Vérification formelle de propriétés de vivacité pour des SMA stochastiques à l'aide de GDT

Mathias Déhais[a] mathias.dehais@unicaen.fr
Bruno Mermet[b] bruno.mermet@unicaen.fr
Grégory Bonnet[a] gregory.bonnet@unicaen.fr

[a]Normandie Univ, UNICAEN, ENSICAEN, CNRS, GREYC

[b]Normandie Univ, UNIHAVRE, ENSICAEN, CNRS, GREYC

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*

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.

Mots-clés : *Formal verification, Multi-agent systems, Stochastic behaviours*

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 stochastiques (SMAS), 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 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	non
Création / destruction d'agents	non

TABLE 1 – Sources de stochasticité potentielles considérées dans cet article.

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 égale-

ment 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.* [17, 16] 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* pose des problèmes de recherche exhaustive. 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 [7] utilisent des solutions ad hoc et il y existe peu d'approches différentes, dont les principales sont données ci-après. Esteva *et al.* [5] 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.* [23] ont proposé PROSOCS, qui spécifie les aspects internes des agents et utilise un modèle d'agence KGP (Knowledge, Goals, Plan). Mermet et Simon [19] 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. De plus, 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. Par conséquent, dans cet article, nous nous concentrons sur l'extension de GDT4MAS aux comportements stochastiques et à la preuve de propriétés de vivacité.

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.* [12] ont développé, et étudient toujours [14], 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 PCL (*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 [13]. Le lecteur intéressé peut se référer à [6]. 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 [18] 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 [8] 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 [18] 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 SMAS

Les travaux précédents ont été adaptés aux SMA, l'aspect *model checking* est donc bien développé depuis plusieurs dizaines d'années, [4] et la vérification de propriétés de vivacité est également un sujet de recherche depuis de nombreuses années, [14]. Récemment, Lomuscio et Pirovano [15] ont traité de la vérification formelle pour SMAS en raisonnant sur un fragment de la logique PCTL. Pirovano [22] a amélioré la méthode afin de travailler sur le problème du passage à l'échelle. *Model checking* et preuve de théorèmes ont toujours avancé côte à côte, partageant les mêmes objectifs avec des outils distincts. Bien que des avancées aient été faites, le passage à l'échelle reste toujours un problème en *model checking*, c'est pourquoi il nous parait 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 [9] 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 [10], pour tenter d'instaurer les preuves de propriétés de vivacité au coeur du modèle.

3 Introduction à GDT4MAS

Étant donnés les éléments précédents, nous choisissons d'étendre le modèle GDT4MAS avec des considérations probabilistes. Nous présentons d'abord des éléments sur le modèle général. Nous renvoyons les lecteurs intéressés vers [19] pour plus de détails. Comme dans des 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.

3.1 Logique temporelle sous-jacente

GDT4MAS utilise la logique temporelle linéaire (LTL) pour donner une sémantique opérationnelle aux GDT. Étant donné un ensemble de variables booléennes V, les formules LTL sur V sont les formules construites à partir des variables de V, les connecteurs propositionnels usuels $(\neg, \wedge, \vee, \rightarrow)$ et les opérateurs standards $\Box$, $\Diamond$ et $\bigcirc$. Les formules LTL sont évaluées par rapport aux *traces* qui sont des séquences infinies de mondes $(\omega_1, \omega_2, \omega_3, \dots)$, où chaque monde est étiqueté avec une interprétation des variables. Étant donné un monde ω, $Path(\omega)$ est l'ensemble de toutes les traces possibles commençant par le monde ω. La validité d'une formule LTL ϕ est évaluée par rapport à une trace τ et un monde ω dans celle-ci, selon la sémantique standard, notée $\tau, \omega \vDash \phi$. Soit ω un monde et ψ une formule LTL, alors $\omega \vDash \psi$ si et seulement si $\tau, \omega \vDash \psi$ pour tous les $\tau \in Path(\omega)$.

3.2 Agent et arbre de décomposition de buts

Dans GDT4MAS, le comportement des agents est représenté par des GDT. Ce 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 agent spécifie l'ensemble de son comportement, et la condition de satisfaction de son nœud racine est donc son objectif principal. De manière intéressante, il

est possible de spécifier plusieurs agents ayant le même comportement avec un même GDT, ce qui permet de vérifier tous les agents avec les mêmes preuves. Nous ne donnons ci-dessous que les notions pertinentes pour cet article. La logique des prédicats est utilisée dans la phase de spécification essentiellement pour exprimer les conditions de satisfaction des objectifs, les conditions préalables des actions et les conditions de branchement. Nous notons cette logique $\mathcal{L}$. La phase de conception nécessite de pouvoir exprimer des objectifs dont le succès dépend de l'évolution de certains prédicats, comme incrémenter une valeur. Pour cela, nous utilisons une extension, notée $\mathcal{L}'$, de la logique choisie 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.

Définition 1 (Environnement)**.** Un *environnement* est un tuple $\epsilon = (V_\epsilon, i_\epsilon)$, où V_ϵ est un ensemble de variables, et $i_\epsilon \in \mathcal{L}$ désigne les invariants de l'environnement.

Les agents interagissent avec l'environnement en modifiant les variables d'environnement. Les invariants d'environnement sont des formules sur ces variables dont il faut prouver qu'elles restent vraies tout au long de l'exécution afin de vérifier que la spécification est correcte.

Définition 2 (Agent)**.** Soit ϵ un environnement. Un *agent* A relatif à ϵ est un tuple :

$$(V_i(A), V_\epsilon(A), init_A, i_A, Actions_A, Beh_A)$$

Chaque agent est décrit grâce à un ensemble de variables internes $V_i(A)$, un ensemble de variables d'environnement $V_\epsilon(A)$ sur lesquelles il peut agir, et un ensemble d'actions $Actions_A$ qu'il peut effectuer. $init_A$ correspond à une fonction qui associe une valeur initiale aux variables internes. i_A est les invariants de l'agent que nous voulons vérifier. Un comportement Beh_A modélisé par un GDT décrit alors son comportement.

Définition 3 (GDT)**.** Soit ϵ un environnement et A un agent. Un *GDT* pour A dans ϵ est un triplet $(pre_T, tc_T, Root_T)$, où $pre_T, tc_T \in \mathcal{L}$ et $Root_T$ est un nœud interne (donc un arbre). pre_T est la condition préalable et t_{C_T} est le contexte de déclenchement, ce qui signifie que le GDT ne peut s'activer que s'il est vrai.

Les GDT sont constitués de feuilles et de nœuds internes. Dans un GDT, les nœuds correspondent aux buts de l'agent. 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. La SC du nœud à la racine est alors le but principal de l'agent, décomposé grâce à des opérateurs en sous-buts. Enfin, toujours 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).

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 $\{init, in, end\}$ dénotent respectivement que l'on commence à exécuter N, que l'on est en train d'exécuter N et que l'on a fini de l'exécuter.

Définition 4 (nœud interne)**.** Soit A un agent dans un environnement ϵ. Un *nœud interne* N (d'un GDT pour A dans ϵ) est un tuple :

$$(name_N, Op_N, Children, sc_N, gpf_N)$$

où $sc_N,\ gpf_N \in \mathcal{L}'$, 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. Les *nœuds feuilles* se comportent comme des nœuds internes sauf qu'ils contiennent une *action* et pas d'opérateur de branchement.

4 GDT4MAS probabiliste

Pour le modèle probabiliste, il est nécessaire d'étendre la logique LTL avec un opérateur probabiliste.

4.1 LTL 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 $\sim \in \{<, \leq, \geq, >, =\}$ 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 SMAS. 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 ω, puis étendue de manière unique à toutes les traces commençant par ω. Pour plus de détails, le lecteur peut se référer à [11]. Ainsi, étant donné un monde ω, une formule LTL ψ, $\lambda \in [0, 1]$, nous avons :

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

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

Exemple 1. Soit un SMAS 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 $0,5$.

Dans la version probabiliste de GDT4MAS, nous rajoutons les opérateurs probabilistes $P_{\texttt{OR}}$ et $P_{\texttt{AND}}$, qui sont des versions probabilistes des opérateurs `OR` et `AND`.

Définition 5 (Opérateur probabiliste `OR`)**.** L'opérateur *probabiliste `OR`*, noté $P_{\texttt{OR}}$, décompose un but en k sous-buts, les exécutions sont choisies de manière probabiliste jusqu'à ce que l'un d'eux réussisse. L'ordre est donné par un ensemble de probabilités P_{op}. À chaque sous-nœud N_i une probabilité $P_{N_i} \in P_{op}$ est associée et il faudra prouver que la somme des P_{N_i} vaut 1. Si toutes les exécutions échouent, l'exécution de l'objectif 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. $\Box(\forall(i \in [\![1, k]\!]).$
   $(end_{N_i} \wedge \neg sat_{N_i} \wedge \exists(j \in [\![1, k]\!]).\neg done_{N_j})$
   $\rightarrow \mathcal{P}_{=P_j}(\bigcirc_A(init_{N_j})))$

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

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

La formule 1 signifie que chaque sous-noeud 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-objectif, la probabilité qu'un nœud donné N_i soit exécuté vaut P_j, sa probabilité initiale divisée par la somme des probabilités des sous-noeuds non-exécutés.

L'opérateur $P_{\texttt{AND}}$ est défini de manière parfaitement similaire 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.

Définition 6 (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 ait lieu. $POST_a = \{post_{a,i} \,|\, i \in [\![1, n]\!]\}$ avec $n \in \mathbb{N}$ étant le nombre de réussites. Chaque $post_{ai}$ représente une formule qui devient potentiellement vraie si l'action réussit. $GPF_a = \{gpf_{a_i} \,|\, i \in [\![1, k]\!]\}$ avec $k \in \mathbb{N}$ étant le nombre de résultats infructueux. Chaque gpf_{ai} représente une formule qui peut devenir fausse si l'action échoue. À chaque résultat est assignée une fonction, qui est contenue dans l'ensemble $P_a = \{p_{post_{a,i}} \,|\, i \in [\![1, n]\!]\} \bigcup \{p_{gpf_{a,i}} | i \in [\![1, k]\!]\}$ où n et k sont tels que définis précédemment. Notons par la suite $OUTCOME_a$ l'union de $POST_a$ et GPF_a. Notons F_A l'ensemble des applications de $V_i(A)$ vers les valeurs. Chaque $p_{post_{a,i}}$ et $p_{gpf_{a,i}}$ est une fonction de F_A dans $\mathbb{R}$, ce qui signifie que la probabilité de chaque résultat dépend de la valeur 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 1. 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.

Remarque 2. 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.

Afin de vérifier des propriétés de vivacité, nous aurons besoin de déterminer si la probabilité qu'une action s'exécute une fois son GDT activé soit strictement positive.

Définition 7 (Action accessible)**.** Une action a est dite *accessible* si la probabilité d'exécuter le nœud feuille correspondant est non nulle. Notons N le nœud feuille associé à a et tc, le contexte déclencheur du GDT contenant N. L'action a est accessible si, et seulement si, $\Box(tc \Rightarrow \mathcal{P}_{>0}(\Diamond init_N))$.

Nous pouvons désormais définir un système multi-agents stochastique.

Définition 8 (Système multi-agents stochastique)**.** Un *système multi-agents stochastique* est un système multi-agents dont les agents peuvent avoir des comportements décrits par des actions probabilistes et des opérateurs probabilistes.

Remarque 3. Un GDT peut contenir à la fois des actions ou des opérateurs probabilistes et non déterministes comme dans le modèle de base.

Prenons en exemple un SMAS avec deux types d'agents, un producteur et un consommateur.

Variables. Soit $S_E \in \mathbb{N}$, une variable d'environnement qui représente le stock consommé par le *Consommateur*. Soit $S_{pro} \in \mathbb{N}$, une variable interne du *Producteur* qui représente son stock. Soit $S_{cons} \in \mathbb{N}$, une variable interne du *Consommateur* qui représente son stock.

Contextes de déclenchement.

$$tc_{pro} = (S_{pro} > 0) \text{ et } tc_{cons} = (S_E > 1)$$

Comportements. Les GDT des deux agents sont illustrés en figure 1. Ici, (a_1, a_2, a_3, a_4) font référence aux actions sur les nœuds feuilles. Les détails des actions sont donnés sur la table 2.

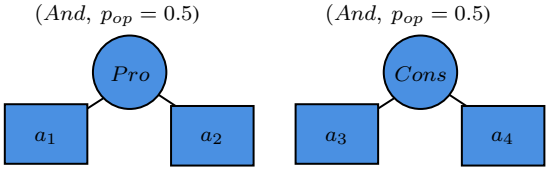

FIGURE 1 – GDT Producteur / Consommateur

4.2 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

TABLE 2 – Actions des deux agents

a_1	$post_1 = (S'_E = S_E + 1)$
a_2	$post_1 = (S'_{pro} = S_{pro} - 1)$ $p_{post1} = \frac{2}{3}$ $gpf_1 = (S'_{pro} = S_{pro})$ $p_{gpf1} = \frac{1}{3}$
a_3	$pre = S_E > 0$ $post_1 = (S'_E = S_E - 2)$ $p_{post1} = \frac{S_E}{1+S_E}$ $post_2 = (S'_E = S_E - 1)$ $p_{post_2} = \frac{1}{1+S_E}$
a_4	$post_1 = (S'_{cons} = S_{cons} + 1)$

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 [21].

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 9 (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 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 [20] 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 stochastique 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 3.

TABLE 3 – Invariants relatifs aux probabilités

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

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

$$i_\epsilon \wedge i_A \wedge c_N \wedge outcome'_a \vDash i'_\epsilon \quad (1)$$

$$i_\epsilon \wedge i_A \wedge c_N \wedge outcome'_a \vDash i'_A \quad (2)$$

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. Ceci est capturé par des schémas de preuve de nœuds feuilles définis comme suit :

Définition 11 (Schémas de preuve de nœuds feuille)**.**

$$i_\epsilon \wedge i_A \wedge c_N \vDash pre_a \quad (3)$$

Pour chaque $post_a \in POST_a$,

$$i_\epsilon \wedge i_A \wedge c_N \wedge post'_a \vDash sc'_N \quad (4)$$

Pour chaque $gpf_a \in GPF_a$,

$$i_\epsilon \wedge i_A \wedge c_N \wedge gpf'_a \vDash gpf'_N \vee sc'_N \quad (5)$$

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.

Reprenons l'exemple du SMAS producteur/consommateur donné en section 4.

Preuve de probabilités. Il nous faut montrer que les probabilités somment bien à 1 pour chaque action et restent comprises entre 0 et 1. Il est facile de vérifier que les probabilités somment toujours à 1 dans les deux types d'agents. Pour a_3, nous utilisons pre pour affirmer que les probabilités restent entre 0 et 1 et cela est trivial pour a_2. Pour des raisons de place, nous ne détaillons pas ici les PO instanciées.

Nous avons décrit le modèle GDT4MAS adapté aux systèmes multi-agents stochastiques. 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 parti du modèle probabiliste pour permettre la preuve de propriétés de vivacité grâce à la notion d'action accessible. Nous proposons une approche similaire à celle Hoang et Abrial [9] 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 [9] à notre modèle. Nous utilisons toujours la logique LTL, mais cette fois, les éléments de base du langage sont les propriétés $P \in \mathcal{L}$ puisque les propriétés que nous considérons ici reposent sur les variables des systèmes et non sur l'ensemble $\{in, sat, end, \dots\}$ présenté de manière informelle en définition 3.2. Ces propriétés sont utilisées comme des formules d'état. 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 12. Un SMAS 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 1. Une exécution d'un GDT consiste toujours en un nombre fini d'actions

Démonstration. (Esquisse). Tous les opérateurs du modèle, même ceux que nous n'avons pas introduits dans cet article, ne permettent de revenir sur un même nœud qu'un nombre de fois fini. Pour plus d'informations, nous renvoyons le lecteur à [19]. □

Ensuite, nous devons définir (pour pouvoir le prouver) le fait qu'un système mène d'un P_1-monde à un P_2-monde, avec $P_1, P_2 \in \mathcal{L}$.

Définition 13 (Mener à)**.** Un SMAS 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 si le SMAS 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 (toujours 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 14 (Convergence en P)**.** Un SMAS 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 méthode de variant, c'est-à-dire :

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

$$pre_a \wedge C_a \wedge P \Rightarrow V \in \mathbb{N} \quad (6)$$

  et pour chaque $outcome_a \in OUTCOME_a$:

$$pre_a \wedge C_a \wedge outcome'_a \wedge P \Rightarrow V' \leq V \quad (7)$$

- Prouver qu'il existe au moins une action accessible dans chaque GDT pour laquelle, dans un monde ¬P :

$$pre_a \wedge C_a \wedge (post'_a \vee gpf'_a) \wedge \neg P \Rightarrow V' < V \quad (8)$$

Proposition 1. *La méthode de la définition 14 prouve la convergence d'un SMAS 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 accessible par GDT qui fait diminuer le variant (voir condition 8 ci-dessus). Comme l'exécution de chaque GDT est finie, d'après la propriété 1, 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 7 ci-dessus), il est infiniment diminué de manière presque sûre. Cependant, puisque V est un entier 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 15 (Sans blocage)**.** Un SMAS S est sans blocage en P si toute trace finie de S ne se termine pas en un P-monde.

Pour prouver une telle propriété, nous écrivons que si tous les contextes C_N de tous les nœuds feuilles sont faux, alors le SMAS 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 2. Soit S un SMAS, avec k types d'agents, alors si $P \Rightarrow \bigvee_i (tc_i)$, avec tc_i étant les contextes de déclenchement alors S est sans blocage en P.

Démonstration. Si un seul contexte de déclenchement est vrai, il suffit de s'assurer que le SMAS 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 4. 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 ».

Règles pour les propriétés de vivacité. Nous adaptons désormais les méthodes proposées par Hoang et Abrial aux GDT4MAS probabilistes. Nous pouvons maintenant appliquer les règles pour prouver les propriétés de vivacité comme la *persistance*, c'est-à-dire que quelque chose doit finalement rester vrai pour toujours. Nous faisons pour cela référence à la règle développée par Hoang et Abrial donnée en table 4.

TABLE 4 – Règle pour la propriété d'apparition

$$\frac{S \vdash_{\downarrow} P \quad S \vdash_{\circlearrowleft} \neg P}{S \vdash \Diamond\Box P}$$

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 [9] 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 14 et en propriété 2.

5.2 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 SMAS. 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 SMAS 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 2. *Soit S un SMAS qui possède k types d'agents, c'est-à-dire qu'il y a k GDT différents. Notons $tc_1, \ldots, tc_k$ le contexte de déclenchement des différents GDT. S se termine finalement si, et seulement si, $S \vDash \Diamond\Box(\neg tc_1 \wedge \cdots \wedge \neg tc_1)$.*

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

Démonstration. ($\Rightarrow$) Supposons que le SMAS se termine dans le futur alors il existe un monde ω dans lequel les contextes de déclenchement sont faux. ($\Leftarrow$) Supposons que $S \vDash \Diamond\Box(\neg tc_1 \wedge \cdots \wedge \neg tc_1)$. L'exécution d'un GDT est finie selon la propriété 1. 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 $S \vDash \Diamond\Box(\neg tc_1 \wedge \cdots \wedge \neg tc_1)$ nécessite d'écrire un variant unique sur tous les GDT, ce qui n'est pas très pratique. Utilisons alors le fait que $\Diamond\Box\neg tc_1 \wedge \cdots \wedge \Diamond\Box\neg tc_k$ est équivalent à $\Diamond\Box(\neg tc_1 \wedge \cdots \wedge \neg tc_1)$, puis prouvons chacun d'eux séparément. Selon la règle de la table 4, pour prouver $\Diamond\Box\neg tc$ il faut d'abord prouver $S \vdash_{\downarrow} \neg tc$ en définissant un variant. Ensuite, il faut prouver $S \vdash_{\circlearrowleft} tc$, et cela est toujours trivial puisqu'il s'agit de la définition d'un SMAS sans blocage. En effet, si un contexte de déclenchement est vrai, le SMAS ne peut pas être bloqué. Il ne reste que la première propriété à prouver.

Remarque 5. Si nous montrons qu'un GDT finit par se terminer, alors il est certain qu'il existe un monde dans lequel le GDT est définitivement terminé. Ainsi, nous pouvons ne pas considérer les actions du GDT pour le reste de la preuve, et donc la simplifier.

Remarque 6. Si un variant ne contient que des variables internes d'un agent, seules ses actions peuvent être prises en compte pour la preuve de convergence. En effet, nous pouvons supposer que cet 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 ne peut modifier le variant.

Reprenons l'exemple du SMAS producteur/consommateur donné en section 4.

Preuve de terminaison. Nous voulons prouver que le SMAS finit par se terminer. Pour ce faire, nous devons prouver que $S \vDash \Diamond\Box\neg tc_{pro}$ et que $S \vDash \Diamond\Box\neg tc_{cons}$.

Concentrons-nous sur le *Producteur*. Nous voulons prouver $S \vDash \Diamond\Box\neg(S_{pro} \leq -1)$. Pour ce faire, nous devons prouver :

$$S \vDash_{\downarrow} (S_{pro} \leq -1) \tag{9}$$

$$S \vDash_{\circlearrowleft} (S_{pro} > -1) \tag{10}$$

Comme indiqué en section 5.2, (10) est déjà prouvée. Pour la preuve que S est convergent en $(\neg tc_{pro})$, considérons le variant $V = S_{pro}$. Nous devons prouver qu'aucune action du *Producteur* n'augmente le variant, ce qui est vrai par définition. Nous devons alors montrer qu'au moins une action du *Producteur* diminue le variant, ce qui est vrai pour l'action a_2.

Nous avons prouvé que $S \vdash \Diamond\Box\neg tc_{pro}$. Prouvons maintenant $S \vdash \Diamond\Box\neg tc_{cons}$. 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. Par contrainte de place, nous ne présentons pas ici les détails mais il s'agit de la même structure de preuve que précédemment où S_E remplace S_{pro}. Nous devons alors montrer qu'au moins une action diminue le variant, ce qui est vrai pour a_3.

Nous avons maintenant prouvé que $S \models (\Diamond\Box\neg tc_{c1}) \wedge (\Diamond\Box\neg tc_{c2})$ ce qui implique $S \models \Diamond\Box\neg tc_{c1} \wedge \neg tc_{c2}$. Par conséquent, le SMAS finit par se terminer.

6 Conclusion et perspectives

Nous avons étendu le modèle GDT4MAS avec des spécifications probabilistes. Pour maintenir la correction du modèle, nous en avons adapté les schémas de preuve et introduit de nouveaux invariants pour chaque agent. 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*. 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.

Références

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

[2] M. Aouadhi, A fully probabilistic extension of Event-B, Technical Report, 2016.

[3] M. Aouadhi, *Introduction de raisonnement probabiliste dans la méthode B événementiel*, PhD. thesis, 2017.

[4] C. Baier and M. Kwiatkowska, Automatic verification of liveness properties of randomized systems, *PODC*, pp. 295, 1997.

[5] M. Esteva, D. Cruz, and C. Sierra, ISLANDER : An electronic institutions editor, *AAMAS*, pp. 1045–1052, 2002.

[6] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker, Automated verification techniques for probabilistic systems, *SFM*, pp. 53–113, 2011.

[7] Z. Graja, F. Migeon, M.-P. Gleize et A. H. Kacem , Vers une modélisation formelle basée sur le raffinement des systèmes multi-agents auto-organisateurs, *JFSMA*, 2014.

[8] T.S. Hoang, *The development of a probabilistic B-method and a supporting toolkit*, PhD. thesis, 2006.

[9] T.S. Hoang and J-R. Abrial, Reasoning about liveness properties in Event-B, *ICFEM*, pp. 456–471, 2011.

[10] S. Hudon, T.S Hoang and J.S. Ostroff, The Unit-B Method, Refinement guided by Progress Concerns, *SoSyM*, pp.1091-1116,2016

[11] J.G. Kemeny, J.L. Snell, and A.W. Knapp, *Denumerable Markov chains*, Springer, 1969.

[12] M. Kwiatkowska, G.Norman, and D.Parker, PRISM 4.0, Verification of probabilistic real-time systems, *CAV*, vol. 6806, pp. 585–591, 2011.

[13] M. Kwiatkowska, G. Norman, D. Parker, and G. Santos, Equilibria-based probabilistic model checking forconcurrent stochastic games, *FM*, pp. 298–315, 2019.

[14] M. Kwiatkowska, G. Norman, D. Parker, and G. Santos, Automatic verification of concurrent stochastic systems, *Formal Methods in System Design*, vol. 58, pp. 188-250, 2021.

[15] A. Lomuscio and E. Pirovano, Parameterised verification of strategic properties in probabilistic multi-agent systems, *AAMAS*, pp. 762–770, 2020.

[16] A. Lomuscio, H. Qu, and F. Raimondi, MCMAS : An opensource model checker for the verification of multi-agent systems, *STTT*, vol. 19, pp. 9-30, 2017.

[17] A. Lomuscio, and F. Raimondi, MCMAS : A model checker for the verification of multi-agent systems, *TACAS*, pp. 450-454, 2006.

[18] A. McIver and C. Morgan, *Abstraction, refinement and proof for probabilistic systems*, Springer, 2005.

[19] B. Mermet and G. Simon, GDT4MAS : An extension of the GDT model to specify and to verify multiagent systems, *AAMAS*, pp. 505–512, 2000.

[20] B. Mermet and G. Simon, A new proof system to verify GDT agents, *IDC*, pp. 181–187, 2013.

[21] S. Owre, J.M. Rushby, and N. Shankar. PVS : A prototype verification system, *CADE* vol. 607, pp. 748-752, 1992.

[22] E. Pirovano. *Parameterised model-checking of probabilistic multi-agent systems*, PhD. thesis, 2021

[23] K. Stathis, A.C. Kakas, W. Lu, N. Demetriou, U. Endriss, and A. Bracciali, PROSOCS : A platform for programming software agents in computational logic, *AT2AI*, pp. 523–528, 2004.