Mathias Déhais, Bruno Mermet, Grégory Bonnet

Vérification formelle de propriétés de vivacité pour des systèmes multi-agents probabilistes à l'aide d'arbre à décomposition de buts

Volume 5, nº 4 (2024), p. 117-144.

https://doi.org/10.5802/roia.89

La Revue Ouverte d'Intelligence Artificielle est membre du Centre Mersenne pour l'édition scientifique ouverte
www.centre-mersenne.org
e-ISSN : 2967-9672

Revue Ouverte d'Intelligence Artificielle
Brouillon d'article, version du 6 décembre 2024

Vérification formelle de propriétés de vivacité pour des systèmes multi-agents probabilistes à l'aide d'arbre à décomposition de buts

Mathias Déhais[a], Bruno Mermet[b], Grégory Bonnet[a]

[a] Normandie Univ, UNICAEN, ENSICAEN, CNRS, GREYC
E-mail : mathias.dehais@unicaen.fr, gregory.bonnet@unicaen.fr

[b] Normandie Univ, UNIHAVRE, ENSICAEN, CNRS, GREYC
E-mail : bruno.mermet@unicaen.fr.

Résumé. — Cet article traite de la preuve formelle de systèmes multi-agents dont les comportements peuvent être caractérisés de manière probabiliste. Nous nous fondons sur le modèle GDT4MAS qui aborde la vérification formelle de la spécification jusqu'à la génération d'obligations de preuve en s'appuyant sur la logique du premier ordre, ce qui lui confère une expressivité importante. Le modèle GDT4MAS ne traite cependant pas des comportements stochastiques et ne permet pas de prouver des propriétés de vivacité. Nous étendons le modèle GDT4MAS en permettant de modéliser des agents pouvant réaliser des actions aux effets stochastiques et redéfinissons les opérateurs dans ce cadre. Afin de prouver des propriétés de vivacité, nous nous appuyons sur des méthodes de variant adaptées à notre cadre et montrons comment prouver par exemple la terminaison presque sûre d'un SMA ou la récurrence d'une propriété particulière.

Mots-clés. — Vérification formelle, systèmes multi-agents, comportements stochastiques.

1. Introduction

Les systèmes multi-agents (SMA) consistent en des agents percevant, agissant et interagissant avec leur environnement. Nous nous intéressons dans cet article aux systèmes multi-agents probabilistes (SMAP), des SMA dans lesquels le comportement et l'ordonnancement des agents peuvent être caractérisés de manière probabiliste. Il est important de noter que cette notion de stochasticité peut être introduite et observée dans les SMA à différents endroits, résumés dans la table 1. En effet, le choix de la prochaine action pour chaque agent peut suivre une certaine loi de probabilité, qui peut dépendre de l'environnement. De plus, le résultat de l'action en lui-même peut être de nature probabiliste tout comme l'ordre d'exécution des agents. Enfin, le système peut être ouvert, c'est-à-dire que des agents peuvent apparaître ou disparaître selon une certaine loi de probabilité. Dans les systèmes centralisés, la stochasticité est étudiée depuis longtemps. Parallèlement, plusieurs modèles ont été développés pour assurer la

vérification formelle des SMA. La combinaison des deux commence à être étudiée avec des méthodes comme le *model checking* mais n'a, à notre connaissance, pas encore été abordée via la preuve de théorèmes.

Source potentielle de stochasticité	Considérée ?
Choix de l'action suivante	oui
Résultat d'une action	oui
Ordre d'exécution des agents	oui
Création / destruction d'agents	non

TABLE 1.1 – Sources de stochasticité potentielles considérées dans cet article.

Dans les modèles de vérification formelle, l'intérêt est tout d'abord porté sur la preuve de propriétés de sûreté, c'est-à-dire prouver que le système n'atteindra pas un état non désirable. Il est cependant également pertinent de vouloir prouver des propriétés de vivacité, c'est-à-dire prouver que le système atteindra bel et bien un état désiré, de manière répétée ou non.

Dans cet article, nous considérons le modèle GDT4MAS utilisé pour prouver formellement des SMA, et nous l'étendons aux sources de stochasticité de la table 1. Nous proposons également une adaptation de méthodes connues pour prouver des propriétés de vivacité. Cet article est organisé comme suit. Dans la section 2, nous présentons l'état de l'art sur la vérification formelle des systèmes multi-agents et sur les systèmes stochastiques. La section 3 présente le modèle GDT4MAS et la section 4 son extension probabiliste. La section 5 aborde la preuve des propriétés de vivacité.

2. ÉTAT DE L'ART

2.1. VÉRIFICATION FORMELLE DE SMA

La preuve de systèmes multi-agents a déjà été abordée sous plusieurs angles, dont le *model checking*. Par exemple Lomuscio *et al.* [18, 19] ont développé MCMAS, un *model checker* conçu pour vérifier des SMA. Dans cet article, nous nous concentrons sur la preuve de théorèmes car le *model checking* s'appuie sur des outils de recherche exhaustive et donc est sensible au passage à l'échelle. Les méthodes de *model checking* sont particulièrement appréciées au sein de la communauté des systèmes multi-agents et des outils efficaces ont été développés, comme MCMAS ou bien PRISM. Toutefois, le problème du *model checking* est PSPACE-complet pour des sytèmes multi-agents en LTL et CTL* [25] bien qu'il existe des approches réduisant l'effet du passage à l'échelle en travaillant sur des versions abstraites du problème. Par exemple, Lomuscio et Pirovano [18] considère des *essaims* d'agents pour éviter de prendre en compte explicitement le nombre d'agents. Bien que ces abstractions présentent des limites quant à l'expressivité des propriétés prouvées et au temps de calcul impliqué par certains paramètres, il s'agit là d'approches prometteuses. La preuve par théorème de

son côté permet, par essence, d'instancier des preuves qui ne dépendent pas du nombre d'agents, se plaçant ainsi dans un tout autre paradigme et supportant beaucoup mieux le passage à l'échelle. La comparaison entre *model checking* et preuve de théorèmes n'est pas si évidente. En effet, en terme de complexité algorithmique, si la vérification par preuve de théorème est indécidable, en pratique elle est souvent proche d'une complexité polynomiale. La méthode B est l'une des principales méthodes utilisant la démonstration de théorème [1]. Cette méthode, et son extension B-événementiel, reposent sur le principe de raffinement, c'est-à-dire travailler à partir d'un niveau très abstrait et spécifier le système vers un niveau plus concret en s'assurant que chaque étape du processus est vérifiée et correcte. Une problématique identique à la nôtre s'est alors posée, *i.e.* considérer des comportements multi-agents ainsi que stochastiques. Pour l'aspect multi-agent, les travaux en B-événementiel [8, 9] utilisent des solutions ad hoc et il existe peu d'approches différentes, dont les principales sont données ci-après. Esteva *et al.* [6] ont proposé ISLANDER, conçu pour la spécification et la vérification des institutions électroniques gérées par des agents. Les institutions représentent les règles du jeu dans une société, bien adaptées pour faire face à la conception de SMA ouverts. ISLANDER se concentre sur les aspects sociaux en se référant à l'infrastructure des institutions plutôt qu'aux aspects internes des agents. Les agents sont abstraits et l'utilisateur peut choisir leur architecture et leur langage. Contrairement à ISLANDER, Bracciali *et al.* [26] ont proposé PROSOCS, qui spécifie les aspects internes des agents et utilise un modèle d'agence KGP (Knowledge, Goals, Plan). Mermet et Simon [21] ont proposé GDT4MAS, qui s'appuie sur la logique du premier ordre pour spécifier et vérifier des SMA. GDT4MAS spécifie le comportement de l'agent avec des opérateurs et des actions, en s'appuyant sur les objectifs de chaque agent pour les spécifier. Des obligations de preuve peuvent être générées afin de prouver formellement que le comportement de l'agent est correct. Ceci est rendu possible en spécifiant des arbres de décomposition de buts (GDT, *Goal Decomposition Tree*). Un GDT est la spécification de la manière dont l'objectif principal d'un agent doit être atteint, par une combinaison d'actions via des opérateurs, formant un arbre. Ces trois modèles n'intègrent pas de stochasticité. De plus, dans ISLANDER, seule la partie des comportements décrits par l'institution peut être vérifiée et non tous les comportements des agents. Dans PROSOCS, seules les propriétés exprimées en logique propositionnelle peuvent être vérifiées. Ceci est moins expressif que la logique du premier ordre qui est utilisée dans GDT4MAS. Enfin, le modèle GDT4MAS prend en compte l'ensemble du processus de preuve de théorème : spécification, génération d'obligations de preuve et preuve en elle-même. Ces arguments nous ont poussés à choisir ce modèle comme base de nos travaux.

2.2. Vérification formelle stochastique

Hors du domaine des SMA, il existe de très nombreux travaux concernant la vérification formelle de systèmes stochastiques. Kwiatkowska *et al.* [14] ont développé, et étudient toujours [16], le *model checking* probabiliste avec le logiciel PRISM. Ce travail repose sur des chaînes de Markov en temps discret, des processus de décision de Markov et des jeux multi-joueurs stochastiques. La logique sous-jacente utilisée

pour raisonner sur ces structures est classiquement PCTL, une extension probabiliste de la logique temporelle arborescente (CTL). Ils ont récemment étendu leurs travaux à des logiques plus puissantes telles que ATL* et PSL (*Probabilistic Strategy Logic*), une logique proche de la logique du premier ordre. Ils modélisent également des jeux, en raisonnant avec la logique rPATL, qui introduit la notion de récompense [15]. Le lecteur intéressé peut se référer à [7]. Ces méthodes sont destinées à explorer tous les cas possibles et à donner des probabilités explicites que les différentes propriétés d'un système soient vraies. En ce qui concerne la vérification de SMA, le principal problème reste le passage à l'échelle. McIver et Morgan [20] ont étudié les systèmes stochastiques à la fois vus comme des programmes pouvant être décrits dans un langage opératoire, et comme des fonctions. Dans cette théorie, les prédicats sont étendus avec la notion d'*expectations*. Une *expectation* permet de raisonner sur la probabilité qu'une propriété soit vraie après l'exécution d'un programme stochastique. Plus généralement, les recherches récentes [10] ont tendance à s'appuyer sur les travaux de McIver et Morgan ainsi que la méthode B [3].

Aouadhi *et al.* [2] ont étendu la méthode B-événementiel. Ce modèle ne s'appuie pas sur les travaux de McIver et Morgan [20] mais introduit des probabilités explicitement dans les machines B, en implémentant des événements probabilistes. Un événement probabiliste est un événement qui peut avoir plusieurs résultats possibles avec des probabilités associées à chaque résultat.

2.3. Vérification formelle de SMAP

Récemment, Lomuscio et Pirovano [17] ont traité de la vérification formelle pour SMA probabilistes en raisonnant sur un fragment de la logique PCTL, une logique qui permet d'introduire des branchements probabilistes. Pirovano [24] a consacré ses travaux de thèse au problème du passage à l'échelle en développant notamment les notions de systèmes paramétrisés utilisés dans les travaux conjoints avec Lomuscio. *Model checking* et preuve de théorèmes ont toujours avancé côte à côte, partageant les mêmes objectifs avec des outils distincts, mais ces derniers temps, les travaux de recherche sur la vérification de SMA se sont plus concentrés sur le model-checking. Pourtant, bien que des avancées aient été faites, le passage à l'échelle reste toujours un problème en *model checking*, c'est pourquoi il nous paraît important de poursuivre les travaux par la preuve de théorèmes.

2.4. Propriétés de vivacité

Les propriétés que nous souhaitons vérifier sur des systèmes sont généralement divisées en *propriétés de sûreté*, c'est-à-dire que quelque chose ne se produira jamais, et en *propriétés de vivacité*, c'est-à-dire que quelque chose finira par se produire. Le modèle GDT4MAS permet la preuve de propriétés de sûreté en permettant la vérification de la préservation des invariants tout au long de l'exécution du système, mais la preuve de propriété de vivacité n'a pas encore été implémentée. En B-événementiel, les propriétés de vivacité sont prises en charge à l'aide de *model-checking*. Hoang et Abrial [11] permettent certaines preuves en B-événementiel grâce à des méthodes

de variant. Hudon *et al.* ont également développé une extension de B-événementiel, unit-B [12], pour tenter d'instaurer les preuves de propriétés de vivacité au cœur du modèle. Remarquons que la vérification de propriétés de vivacité est également un sujet de recherche depuis de nombreuses années en *model checking* [16].

3. Introduction à GDT4MAS

Au vu des éléments précédents, nous choisissons d'étendre le modèle GDT4MAS pour prendre en compte des effets probabilistes. Nous présentons d'abord des éléments sur le modèle général. Nous renvoyons les lecteurs intéressés vers [21] pour plus de détails. A l'instar de modèles similaires, comme B-événementiel, le processus de preuve se déroule en 3 phases : (1) la spécification du système, qui comprend notamment la définition des invariants ; (2) la création d'obligations de preuve qui, une fois prouvées, garantissent que toutes les propriétés spécifiées sur le système sont vraies ; (3) la preuve des propriétés, déléguée à un prouveur externe comme PVS.

Nous présentons tout d'abord LTL, la logique temporelle qui sous-tend la sémantique du modèle GDT4MAS. Nous présentons ensuite la logique présente dans la spécification et dans les preuves en elle-même. En dernier lieu, nous détaillons ce qu'est un système multi-agent dans le modèle GDT4MAS.

3.1. Logique temporelle sous-jacente

Le modèle GDT4MAS utilise la logique temporelle linéaire (LTL) pour donner une sémantique opérationnelle aux GDT. LTL est construite à partir d'un ensemble fini de variables propositionnelles AP, des opérateurs logiques $\neg$ et $\vee$, et des opérateurs temporels modaux $\bigcirc$ et $\mathcal{U}$. Formellement, l'ensemble des formules de LTL sur AP est inductivement défini comme suit :

- si $p \in AP$ alors p est une formule de LTL,
- si ψ et ϕ sont des formules de LTL alors $\neg\psi$, $\phi \vee \psi$, $\bigcirc\psi$, et $\phi \mathcal{U} \psi$ le sont aussi.

Une formule de LTL peut être satisfaite par une suite infinie d'évaluations des variables dans AP. Soit $w = \omega_0, \omega_1, \omega_2, \ldots$ La relation de satisfaction $\vDash$ entre une trace et une formule de LTL est définie comme suit :

(1) $w \vDash p$ si $p \in w(0)$,
(2) $w \vDash \neg\psi$ si $w \nvDash \psi$,
(3) $w \vDash \phi \vee \psi$ si $w \vDash \phi$ ou $w \vDash \psi$,
(4) $w \vDash \bigcirc\psi$ si $w_1 \vDash \psi$ (ψ doit être vrai à l'étape suivante),
(5) $w \vDash \phi \mathcal{U} \psi$ s'il existe $i \geqslant 0$ tel que $w_i \vDash \psi$ et que pour tout $0 \leqslant k < i$, $w_k \vDash \phi$ (ϕ doit rester vrai jusqu'à ce que ψ devienne vrai).

Nous disons qu'une trace w satisfait une formule LTL ψ quand $w \vDash \psi$.

Nous considérons également les opérateurs $\wedge$, $\rightarrow$, $\leftrightarrow$, vrai et faux qui peuvent être définis à partir de ceux du langage. Nous considérons également deux autres opérateurs

temporels : □ pour toujours ; ◊ pour finalement. Ces opérateurs supplémentaires sont définis comme suit :

(1) $\phi \wedge \psi \equiv \neg(\neg\phi \vee \neg\psi)$,
(2) $\phi \rightarrow \psi \equiv \neg\phi \vee \psi$,
(3) $\phi \leftrightarrow \psi \equiv (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$,
(4) vrai $\equiv p \vee \neg p$, où $p \in AP$,
(5) faux $\equiv \neg$vrai,
(6) $\Diamond\psi \equiv$ vrai $\mathcal{U}\psi$ (ψ devient éventuellement vrai),
(7) $\Box\psi \equiv \neg\Diamond\neg\psi$ (ψ reste toujours vrai).

Exemple 3.1. — Soit w une trace représentant l'exécution d'un SMA. L'assertion $\omega_0 \vDash \Box\Diamond(stock > 50)$ signifie que la valeur de la variable *stock* finira par dépasser 50, *i.e.* il existe une infinités de mondes dans lesquels $stock > 50$ est vraie dans cette exécution. Une propriété intéressante à prouver serait alors que, pour chaque trace possible d'exécution d'un SMA, cette assertion est vérifiée.

Dans la suite, nous utilisons LTL non pas comme support pour du *model checking* sur des automates de Büchi mais pour donner une sémantique au modèle GDT4MAS. Chaque trace d'exécution d'un agent doit alors être valide selon des règles dont nous donnons quelques exemples en section 4.2.2. Ne pas être dépendant de telles structures est une des motivations du choix de cette logique, tout comme la possibilité d'exprimer des propriétés du type $\Diamond\Box\Phi$, concernant la stabilité des systèmes.

3.2. Logique du premier ordre

Nous utilisons une logique des prédicats élargie notamment aux booléens et à la logique arithmétique, donc aux opérateurs de comparaisons et d'égalité classiques. Cette logique est utilisée dans la phase de spécification essentiellement pour exprimer les conditions de satisfaction des objectifs, les résultats des actions ainsi que les conditions de branchement. Nous notons cette logique $\mathcal{L}$. Le domaine d'interprétation est défini par l'utilisateur lui-même lors de la spécification. La phase de conception nécessite de pouvoir exprimer des objectifs dont le succès dépend de l'évolution de certains prédicats, comme par exemple incrémenter une valeur. Pour cela, nous utilisons une extension, notée $\mathcal{L}'$, de la logique $\mathcal{L}$ dans laquelle nous nous référons à deux instants dans le temps. Dans $\mathcal{L}'$, les variables primées désignent les valeurs dans le second instant tandis que celles non primées désignent le premier.

Exemple 3.2. — Considérons une logique arithmétique standard, et supposons que x, y sont deux variables avec un domaine $\mathbb{N}$. La formule $x = y + 1$ est une formule dans $\mathcal{L}$, tandis que la formule $x' = y + 1$ en est une dans $\mathcal{L}'$. La première est vraie lorsque la valeur de x est actuellement égale à $y + 1$ et la seconde est vraie lorsque la valeur de x sera égale à la valeur actuelle de y augmentée de 1. Les variables primées ne peuvent pas être elles-mêmes primées, ce qui signifie qu'une formule dans $\mathcal{L}'$ se réfère à au plus deux instants dans le temps.

3.3. Agent et arbre de décomposition de buts

Dans le modèle GDT4MAS, un agent est représenté par des variables, des invariants ainsi qu'un comportement. Le comportement des agents est représenté par des GDT et chaque ensemble d'agents ayant le même comportement constitue un type d'agent pour lequel il suffit de définir un seul GDT. Ces GDT sont des arbres dont les nœuds sont des buts, définis par une condition de satisfaction et associés soit à des actions atomiques, soit à des décompositions en sous-buts. Le GDT d'un type d'agent spécifie l'ensemble du comportement de chaque agent de ce type, et la condition de satisfaction de son nœud racine est donc l'objectif principal de chacun de ces agents. Chaque agent possède bien son propre GDT défini pour son type d'agent. Cependant, et c'est ici l'un des intérêts de cette approche, les obligations de preuve peuvent être générées pour les types d'agent et non pour chaque agent. Nous ne présentons ci-dessous que les notions pertinentes pour cet article.

Le système et par conséquent les variables qui le modélisent évoluent grâce aux actions des agents.

Exemple 3.3. — Présentons un exemple qui sera utilisé tout au long de cet article. Il s'agit d'un exemple archétypique de système multi-agents contenant deux types d'agents : producteur et consommateur. Les variables sont les suivantes :

- $S_E \in, \mathbb{N}$ représente le stock global, consommé par les *consommateurs*,
- $S_{pro} \in \mathbb{N}$ représente le stock interne d'un producteur
- $S_{cons} \in \mathbb{N}$ représente le stock interne d'un consommateur

Définition 3.4 (Action). — *Une* action *a est un quadruplet :*

$$(name_a, pre_a, post_a, gpf_a)$$

où $name_a$ est le nom de l'action, pre_a est une formule dans $\mathcal{L}$ et $post_a$ et gpf_a sont des formules dans $\mathcal{L}'$. Soit ω_a un monde dans lequel un agent commence à exécuter une telle action. Alors $\omega_a \vDash pre_a$ doit être valide. $post_a$ est la formule qui est garantie être vraie si l'action réussit et gpf_a la formule qui est garantie être vraie si l'action échoue.

Il est important de noter que $post_a$ et gpf_a ne sont pas forcément exclusifs. Prenons l'exemple d'une action portant sur un entier x avec $post : x' > 10$ et $gpf : x' > 5$. Ici, nous garantissons bien des formules différentes selon la réussite ou l'échec de l'action et pourtant les deux résultats ne sont pas totalement exclusifs.

Exemple 3.5. — Soit le système producteur / consommateur. Nous considérons 4 actions, a_1, a_2, a_3, a_4 dont les détails sont donnés dans le tableau 3.1. Si *pre* ou *gpf* ne sont pas mentionnés, ils sont supposé être *vrai*. Les actions a_1 et $_2$ correspondent aux actions du type producteur. L'action a_1 permet de consommer le stock personnel du producteur et l'action a_2 de produire du stock d'environnement. Symétriquement, les actions a_3 et a_4 correspondent aux actions liées aux type du consommateur et permettent respectivement de consommer du stock global et de produire du stock personnel au consommateur. Remarquons que le producteur peut ne pas produire de stock environnemental mais qu'il consomme quand même son propre stock.

Nom	Type d'agent concerné	Effets
a_1	Producteur	$pre_{a_1} : S_{pro} > 0$ $post_{a_1} : (S'_{pro} = S_{pro} - 1)$
a_2	Producteur	$post_{a_2} : (S'_E = S_E + 1)$ $gpf_{a_2} : (S'_E = S_E)$
a_3	Consommateur	$pre_{a_3} : S_E > 1$ $post_{a_3} : (S'_E = S_E - 2)$
a_4	Consommateur	$post_{a_4} : (S'_{cons} = S_{cons} + 1)$

TABLE 3.1 – Actions des deux types d'agents

Les GDTs sont des arbres composés de deux types de nœuds : les nœuds internes et les nœuds feuilles. Seuls les nœuds feuilles sont associés à une action. Les nœuds internes servent, quant à eux, à décomposer le nœud racine – et donc le but principal – en sous-buts et permettent de déterminer quelle action doit être effectuée. Une telle décomposition d'un but en sous-buts peut être assimilée dans une certaine mesure à la notion de raffinement présente en B par exemple. Une condition de satisfaction (SC) est associée à chaque nœud N. Intuitivement, un but est satisfait si et seulement si sa SC est rendue vraie. Enfin, pour le processus de preuve, il est nécessaire de savoir ce qu'il se passe si l'exécution du nœud échoue. Cela se traduit par une propriété Garantie en cas de d'Échec (GPF).

DÉFINITION 3.6 (Nœud interne). — *Un* nœud interne N *est un tuple :*

$$(name, Op_N, Children_N, sc_N, gpf_N)$$

où $sc_N,\ gpf_N \in \mathcal{L}'$ *sont respectivement la condition de satisfaction et la garantie en cas d'échec,* i.e. *des formules satisfaites en cas de réussite ou d'échec du nœud,* Op_N *est un opérateur de décomposition, et* $Children_N$ *est une séquence de nœuds internes et de nœuds feuilles dont la longueur correspond à l'arité de* Op_N.

Chaque nœud interne d'un GDT est associé à une décomposition en sous-buts via un opérateur. Par exemple, l'opérateur `SEQAND` est un opérateur `AND` logique paresseux et ordonné. Comme autre exemple, l'opérateur `OR` permet de choisir de manière non déterministe entre les sous-objectifs, et de continuer jusqu'à ce que l'un réussisse.

DÉFINITION 3.7 (Nœud feuille). — *Un* nœud feuille N *est un tuple :*

$$(name, sc_N, gpf_N, a)$$

où $sc_N,\ gpf_N \in \mathcal{L}'$ *sont respectivement la condition de satisfaction et la garantie en cas d'échec,* i.e. *des formules satisfaites en cas de réussite ou d'échec du nœud, et a correspond à une action.*

Ainsi, les *nœuds feuilles* se comportent comme des nœuds internes à la différence qu'ils sont associés à une *action* et non pas à un opérateur de décomposition, ni à des nœuds fils.

Exemple 3.8. — Les conditions de satisfaction de chaque nœud des GDT dans le système producteur/consommateur sont données dans le tableau 3.2. Remarquons que seuls deux nœuds peuvent échouer, ceux qui possèdent un GPF.

Nodes	SC	GPF
Pro	$S'_E > S_E$	$S'_{pro} < S_{pro}$
N_1	$S'_{pro} < S_{pro}$	
N_2	$S'_E > S_E$	$S'_E = S_E$
Cons	$S'_{cons} > S_{cons}$	
N_3	$S'_E < S_E$	
N_4	$S'_{cons} > S_{cons}$	

TABLE 3.2 – SC et GPF du système producteur / consommateur

Les nœuds interne et feuilles sont organisés au sein d'un arbre qui représente le comportement global de chaque agent d'un type donné. Un agent ne peut commencer son exécution que si une formule spécifique, le *contexte de déclenchement*, est vrai. Si aucun contexte de déclenchement n'est vrai et qu'aucun agent n'a commencé son exécution, alors le système est à l'arrêt.

DÉFINITION 3.9 (GDT). — *Un* GDT *est un couple* $(tc_T, Root_T)$*, où* $pre_T, tc_T \in \mathcal{L}$ *et* $Root_T$ *est un nœud interne (donc un arbre).* tc_T *est le contexte de déclenchement, ce qui signifie que le GDT ne peut s'activer que si* tc_T *est vrai.*

Exemple 3.10. — Donnons des comportements au producteur et au consommateur de l'exemple précédent. Le but du producteur est de consommer son stock interne pour produire du stock d'environnement et inversement pour le consommateur. Nous décomposons chacun de ces buts en deux sous-buts. Les GDT, P et C des deux agents sont illustrés dans la figure 3.1. Dans N_1, le but est de consommer S_{pro} et dans N_2, le but est de produire S_E et c'est l'inverse pour N_3 et N_4. On considére également des contextes de déclenchement pour chaque agent : $tc_P : S_{pro} > 0$ et $tc_C : S_E > 1$.

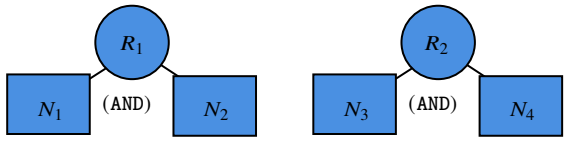

FIGURE 3.1 – GDT Producer / Consumer

Les nœuds feuilles, représentés par des rectangles dans la figure 3.1, sont différents des nœuds internes, représentés par des cercles, car les premiers contiennent des actions.

Les agents interagissent avec un environnement dont ils peuvent modifier des variables, appelées variables d'environnement. Un environnement est donc associé

à ces variables ainsi qu'à des invariants qui sont des formules sur ces variables qui doivent rester vraies tout au long de l'exécution.

Définition 3.11 (Environnement). — *Un* environnement *est un couple* $\epsilon = (V_\epsilon, i_\epsilon)$ *où* V_ϵ *est un ensemble de variables et* $i_\epsilon \in \mathcal{L}$ *désigne les invariants de l'environnement.*

Lors de la spécification d'un type d'agent, il est possible que certaines des actions décrites fassent intervenir des variables d'environnement. Pour exprimer cela, nous disons qu'un type d'agent est *relatif* à un environnement ϵ si les variables d'environnement peuvent intervenir dans les conditions de satisfaction, garanties en cas d'échec ainsi que dans les résultats des actions relatives aux agents de ce type.

Définition 3.12 (Type d'agent). — *Soit* ϵ *un environnement. Un* type d'agent A *relatif à* ϵ *est un tuple :*

$$(V_i(A), V_\epsilon(A), i_A, Actions, Beh))$$

où $V_i(A)$ *est un ensemble de variables internes qui sont instanciées pour chaque agent du type,* $V_\epsilon(A)$ *un ensemble de variables d'environnement sur lesquelles les agents de ce type peuvent agir, Actions est un ensemble d'actions,* i_A *est l'ensemble des invariants du type d'agent, Beh est un comportement modélisé par un GDT.*

Exemple 3.13. — Il existe des invariants pour chaque type d'agent tout comme pour l'environnement. Ils sont spécifiés par l'utilisateur et permettent d'assurer que certaines propriétés resteront vraies. Concernant le système producteur et consommateur, il s'agit simplement de s'assurer que les variables restent des entiers naturels comme indiqué dans la table 3.3.

Nom	Formules	Portée
EI.1	$S_E \in \mathbb{N}$	Environnement
PI.1	$S_{pro} \in \mathbb{N}$	Producteur
CI.1	$S_{cons} \in \mathbb{N}$	Consommateur

Table 3.3 – Invariants du système producteur / consommateur

Remarque 3.14. — Dans la table 3.3, l'invariant PI.1 signifie que pour tous les agents du type producteur la variable interne de chaque agent concernant son stock reste un entier naturel.

Nous disposons à présent de tous les éléments nécessaires pour définir les systèmes multi-agents.

Définition 3.15 (SMA). — *Un* système multi-agent *est un triplet* $(A, \epsilon, init)$ *où* A *est un ensemble de type d'agents,* ϵ *un environnement et init une fonction associant une valeur aux variables d'environnement, un nombre d'agent par type ainsi qu'une valeur aux variables internes de chaque agents. Soit* ω *un monde. Notons* $Act_T(\omega)$ *tous les GDT qui sont actifs ou peuvent être activés dans* ω*. Notons* $Act_a(\omega)$ *toutes les actions qui peuvent être activées dans* ω*.*

La sémantique opérationnelle des GDT est donnée par des règles en LTL. Afin d'exprimer l'évolution de l'exécution des GDT le modèle utilise des atomes LTL. Soit N un nœud, les atomes $\{init, in, end, done, sat\}$ dénotent respectivement que l'on commence à exécuter N, que l'on est en train d'exécuter N, que N vient de finir son exécution, que N a déjà été exécuté par le passé et enfin que N a réussi son exécution.

4. GDT4MAS probabiliste

Dans la section précédente, nous avons introduit le modèle GDT4MAS, Nous modifions désormais les GDT, les opérateurs et les actions. Nous commençons cette section en définissant un Système Multi-Agent Probabiliste (SMAP) en s'appuyant sur les définitions présentées dans toute la section ci-dessous. Fondamentalement, nous adaptons les définitions précédentes en introduisant des probabilités.

4.1. LTL Probabiliste

Étendons dans un premier temps la logique LTL avec un opérateur probabiliste. Comme dans la logique PCTL, où la mesure de probabilité est définie sur des ensembles de formules de chemins, nous considérons les probabilités dans la logique LTL en s'intéressant tout d'abord à l'opérateur $\mathcal{P}_{\sim\lambda}$ sur les formules LTL, où $\sim \in \{<, \leqslant, \geqslant, >, =\}$ et λ est un seuil de probabilité. Les formules de cette logique doivent être interprétées sur des chaînes de Markov discrètes, qui pourraient être instanciées pour chaque SMAP. Tous les mondes possibles sont des états et les transitions probabilistes sont données par les probabilités associées aux actions et aux opérateurs. Ensuite, nous définissons une mesure de probabilité μ_ω sur un ensemble de traces, qui est d'abord définie sur toute trace finie commençant par le monde ω, puis étendue de manière unique à toutes les traces commençant par ω. Pour plus de détails, le lecteur peut se référer à [13]. Ainsi, étant donné un monde ω, une formule LTL ψ, $\lambda \in [0, 1]$, nous avons :

$$\omega \vDash \mathcal{P}_{\sim\lambda}(\psi) \text{ si } \mu_\omega(\{\tau \in Path(\omega) \mid \tau \vDash \psi\} \sim \lambda$$

qui spécifie de manière informelle la probabilité que la formule ψ soit valide pour $\sim \lambda$.

Exemple 4.1. — Soit un SMAP qui inclut un nœud N et un monde initial ω_{init}. Alors, $\omega_{init} \vDash \mathcal{P}_{>0.5}(\omega_{init}, \Diamond in_N)$ signifie que la probabilité que le système exécute finalement le nœud N est supérieure à $0,5$.

4.2. Remplacer le non-déterminisme par des choix probabilistes

Désormais, toutes les sources de non-déterminisme dans l'exécution d'un SMA peuvent être adaptées aux comportements stochastiques.

4.2.1. *Au niveau des actions*

Dans cette section, nous introduisons la probabilité au niveau des actions, ce qui signifie que les actions peuvent désormais avoir des résultats multiples qui sont décrits par des distributions de probabilité.

Définition 4.2 (Action probabiliste). — *Une* action probabiliste *a est un tuple :*

$$(pre_a, POST_a, GPF_a, P_a)$$

où pre_a est la formule qui doit être vraie avant qu'une action n'ait lieu. $POST_a = \{post_{a,i} \mid i \in [\![1, n]\!]\}$ avec $n \in \mathbb{N}$ étant le nombre de résultats frutueux, i.e. *chaque $post_{ai}$ représente une des formules devenant vraie si l'action réussit. $GPF_a = \{gpf_{a_i} \mid i \in [\![1, k]\!]\}$ avec $k \in \mathbb{N}$ étant le nombre de résultats infructueux,* i.e. *chaque gpf_{ai} représente une des formules devenant vraie si l'action échoue. À chaque résultat est assignée une fonction dont l'évaluation permet de donner la probabilité que le résultat soit choisi. Cette fonction est contenue dans l'ensemble $P_a = \{p_{post_{a,i}} \mid i \in [\![1, n]\!]\} \bigcup \{p_{gpf_{a,i}} | i \in [\![1, k]\!]\}$ où n et k sont tels que définis précédemment. Notons par la suite $OUTCOME_a$ l'union de $POST_a$ et GPF_a. Chaque $p_{post_{a,i}}$ et $p_{gpf_{a,i}}$ sont des fonctions de $V_i(A) \times \cdots \times V_i(A)$ dans $\mathbb{R}$, ce qui signifie que la probabilité de chaque résultat dépend des valeurs des variables.*

La sémantique d'une action probabiliste a est qu'un agent ne peut tenter d'effectuer a à un instant donné que si, et seulement si, pre_a est vraie au même instant. Le résultat de l'action est probabiliste, et les probabilités $p_i \in P_a$ dépendent des valeurs des variables. Un seul résultat de $OUTCOME_a$ est garanti comme vrai après l'exécution de a et il est choisi en fonction des probabilités P_a.

Remarque 4.3. — Les résultats de $POST_a$ et GPF_a ne sont pas exclusifs. En effet, plusieurs résultats peuvent être vrais par effet de bord mais un seul est garanti vrai selon les probabilités P_a.

De manière pratique, un seul résultat de l'ensemble $OUTCOME_a$ est garanti d'être vrai, cela ne signifie pas pour autant qu'il existe un autre résultat qui aurait pu produire un résultat *similaire*, c'est entièrement à l'utilisateur de choisir.

Remarque 4.4. — Une action peut ne jamais échouer. C'est pourquoi il est possible d'avoir $GPF_a = \emptyset$. Cependant, il doit y avoir au moins un résultat où elle réussit.

Exemple 4.5. — Les actions probabilistes du système producteur/consommateur sont indiquées dans le tableau 4.1. Ici, l'action a_2 peut soit réussir soit échouer avec des probabilités fixes tandis que les probabilités concernant l'action a_3 dépendent de la valeur de S_E. De plus, l'action a_3 ne peut pas échouer mais elle a plusieurs issues possibles tandis que les actions a_1 et a_4 ne peuvent pas échouer et n'ont qu'un résultat possible.

4.2.2. *Au niveau des nœuds*

Pour une version probabiliste du modèle GDT4MAS, nous rajoutons les opérateurs probabilistes P_{OR} et P_{AND}, qui sont des versions probabilistes des opérateurs `OR` et `AND`.

Définition 4.6 (Opérateur probabiliste P_{OR}). — *L'opérateur* probabiliste `OR`*, noté P_{OR}, décompose un but en k sous-buts, les exécutions sont choisies de manière probabiliste jusqu'à ce que l'un d'eux réussisse ou que tous échouent. L'ordre est donné par un ensemble de probabilités P_{op}. À chaque sous-nœud N_i est associée une probabilité*

Nom	Type d'agent concerné	Effets
a_1	Producteur	$pre_{a_1} : S_{pro} > 0$ $post_{a_1} : (S'_{pro} = S_{pro} - 1)$
a_2	Producteur	$post_{a_2} : (S'_E = S_E + 1)$ $p_{post_{a_2}} = \frac{2}{3}$ $gpf_{a_2} : (S'_E = S_E)$ $p_{gpf_{a_2}} = \frac{1}{3}$
a_3	Consommateur	$pre_{a_3} : S_E > 1$ $post_{a_3,1} : (S'_E = S_E - 2)$ $p_{post_{a_3,1}} = \frac{S_E}{1+S_E}$ $post_{a_3,2} : (S'_E = S_E - 1)$ $p_{post_{a_3,2}} = \frac{1}{1+S_E}$
a_4	Consommateur	$post_{a_4} : (S'_{cons} = S_{cons} + 1)$

TABLE 4.1 – Actions des deux types d'agents

$P_{N_i} \in P_{op}$ (il est nécessaire de prouver que la somme des P_{N_i} vaut 1). Si toutes les exécutions échouent, l'exécution du but parent se termine. Pour marquer qu'un sous-but a été tenté, une variable temporelle $done_{N_i}$ est utilisée. La sémantique de cet opérateur est donnée par une conjonction de formules dont nous ne donnons ici que celles s'appuyant sur des effets probabilistes :

1. $\Box(\forall(i \in [\![1,k]\!]).(init_N \rightarrow Prob_{=P_{N_i}}(init_{N_i})))$
2. $\Box(\forall(i \in [\![1,k]\!]).(end_{N_i} \wedge \neg sat_{N_i} \wedge \exists(j \in [\![1,k]\!]).\neg done_{N_j}) \rightarrow \mathcal{P}_{=P_j}(\bigcirc_A(init_{N_j})))$

où :

$$P_j = P_{N_j} \times \frac{1}{\sum_{\substack{l \in [\![1,k]\!]. \\ \neg done_{N_l}} P_{N_l}}}$$

correspond à la probabilité que N_j soit exécuté au monde suivant sachant quels nœuds ont déjà été exécutés.

Dans la définition 4.6, la formule 1 signifie que chaque sous-nœud N_i a une probabilité P_{N_i} de s'exécuter en premier. La formule 2 signifie que lorsqu'un sous-nœud termine son exécution sans atteindre le sous-but, la probabilité qu'un nœud donné N_i soit exécuté vaut P_j, *i.e.* sa probabilité initiale divisée par la somme des probabilités des sous-nœuds non-exécutés.

DÉFINITION 4.7 (Opérateur probabiliste $P_{\texttt{AND}}$). — *L'opérateur* probabiliste `AND`*, noté $P_{\texttt{AND}}$, décompose un but en k sous-buts, les exécutions sont choisies de manière probabiliste jusqu'à ce que l'un d'eux échoue ou que tous réussissent. L'ordre est donné par un ensemble de probabilités P_{op}. À chaque sous-nœud N_i est associée une probabilité $P_{N_i} \in P_{op}$ (il est nécessaire de prouver que la somme des P_{N_i} vaut 1). Si toutes les exécutions réussissent, l'exécution de le but parent se termine. Pour marquer qu'un sous-but a été tenté, une variable temporelle $done_{N_i}$ est utilisée. La sémantique*

de cet opérateur est donnée par une conjonction de formules dont nous ne donnons ici que celles contenant des considérations probabilistes :

1. $\Box(\forall(i \in [\![1,k]\!]).(init_N \rightarrow \mathcal{P}_{=P_{N_i}}(init_{N_i})))$
2. $\Box(\forall(i \in [\![1,k]\!]).(end_{N_i} \wedge sat_{N_i} \wedge \exists(j \in [\![1,k]\!]).\neg done_{N_j}) \rightarrow \mathcal{P}_{=P_j}(\bigcirc_A(init_{N_j})))$

où :

$$P_j = P_{N_j} \times \frac{1}{\sum_{\substack{l \in [\![1,k]\!]. \\ \neg done_{N_l}}} P_{N_l}}$$

correspond à la probabilité que N_j soit exécuté au monde suivant sachant quels nœuds ont déjà été exécutés.

L'opérateur $P_{\texttt{AND}}$ est défini de manière parfaitement similaire à $P_{\texttt{OR}}$ à l'exception que l'exécution de l'objectif parent se termine avec succès si tous les nœuds réussissent et échoue si l'exécution de l'un des nœuds fils échoue.

4.2.3. *Au niveau des GDT*

Dans le modèle GDT4MAS non stochastique, lorsque plusieurs GDT peuvent être activés ou sont déjà actifs, le choix du prochain GDT qui sera actif est non-déterministe. Afin d'introduire un choix probabiliste entre les GDT qui deviendront actifs, nous associons un poids à chaque GDT pour définir un GDT probabiliste (PGDT). Étant donné le poids d'un GDT, nous pouvons calculer la probabilité qu'il devienne actif : plus le poids d'un type d'agent est élevé parmi tous les types d'agent, plus sa probabilité qu'un agent de ce type soit choisi pour être exécuté est élevée.

Définition 4.8 (GDT probabiliste (PGDT)). — *Un PGDT T est un tuple :*

$$(tc_T, Root_T, m_T)$$

où tc_T et $Root_T$ sont définis comme pour la définition 3.9 et $m_T \in \mathbb{N}$ représente le poids du GDT. Soit A un agent avec T son GDT possédant un poids de m_T, la probabilité d'être actif après qu'une action a été effectué est :

$$P_T = \begin{cases} 0 & si \sum_{D \in Act_T(\omega)} m_D = 0, \\ \frac{m_T}{\sum_{D \in Act_T(\omega)} m_D} & sinon. \end{cases}$$

La probabilité d'être choisi est donc calculée en fonction des poids des agents actifs ou activables.

Exemple 4.9. — La figure 4.1 reprend l'exemple du système producteur / consommateur en y intégrant les concepts probabilistes introduit précédemment d'une part sur les nœuds et d'autres sur l'ordonnancement des GDT. Ici, les GDT comportent des nœuds probabilistes et le non-déterminisme de l'ordonnancement est remplacé par un branchement probabiliste. Le poids associé aux PGDT signifie que chaque agent consommateur a deux fois plus de chance d'être exécuté qu'un agent producteur.

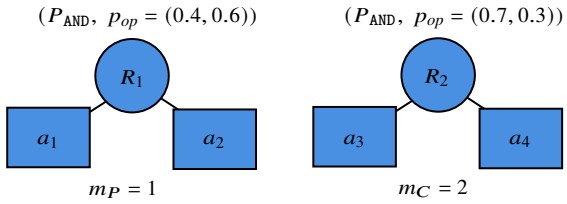

FIGURE 4.1 – PGDT du producteur et du consommateur

Afin de vérifier des propriétés de vivacité, nous avons besoin de déterminer si la probabilité qu'une action s'exécute une fois son PGDT activé est strictement positive. C'est la notion de nœud accessible.

DÉFINITION 4.10 (Nœud accessible). — *Un nœud N est* accessible *si la probabilité de l'exécuter, sachant que le PGDT auquel il appartient peut être activé et actif, est non nulle. Soit N un nœud et T le PGDT auquel il appartient. Soit tc le contexte de déclenchement de T. Le nœud N est* accessible *si, et seulement si,*

$$(\Box(m_T > 0) \wedge \Diamond init_{Root_T}) \rightarrow \mathcal{P}_{>0}(\Diamond init_N)$$

DÉFINITION 4.11 (SMAP). — *Un* système multi-agent probabiliste *(SMAP) est un SMA (définition 3.15) dont les agents ont des comportements décrits par des PGDT.*

4.3. OBLIGATIONS DE PREUVE ET VÉRIFICATION

Les différentes obligations de preuve (PO) sont générées grâce à des schémas de preuve. Fondamentalement, un schéma de preuve est un patron de PO, et ils doivent être conçus de telle sorte que si toutes les PO sont instanciées et vérifiées, l'ensemble du système est correctement prouvé. L'objectif principal de la génération de PO est de vérifier que chaque invariant reste vrai pendant l'exécution du système. De manière générale, cette vérification peut être effectuée automatiquement par un démonstrateur de théorème comme PVS. Les lecteurs intéressés peuvent se référer à [23].

Un point très important à souligner est que le nombre de preuves engendrées par les obligations de preuve ne varie pas selon le nombre d'agents. En effet, chaque schéma de preuve porte sur un type d'agent, et donc un GDT, et non sur un agent en lui-même. Ainsi, par exemple, pour vérifier que les actions des agents ne transgressent pas un invariant, il suffit de prouver cet état de fait une seule fois par action, et non une fois par action par agent. Cela représente l'atout principal d'un modèle de preuve par théorèmes par rapport au *model checking*.

Un *contexte* est ajouté à presque toutes les obligations de preuve. En effet, certaines informations peuvent être déduites de chaque nœud et peuvent être utilisées pour générer des obligations de preuve plus facilement prouvables, puisque les contextes et les GPF seront des hypothèses supplémentaires.

DÉFINITION 4.12 (Contexte). — *Soit T un GDT, et soit N un nœud de T. Le* contexte *de N, noté c_N, est une formule de $\mathcal{L}$ qui est logiquement équivalente à une sous-partie*

de la conjonction de toutes les formules $\psi \in \mathcal{L}$ telles que pour tous les mondes ω satisfaisant $init_N$, ω satisfait ψ.

Nous renvoyons le lecteur à [22] pour plus de détails sur le calcul des contextes. Le point essentiel dans cet article est que les contextes fournissent des informations qui peuvent aider durant la phase de preuve.

La version probabiliste du modèle nécessite l'introduction d'invariants liés à la définition probabiliste des actions et des opérateurs. En effet, il faut préserver le fait que les probabilités soient des nombres réels compris entre 0 et 1 et qui somment à 1 pour chaque action et opérateur à chaque instant. Ces invariants sont donnés dans la table 4.2.

Table 4.2 – Invariants relatifs aux probabilités

Nom	formule
INV.proba1	$\forall p \in P_a, 0 \leqslant p \leqslant 1$
INV.proba2	$\sum_{p, p \in P_a} = 1$
INV.proba3	$\forall p \in P_{op}, 0 \leqslant p \leqslant 1$
INV.proba4	$\sum_{p, p \in P_{op}} = 1$
INV.proba5	$\forall m_T \in M, 0 \leqslant m_T$

Nous n'avons pas besoin d'introduire explicitement de nouveaux schémas de preuve dans le modèle pour prouver que les probabilités sont correctes. En effet, nous nous appuyons sur le fait que les invariants doivent être préservés par les PO. Cependant, nous devons adapter les schémas de preuve existants pour que notre dernière phrase soit vraie. En ce qui concerne les PO associées aux actions, nous devons montrer que leurs nœuds feuilles respectent les invariants d'agents et d'environnement. Cela est capturé par des schémas de preuve d'action définis comme suit :

Définition 4.13 (Schémas de preuve d'action). — *L'exécution d'une action doit préserver les invariants. Pour chaque $outcome_a \in OUTCOME_a$:*

$$i_\epsilon \wedge i_A \wedge c_N \wedge outcome'_a \vDash i'_\epsilon \tag{4.1}$$

$$i_\epsilon \wedge i_A \wedge c_N \wedge outcome'_a \vDash i'_A \tag{4.2}$$

où c_N est le contexte d'un nœud, c'est-à-dire toutes les formules qui sont garanties vraies au moment de l'exécution du nœud N.

Ces schémas de preuve signifient que, quel que soit le déroulement de l'exécution de l'action, celle-ci doit préserver les invariants. Il faut aussi montrer que, si l'action réussit, la condition de satisfaction associée aux nœuds feuilles est vérifiée (et la GPF sinon).

Nous devons également montrer que les actions induisent correctement le succès ou l'échec du nœud feuille correspondant. Rappelons que les littéraux sc_N et gpf_N désignent respectivement la condition de satisfaction et la garantie en cas d'échec du

nœud N, c'est-à-dire les prédicats de $\mathcal{L}'$ qui doivent être montrés vrais pour dire que le nœud réussit ou échoue. Ceci est capturé par des schémas de preuve de nœuds feuilles définis comme suit :

Définition 4.14 (Schémas de preuve de nœuds feuille).

$$i_\epsilon \wedge i_A \wedge c_N \vDash pre_a \tag{4.3}$$

$$\textit{Pour chaque } post_a \in POST_a : i_\epsilon \wedge i_A \wedge c_N \wedge post'_a \vDash sc'_N \tag{4.4}$$

$$\textit{Pour chaque } gpf_a \in GPF_a : i_\epsilon \wedge i_A \wedge c_N \wedge gpf'_a \vDash gpf'_N \vee sc'_N \tag{4.5}$$

Ces schémas de preuve signifient que si une action réussit (ou échoue), alors le nœud feuille correspondant doit voir sa condition de satisfaction (ou sa GPF) vérifiée.

Exemple 4.15. — Reprenons l'exemple du SMAP producteur/consommateur donné en exemple 4.5. Nous devons montrer que les probabilités somment bien à 1 pour chaque action et restent comprises entre 0 et 1. Les obligations de preuve ainsi générées sont pour l'action a_1 :

$$\frac{\begin{array}{l} 0 \leqslant p_{post_{a2}} \leqslant 1 \\ 0 \leqslant p_{gpf_{a2}} \leqslant 1 \\ (p_{post_{a_2}} + p_{gpf_{a_2}} = 1) \\ c_N \\ post_{a_2} \wedge gpf_{a_2} \end{array}}{(0 \leqslant p'_{post_{a_2}} \leqslant 1) \wedge (0 \leqslant p'_{gpf_{a_2}} \leqslant 1) \wedge (p'_{post_{a_2}} + p'_{gpf_{a_2}} = 1)}$$

$$\frac{\begin{array}{l} 0 \leqslant p_{post_{a2}} \leqslant 1 \\ 0 \leqslant p_{gpf_{a2}} \leqslant 1 \\ p_{post_{a_2}} + p_{gpf_{a2}} = 1 \\ c_N \\ gpf_{a2} \end{array}}{(0 \leqslant p'_{post_{a_2}} \leqslant 1) \wedge (0 \leqslant p'_{gpf_{a2}} \leqslant 1) \wedge (p'_{post_{a2}} + p'_{gpf_{a2}} = 1)}$$

Nous obtenons alors en remplaçant par les valeurs :

$$\frac{\begin{array}{l} 0 \leqslant \frac{2}{3} \leqslant 1 \\ (0 \leqslant \frac{1}{3} \leqslant 1 \\ \frac{2}{3} + \frac{1}{3} = 1 \\ c_N \\ S'_E = S_E + 1 \end{array}}{\left(0 \leqslant \frac{2}{3} \leqslant 1\right) \wedge \left(0 \leqslant \frac{1}{3} \leqslant 1\right) \wedge \left(\frac{2}{3} + \frac{1}{3} = 1\right)}$$

$$\frac{\left(0 \leqslant \frac{2}{3} \leqslant 1\right) \quad \left(0 \leqslant \frac{1}{3} \leqslant 1\right) \quad \left(\frac{2}{3} + \frac{1}{3} = 1\right) \quad c_N \quad S'E = S_E}{\left(0 \leqslant \frac{2}{3} \leqslant 1\right) \wedge \left(0 \leqslant \frac{1}{3} \leqslant 1\right) \wedge \left(\frac{2}{3} + \frac{1}{3} = 1\right)}$$

Ce qui est directement vrai puisque les probabilités sont fixes. Pour l'action a_3 nous obtenons :

$$\frac{(0 \leqslant p_{post_{a_3,1}} \leqslant 1) \quad (0 \leqslant p_{post_{a_3,2}} \leqslant 1) \quad (p_{post_{a_3,1}} + p_{post_{a_3,2}} = 1) \quad post_{a_3,1} \quad pre_{a_3}}{(0 \leqslant p'_{post_{a_3,1}} \leqslant 1) \wedge (0 \leqslant p'_{post_{a_3,2}} \leqslant 1) \wedge (p'_{post_{a_3,1}} + p'_{post_{a_3,2}} = 1)}$$

$$\frac{(0 \leqslant p_{post_{a_3,1}} \leqslant 1) \quad (0 \leqslant p_{post_{a_3,2}} \leqslant 1) \quad (p_{post_{a_3,1}} + p_{post_{a_3,2}} = 1) \quad post_{a_3,2} \quad pre_{a_3}}{(0 \leqslant p'_{post_{a_3,1}} \leqslant 1) \wedge (0 \leqslant p'_{post_{a_3,2}} \leqslant 1) \wedge (p'_{post_{a_3,1}} + p'_{post_{a_3,2}} = 1)}$$

Nous obtenons alors :

$$\frac{0 \leqslant \frac{S_E}{1+S_E} \leqslant 1 \quad 0 \leqslant \frac{1}{1+S_E} \leqslant 1 \quad \frac{S_E}{1+S_E} + \frac{1}{1+S_E} = 1 \quad S'_E = S_E - 2 \quad S_E > 1}{\left(0 \leqslant \frac{S'_E}{1+S'_E} \leqslant 1\right) \wedge \left(0 \leqslant \frac{1}{1+S'_E} \leqslant 1\right) \wedge \left(\frac{S'_E}{1+S'_E} + \frac{1}{1+S'_E} = 1\right)}$$

$$\frac{\begin{array}{l} 0 \leqslant \dfrac{S_E}{1+S_E} \leqslant 1 \\ 0 \leqslant \dfrac{1}{1+S_E} \leqslant 1 \\ \dfrac{S_E}{1+S_E} + \dfrac{1}{1+S_E} \\ S'_E = S_E - 1 \\ S_E > 1 \end{array}}{\left(0 \leqslant \dfrac{S'_E}{1+S'_E} \leqslant 1\right) \wedge \left(0 \leqslant \dfrac{S'_E}{1+S'_E} \leqslant 1\right) \wedge \left(\dfrac{S'_E}{1+S'_E} + \dfrac{1}{1+S'_E} = 1\right)}$$

Remplaçons S'_E par son expression en fonction de S_E

$$\frac{\begin{array}{l} 0 \leqslant \dfrac{S_E}{1+S_E} \leqslant 1 \\ 0 \leqslant \dfrac{1}{1+S_E} \leqslant 1 \\ \dfrac{S_E}{1+S_E} + \dfrac{1}{1+S_E} = 1 \\ S_E > 1 \end{array}}{\left(0 \leqslant \dfrac{S_E - 2}{S_E - 1} \leqslant 1\right) \wedge \left(0 \leqslant \dfrac{1}{S_E - 1} \leqslant 1\right) \wedge \left(\dfrac{S_E - 2}{S_E - 1} + \dfrac{1}{S_E - 1} = 1\right)}$$

$$\frac{\begin{array}{l} 0 \leqslant \dfrac{S_E}{1+S_E} \leqslant 1 \\ 0 \leqslant \dfrac{1}{1+S_E} \leqslant 1 \\ \dfrac{S_E}{1+S_E} + \dfrac{1}{1+S_E} \\ S_E > 1 \end{array}}{\left(0 \leqslant \dfrac{S_E - 1}{S_E} \leqslant 1\right) \wedge \left(0 \leqslant \dfrac{S_E - 1}{S_E} \leqslant 1\right) \wedge \left(\dfrac{S_E - 1}{S_E} + \dfrac{1}{S_E} = 1\right)}$$

Comme $S_E > 1$, nous obtenons bien que toutes ces obligations de preuves sont vraies.

Nous avons décrit le modèle GDT4MAS adapté aux systèmes multi-agents probabilistes. Nous pouvons maintenant permettre la vérification de certaines preuves de vivacité en nous appuyant sur la définition de l'accessibilité d'une action, qui est caractérisée grâce à des distributions de probabilités.

5. Propriétés de vivacité

Nous tirons partie du modèle probabiliste pour permettre la preuve de propriétés de vivacité grâce à la notion de nœud accessible. Nous proposons une approche similaire

à celle de Hoang et Abrial [11] puisque cela ne nécessite pas d'ajouter plus d'outils au modèle.

5.1. Méthodes de variant

Nous adaptons ici l'approche de Hoang et Abrial [11] à notre modèle. Nous utilisons toujours la logique LTL. Soit $P \in \mathcal{L}$, les mondes ω satisfaisant la propriété P sont appelés P-mondes. Pour une trace $\sigma = (w_0, w_1, \dots)$, $\sigma \vDash P$ si et seulement si w_0 est un monde dans lequel P est vérifiée.

Définition 5.1. — *Un SMAP S vérifie une formule temporelle ϕ ($S \vDash \phi$) si, et seulement si, toutes ses traces possibles vérifient ϕ.*

Nous écrivons $S \vdash \phi$ pour signifier que $S \vDash \phi$ est démontrable.

Afin de pouvoir prouver les propriétés de vivacité, nous nous appuyons sur les règles de Hoang et Abrial. Cependant, nous devons les adapter à notre modèle. Premièrement, nous avons besoin de la propriété suivante pour prouver que notre méthode est correcte.

Propriété 5.2. — *Une exécution d'un GDT consiste toujours en un nombre fini d'actions.*

Démonstration. — Les opérateurs qui n'ont pas été cités dans cet article sont `ITER` et `CASE`. `ITER` permet de répéter une partie de l'arbre un nombre fini de fois, des obligations de preuves sont présentes pour assurer que le nœud contenant l'opérateur `ITER` termine et réussit assurément en un nombre fini d'étapes. En pratique, un nœud contenant un opérateur `ITER` ne possède qu'un unique sous-fils. Si le système a été prouvé, alors il est impossible que ce nœud fils s'exécute une infinité de fois puisque le nœud père doit réussir en un nombre fini d'étapes.

Le fonctionnement de l'opérateur `CASE` est tel qu'il garantit l'exécution d'un seul de ses fils. L'idée est de permettre des alternatives selon l'état du système ou de l'agent, cependant, un seul fils est choisi et les autres ne seront pas exécutés. Une fois cette exécution terminée, le nœud contenant l'opérateur `CASE` est considéré comme terminé. Il n'y a qu'une seule exécution de ce nœud fils et lorsque ce-dernier est marqué comme terminé, il en est de même pour le nœud père. Dans l'éventualité où les nœuds fils contiennent tous des opérateurs `ITER` , il n'y a quand même qu'une seule exécution d'un nœud fils, qui se fera en un nombre fini d'étapes selon ce que nous avons écrit précédemment sur l'opérateur `ITER` , il n'est donc pas possible d'engendrer une boucle infinie avec un opérateur `CASE`.

Pour le comportement des nœuds contenant les opérateurs P_{OR}, P_{AND}, `SEQOR` et `SEQAND` présentés en section 4.2.2, chaque sous-nœud ne peut être exécuté au plus qu'une seule fois puisqu'une fois leur exécution terminée ils sont dénotés comme terminés et ayant réussi ou échoué, ce qui permet de savoir si l'on exécute un autre nœud fils ou bien si le nœud père est terminé. Une fois qu'un nœud fils est terminé, il ne peut pas être réexécuter comme spécifié par la règle 2 des opérateurs probabilistes. Concernant les opérateurs séquentiels, il n'est pas non plus possible de revenir en arrière sur un nœud précédemment exécuté.

Ainsi, pour une itération du nœud parent, chaque fils du nœud associé est exécuté un nombre fini de fois, au plus un pour tous les opérateurs sauf pour `ITER`. Étant donné que chaque nœud contient un nombre fini de fils, nous avons montré que chaque exécution de GDT était finie, ce qui revient donc à dire qu'une exécution consiste en un nombre fini d'actions puisqu'au moins un nœud feuille est exécuté. □

Ensuite, nous devons définir le fait qu'un système mène d'un P_1-monde à un P_2-monde, avec $P_1, P_2 \in \mathcal{L}$. En effet, cette notion de mener d'un monde à un autre est utile pour prouver des propriétés d'invariance ou des dépendances entre propriétés.

Définition 5.3 (Mener à). — *Un SMAP S mène de P_1 à P_2 si, pour toute trace possible, $S \vDash \Box(P_1 \rightarrow \bigcirc P_2)$, c'est-à-dire que si le SMAP est dans un P_1-monde, chaque action possible mène à un P_2-monde. Cette propriété se vérifie en prouvant que pour chaque action a dans S, pour chaque $outcome_a \in OUTCOME_a$:*

$$pre_a \wedge C_a \wedge outcome'_a \wedge P_1 \Rightarrow P'_2$$

Ensuite, nous devons définir (pour pouvoir le prouver) le fait qu'un système converge en P, ce qui signifie que chaque trace infinie se termine par une quantité infinie de P-monde.

Définition 5.4 (Convergence en P). — *Un SMAP S est convergent en $P \in \mathcal{L}$ si toute trace infinie de celui-ci se termine par une suite infinie de P-mondes. Cette propriété peut se vérifier en suivant une adaptation des méthodes de variant présentées notamment par Hoang et Abrial* [11]*, c'est-à-dire :*

- *Soit une expression entière V, le variant.*
- *Lorsque le SMAP est dans un P-monde, prouver que pour chaque action a :*

$$pre_a \wedge C_a \wedge P \Rightarrow V \in \mathbb{N} \tag{5.1}$$

  et pour chaque $outcome_a \in OUTCOME_a$:

$$pre_a \wedge C_a \wedge outcome'_a \wedge P \Rightarrow V' \leqslant V \tag{5.2}$$

- *Prouver qu'il existe au moins une action d'un nœud feuille accessible dans chaque type de GDT pour laquelle, dans un monde $\neg P$:*

$$pre_a \wedge C_a \wedge (post'_a \vee gpf'_a) \wedge \neg P \Rightarrow V' < V \tag{5.3}$$

Proposition 5.5. — *La méthode de la définition 5.4 prouve la convergence d'un SMAP S vers une propriété $P \in \mathcal{L}$.*

Démonstration. — Supposons qu'une trace se termine par une suite infinie de $\neg P$-mondes. V a donc une probabilité 1 d'être diminué infiniment souvent. En effet, tout d'abord il y a une action d'un nœud feuille accessible par GDT qui fait diminuer le variant (voir condition (5.3) ci-dessus). Comme l'exécution de chaque GDT est finie, d'après la propriété 5.2, et que la trace est infinie par hypothèse, il y a une infinité d'activation de GDT. Comme pour chaque exécution de GDT, il y a une probabilité non nulle de diminuer le variant et qu'aucune action n'augmente le variant (voir condition (5.2) ci-dessus), il est infiniment diminué de manière presque sûre. Cependant, puisque $V \in \mathbb{N}$ dans les $\neg P$-mondes, cela constitue une contradiction.

Par conséquent, il ne peut y avoir de trace qui se termine par une suite infinie de $\neg P$-mondes. □

Par la suite, notons $S \vdash\!\downarrow P$ le fait que « S est convergent en P » est prouvable.

Définition 5.6 (Sans blocage). — *Un SMAP S est sans blocage en P si toute trace finie de S ne se termine pas en un P-monde.*

Par la suite, étant donné un SMA S possédant k types d'agents, notons TC_k la disjonction de tous les contextes de déclenchement des agents partageant le k-ème type.

Pour prouver une telle propriété, nous écrivons que si tous les contextes C_N de tous les nœuds feuilles sont faux, alors le SMAP est bloqué. Cependant, l'inverse n'est pas vrai car le système peut s'arrêter pour d'autres raisons. Une condition suffisante de non-blocage en P est donc :

Propriété 5.7. — *Soit S un SMAP, avec k types d'agents, alors si $P \Rightarrow \bigvee_{i=1}^{k}(TC_i)$, avec TC_i la disjonction des contextes de déclenchement des agents du i-ème type alors S est sans blocage en P.*

Démonstration. — Si un seul contexte de déclenchement est vrai, il suffit de s'assurer que le SMAP n'est pas bloqué, donc si P implique qu'au moins un contexte de déclenchement est vrai, cela signifie qu'il ne peut y avoir de trace finie qui se termine par un P-monde. □

Remarque 5.8. — C'est une condition suffisante mais elle n'est pas nécessaire puisque tous les contextes de déclenchement peuvent être faux à un instant donné alors que les agents ont encore des actions à exécuter.

Par la suite, notons par $S \vdash\!\circlearrowleft P$ le fait que « S est sans blocage dans n'importe quel P-monde est prouvable ».

5.2. Règle de preuve

Nous adaptons désormais les méthodes proposées par Hoang et Abrial au modèle GDT4MAS probabiliste. Nous pouvons maintenant appliquer les règles pour prouver les propriétés de vivacité comme la *persistance*, c'est-à-dire qu'une propriété doit finalement rester vraie pour toujours. Nous faisons pour cela référence à la règle développée par Hoang et Abrial donnée en table 5.1.

Table 5.1 – Règle pour la propriété d'apparition

$$\frac{S \vdash\!\downarrow P \qquad S \vdash\!\circlearrowleft \neg P}{S \vdash \Diamond\Box P}$$

Cette règle signifie que si un système converge en P et est sans blocage en $\neg P$ alors il en découle que P finira par rester vrai indéfiniment. Nous considérons principalement

cette règle dans la suite, mais le lecteur peut se référer à [11] pour les autres règles. Si l'utilisateur spécifie le variant et les actions qui le font décroître, ces preuves sont automatisables de la même manière que les preuves classiques. En effet, une fois le variant et les actions qui le font décroître connues, un logiciel comme PVS peut prouver les points présentés en définition 5.4 et en propriété 5.7.

Remarquons que nous retrouvons également la vérification de propriétés de sûreté, notamment la vérification des invariants d'environnement avec la règle données en table 5.2.

TABLE 5.2 – Règle pour l'invariance

$$\frac{\vdash Init \Rightarrow I \qquad S \vdash I \rightarrow I}{S \vdash \Box I}$$

Cette règle permet de vérifier des propriétés d'invariance globale et est donc appliquée en pratique uniquement pour les invariants d'environnement puisque, pour les invariants d'agents, il n'y a pas besoin de vérifier le comportement de tous les autres types d'agent que celui dont nous prouvons les invariants.

5.3. LE CAS DE LA TERMINAISON

La terminaison, qui consiste à prouver que le système finira par se terminer, est toujours une question importante en preuve formelle. Nous proposons ici une première approche pour prouver la terminaison d'un SMAP. Remarquons que les spécifications stochastiques données en section 4 n'interviennent pas dans cette approche, mais elles seront nécessaires pour prouver davantage de cas de terminaison dans le futur.

Nous caractérisons qu'un SMAP s'est terminé lorsque tous les contextes de déclenchement sont faux et qu'aucun agent n'a d'action en attente, c'est-à-dire que toutes les exécutions de GDT sont terminées et qu'aucune ne peut commencer. Bien que notre modèle ne permette pas d'écrire formellement la fin d'exécution d'un GDT dans le cas général, nous pouvons néanmoins exprimer cette propriété en nous appuyant uniquement sur les contextes de déclenchement.

PROPOSITION 5.9. — *Soit S un SMAP qui possède k types d'agents, c'est-à-dire qu'il y a k types de GDT différents. Notons $TC_1, \ldots, TC_k$ les disjonctions respectives des contextes de déclenchement des différents GDT. S se termine finalement si, et seulement si, $S \vDash \Diamond\Box(\neg TC_1 \wedge \cdots \wedge \neg TC_k)$.*

Ainsi, un SMAP S se termine finalement si S vérifie que tous les contextes de déclenchement finiront par rester faux.

Démonstration. — ($\Rightarrow$) Supposons que le SMAP se termine dans le futur alors il existe un monde ω dans lequel les contextes de déclenchement sont faux. ($\Leftarrow$) Supposons que $S \vDash \Diamond\Box(\neg TC_1 \wedge \cdots \wedge \neg TC_k)$. L'exécution d'un GDT est finie selon la

propriété 5.2. Puisqu'il existe un monde à partir duquel les contextes de déclenchement restent faux, il existe bien un monde futur dans lequel toutes les exécutions sont terminées. □

Prouver $S \vDash \Diamond\Box(\neg TC_1 \wedge \cdots \wedge \neg TC_k)$ nécessite d'écrire un variant unique sur tous les types de GDT, ce qui n'est pas pratique. Nous pouvons alors utiliser le fait que $\Diamond\Box\neg TC_1 \wedge \cdots \wedge \Diamond\Box\neg TC_k$ est équivalent à $\Diamond\Box(\neg TC_1 \wedge \cdots \wedge \neg TC_k)$, puis prouver chacun d'eux séparément.

Selon la règle de la table 5.1, pour prouver $\Diamond\Box\neg TC$ il est nécessaire de d'abord prouver $S \vdash\!\downarrow \neg TC$ en définissant un variant. Ensuite, il est nécessaire de prouver $S \vdash\!\circlearrowleft TC$. Or, cela est toujours trivial puisqu'il s'agit de la définition d'un SMAP sans blocage. En effet, si un contexte de déclenchement est vrai, le SMAP ne peut pas être bloqué. Il ne reste que la première propriété à prouver.

Remarque 5.10. — Si nous montrons qu'un type de GDT finit par se terminer, alors il est certain qu'il existe un monde dans lequel tous les agents partageant ce type de GDT ont définitivement terminé leur exécution. Ainsi, nous pouvons ne pas considérer les actions du GDT pour le reste de la preuve, et donc la simplifier.

Remarque 5.11. — Si un variant ne contient que des variables internes d'un même type d'agent, seules ses actions peuvent être prises en compte pour la preuve de convergence. En effet, nous pouvons supposer que ce type d'agent s'exécute autant de fois que nécessaire car, s'il ne le fait pas, c'est qu'il est terminé et aucun autre agent d'un autre type ne peut modifier le variant.

Exemple 5.12. — Reprenons l'exemple du SMAP producteur / consommateur donné en exemples 4.5 et 4.9. Supposons que le système possède p agents de type producteur et m agents de type consommateur avec $p, m \in \mathbb{N}$. Nous indiçons les variables internes afin d'indiquer à quel agent elles appartiennent. Nous voulons prouver que le SMAP finit par se terminer. Pour ce faire, nous devons prouver que $S \vDash \Diamond\Box\neg TC_P$ et que $S \vDash \Diamond\Box\neg TC_C$.

Concentrons-nous sur le producteur. Nous voulons prouver :

$$S \vDash \Diamond\Box\neg\left(\bigvee_{i=1}^{p} S_{pro,i} > 0\right)$$

Pour ce faire, nous devons prouver :

$$S \vDash\!\downarrow \left(\bigwedge_{i=1}^{p} (S_{pro,i} \leqslant 0)\right) \tag{5.4}$$

$$S \vDash\!\circlearrowleft \left(\bigwedge_{i=1}^{p} (S_{pro,i} \leqslant 0)\right) \tag{5.5}$$

Comme indiqué en section 5.3, (5.5) est déjà prouvée. Pour la preuve que S est convergent en $(\neg TC_P)$, considérons le variant $V = \sum_{i=1}^{p}(S_{pro,i})$. Par définition, aucune action du type d'agent producteur ne peut faire augmenter le variant. De plus, dans

le cadre où $\bigvee_{i=1}^{p}(S_{pro,i} > 0)$ est vrai, l'action a_2 du type producteur fait décroître le variant et comme le variant ne contient que des variables internes du type producteur, on peut ignorer les autres pour le cas particulier de la terminaison.

Nous avons prouvé que $S \vdash \Diamond\Box\neg TC_P$. Prouvons maintenant $S \vdash \Diamond\Box\neg TC_C$. Pour ce faire, nous répétons exactement le même processus, en nous appuyant sur le fait que nous avons déjà prouvé que le producteur se termine. Dès lors, nous pouvons nous placer dans un monde où celui-ci est déjà terminé.

Nous voulons maintenant prouver :

$$S \vDash \Diamond\Box\neg\left(\bigvee_{i=1}^{m}(S_{E,i} > 1)\right)$$

Pour ce faire, nous devons prouver :

$$S \vDash\!\downarrow \left(\bigwedge_{i=1}^{m}(S_{E,i} \leqslant 1)\right) \tag{5.6}$$

$$S \vDash \circlearrowleft \left(\bigwedge_{i=1}^{m}(S_{E,i} \leqslant 1)\right) \tag{5.7}$$

Comme indiqué en section 5.3, (5.7) est déjà prouvée. Pour la preuve que S est convergent en $(\neg TC_C)$, considérons le variant $V = S_E$. Par définition, aucune action du type d'agent consommateur ne peut faire augmenter le variant. De plus, dans le cadre où $S_E > 1$) est vrai, l'action a_3 du type consommateur et, comme indiqué précédemment, nous avons déjà prouvé la terminaison du type producteur donc cela suffit pour prouver la terminaison du type consommateur.

Nous avons maintenant prouvé que $S \vDash (\Diamond\Box\neg TC_P) \wedge (\Diamond\Box\neg TC_C)$, ce qui implique $S \vDash (\Diamond\Box\neg TC_P \wedge \neg TC_C)$. Par conséquent, le SMAP finit par se terminer.

Nous avons montré que pour les propriétés de vivacité également, le nombre de preuve pouvait, dans certains cas, ne pas dépendre du nombre d'agents, ce qui permet de montrer des comportements portant sur les types d'agents et non sur les agents eux-mêmes.

6. Conclusion et perspectives

Nous avons étendu le modèle GDT4MAS avec des spécifications probabilistes. Il est désormais possible de spécifier des actions probabilistes, des choix probabilistes entre les actions et de déterminer de manière probabiliste l'ordre d'activation des agents. Pour maintenir la correction du modèle, nous en avons adapté les schémas de preuve et introduit de nouveaux invariants pour chaque agent sans changer la structure du modèle. Ainsi, il est possible de continuer d'utiliser les méthodes, prouveurs et implantation actuels. De plus, nous avons adapté une méthode de variant afin de pouvoir prouver certaines propriétés de vivacité. Les travaux futurs consisteront à utiliser ce modèle

afin de prouver formellement des propriétés telles que « la probabilité d'une formule donnée est égale à une quantité donnée », comme cela se fait en *model checking*. Une piste pertinente pour cela est de considérer une sémantique CTL* afin de conserver l'expressivité de LTL tout en intégrant la notion de branchement. De plus, de manière plus générale, nous voudrions considérer des systèmes ouverts, avec apparition ou disparition d'agents, des dépendances aux conditions initiales ainsi que la preuve de propriétés émergentes. Cela nécessitera de préciser plus en détails la manière dont la théorie des ensembles est implantée dans le modèle pour que chaque type d'agent soit représenté par un ensemble. Concernant la preuve de propriétés émergentes, il sera intéressant de considérer des exemples connus de la communauté multi-agent afin de pouvoir vérifier s'il est possible de prouver l'apparition de ces propriétés et peut-être également d'envisager pouvoir aider à leur prédiction. Plus généralement, il serait également intéressant de considérer des opérateurs des GDT ayant un effet sur les GDTs eux-même, tout comme cela a été fait en Event-B avec le module EB4EB. En effet, ce module permet de concevoir les machines comme des variables même du système, et cela nous permettrait de pouvoir spécifier de manière plus simple des comportements complexes et des adaptations réflexives des agents sur leur comportement.

Bibliographie

[1] J.-R. Abrial, A. Hoare & P. Chapron, *The B-Book*, Cambridge University Press, 1996.

[2] M. Aouadhi, « A fully probabilistic extension of Event-B », Tech. report, LINA - Univ. Nantes, 2016.

[3] ———, « Introduction de raisonnement probabiliste dans la méthode B événementiel », Thèse, Université de Nantes, 2017.

[4] C. Baier & M. Kwiatkowska, « Automatic verification of liveness properties of randomized systems », in *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing, Santa Barbara, California, USA, August 21-24, 1997* (J. E. Burns & H. Attiya, éds.), ACM, 1997.

[5] C. Baier & M. Kwiatkowska, « Model checking for a probabilistic branching time logic with fairness », *Distributed Computing* **11** (2004), p. 125-155.

[6] M. Esteva, D. de la Cruz & C. Sierra, « ISLANDER: An electronic institutions editor », in *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 3*, AAMAS '02, ACM, 2002, p. 1045-1052.

[7] V. Forejt, M. Kwiatkowska, G. Norman & D. Parker, « Automated Verification Techniques for Probabilistic Systems », in *Formal Methods for Eternal Networked Software Systems: 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM 2011, Bertinoro, Italy, June 13-18, 2011. Advanced Lectures* (M. Bernardo & V. Issarny, éds.), Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, p. 53-113.

[8] Z. Graja, F. Migeon, C. Maurel, M. Gleizes & A. H. Kacem, « Vers une modélisation formelle basée sur le raffinement des systèmes multi-agents auto-organisateurs », in *Principe de Parcimonie – JFSMA 14 – Vingt-deuxièmes Journées Francophones sur les Systèmes Multi-Agents, Loriol-sur-Drôme, France, Octobre 8-10, 2014* (R. Courdier & J. Jamont, éds.), Cepadues Editions, 2014, p. 139-148.

[9] ———, « Vers une modélisation formelle basée sur le raffinement des systèmes multi-agents auto-organisateurs », *RIA* **30** (2016), n° 1-2, p. 159-183.

[10] T. S. Hoang, « The development of a probabilistic B-method and a supporting toolkit », Thèse, University of New South Wales, 2006.

[11] T. S. Hoang & J.-R. Abrial, « Reasoning about Liveness Properties in Event-B », in *Formal Methods and Software Engineering* (S. Qin & Z. Qiu, éds.), Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, p. 456-471.

[12] S. Hudon, T. S. Hoang & J. S. Ostroff, « The Unit-B method: Refinement guided by progress concerns », *SoSyM* **15** (2016), p. 1091-1116.

[13] J. G. Kemeny, J. L. Snell & A. W. Knapp, *Denumerable Markov chains*, 2nd éd., Grad. Texts Math., vol. 40, Springer, Cham, 1976.

[14] M. Kwiatkowska, G. Norman & D. Parker, « PRISM 4.0: Verification of Probabilistic Real-Time Systems », in *Computer Aided Verification* (G. Gopalakrishnan & S. Qadeer, éds.), Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, p. 585-591.

[15] M. Kwiatkowska, G. Norman, D. Parker & G. Santos, « Equilibria-Based Probabilistic Model Checking for Concurrent Stochastic Games », in *Formal methods – the next 30 years. Third world congress/23rd symposium, FM 2019, Porto, Portugal, October 7–11, 2019. Proceedings* (M. H. ter Beek, A. McIver & J. N. Oliveira, éds.), Lect. Notes Comput. Sci., vol. 11800, Springer International Publishing, Cham, 2019, p. 298-315.

[16] ———, « Automatic verification of concurrent stochastic systems », *Form. Methods Syst. Des.* **58** (2021), nº 1-2, p. 188-250.

[17] A. Lomuscio & E. Pirovano, « Parameterised Verification of Strategic Properties in Probabilistic Multi-Agent Systems », AAMAS '20, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2020, p. 762-770.

[18] A. Lomuscio, H. Qu & F. Raimondi, « MCMAS: An opensource model checker for the verification of multi-agent systems », *Int J Softw Tools Technol Transfer* **19** (2017), p. 9-30.

[19] A. Lomuscio & F. Raimondi, « MCMAS: a model checker for multi-agent systems », in *Tools and algorithms for the construction and analysis of systems. 12th international conference, TACAS 2006, held as part of the joint European conferences on theory and practice of software, ETAPS 2006, Vienna, Austria, March 25 – April 2, 2006.*, Springer, Berlin, 2006, p. 450-454.

[20] A. McIver & C. Morgan, *Abstraction, refinement and proof for probabilistic systems*, Springer, New York, NY, 2005.

[21] B. Mermet & G. Simon, « GDT4MAS: an extension of the GDT model to specify and to verify MultiAgent systems », in *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems – Volume 1*, AAMAS '09, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2009, p. 505-512.

[22] B. Mermet & G. Simon, « A New Proof System to Verify GDT Agents », in *Intelligent Distributed Computing VII* (F. Zavoral, J. J. Jung & C. Badica, éds.), Springer International Publishing, Cham, 2014, p. 181-187.

[23] S. Owre, J. M. Rushby & N. Shankar, « PVS: A prototype verification system », in *Automated Deduction—CADE-11* (D. Kapur, éd.), Springer Berlin Heidelberg, Berlin, Heidelberg, 1992, p. 748-752.

[24] E. Pirovano, « Parameterised model-checking of probabilistic multi-agent systems », Thèse, Imperial College London, 2021.

[25] P. Schnoebelen, « The complexity of temporal logic model checking », in *Advances in modal logic. Vol. 4. Selected papers from the 4th conference (AiML 2002), Toulouse, France, October 2002*, King's College Publications, London, 2003, p. 393-436.

[26] K. Stathis, A. Kakas, W. Lu, N. emetriou, U. Endriss & A. Bracciali, « PROSOCS: a platform for programming software agents in computational logic », in *Proceedings of the Fourth International Symposium "From Agent Th eory to Agent Implementation" (AT2AI-4 – EMCSR'2004 Session M)* (J. Müller & P. Petta, éds.), 2004, p. 523-528.

[27] M. Y. Vardi, « Automatic verification of probabilistic concurrent finite state programs », in *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, 1985, p. 327-338.

[28] ———, « Branching vs. linear time: Final showdown », in *Tools and algorithms for the construction and analysis of systems. 7th international conference, TACAS 2001, held as part of the joint European conferences on theory and practice of software, ETAPS 2001, Genova, Italy, April 2–6, 2001. Proceedings*, Springer, Berlin, 2001, p. 1-22.

Abstract. — This article deals with the formal proof of multi-agent systems whose behaviors can be characterized probabilistically. We consider the GDT4MAS model which addresses formal verification from specification to proof obligation generation by relying on first-order logic, which gives it an important expressiveness. However, the GDT4MAS model does not deal with stochastic behavior and does not allow the proof of liveness properties. We then extend GDT4MAS to model agents that can perform actions with stochastic effects and redefine the operators in this framework. To prove liveness properties, we rely on variant methods adapted to our framework and show how to prove the MAS almost sure termination, or the recurrence of a property.

Keywords. — Formal verification, multi-agent systems, stochastic behaviours.