automated Verification Techniques for Probabilistic Systems. In M. Bernardo and V. Issarny, editors, *Formal Methods for Eternal Networked Software Systems (SFM'11)*, volume LNCS 6659, pages 53–113. Springer, 2011.

[57] Andrea Bianco and Luca De Alfaro. Model-checking of probabilistic and nondeterministic systems. pages 499–513. Springer-Verlag, 1995.

[58] Dimitri P. Bertsekas and John N. Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16 :580–595, 1991.

[59] Marc L. Corliss Hobart and William Smith Colleges. *Java Components for Mathematics Version 1.0*, http ://math.hws.edu/javamath/, consulté le 03 Novembre 2011.

[60] IEEE Task P754. *ANSI/IEEE 754-1985, Standard for Binary Floating-Point Arithmetic*. IEEE, New York, 12 Août 1985.

[61] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.*, 23(1) :5–48, 1991.

[62] Amit Goel and Randal E. Bryant. Symbolic simulation, model checking and abstraction with partially ordered boolean functional vectors. In Rajeev Alur and Doron Peled, editors, *CAV*, volume 3114 of *Lecture Notes in Computer Science*, pages 255–267. Springer, 2004.

[63] Joost-Pieter Katoen and Ivan S. Zapreev. Simulation-based CTMC model-checking : An empirical evaluation. In *Proceedings of the Sixth International Conference on the Quantitative Evaluation of Systems, QEST '09*, pages 31–40, Los Alamitos, Septembre 2009. IEEE Computer Society Press.

[64] Josée Desharnais, François Laviolette, and Mathieu Tracol. Approximate analysis of probabilistic processes : Logic, simulation and games. In *Fifth International Conference on the Quantitative Evaluaiton of Systems (QEST 2008), 14-17 Septembre 2008, Saint-Malo, France*, pages 264–273. IEEE Computer Society, 2008.

[65] Mathieu Tracol, Josée Desharnais, and Abir Zhioua. Computing distances between probabilistic automata. In Mieke Massink and Gethin Norman, editors, *QAPL*, volume 57 of *EPTCS*, pages 148–162, 2011.

[66] Simone Tini. Non-expansive ϵ-bisimulations for probabilistic processes. *Theoretical Computer Science*, 411(22-24) :2202–2222, 2010.

[67] Nils Grimsmo, Truls Amundsen Bjørklund, and Magnus Lie Hetland. Linear computation of the maximum simultaneous forward and backward bisimulation for node-labeled trees. In *Proceedings of the 7th international XML database conference on Database and XML technologies*, XSym'10, pages 18–32, Berlin, Heidelberg, 2010. Springer-Verlag.

[68] Mathieu Tracol. *Vérification approchée de systèmes probabilistes, http ://books.google.ca/books ?id=RxldygAACAAJ.* 2010, consulté le 11 mars 2014.