Vérification formelle et robots mobiles

Béatrice Bérard, Laure Millet, Maria Gradinariu Potop-Butucaru, Yann Thierry-Mieg, Sébastien Tixeuil

Vérification formelle et robots mobiles

Béatrice Bérard[1] et Laure Millet[1] et Maria Potop-Butucaru[1] et Yann Thierry-Mieg[1] et Sébastien Tixeuil[1]

[1]*LIP6,UPMC, 4 Place Jussieu, 75005 Paris, France*

Les tâches susceptibles d'être exécutées par des robots mobiles sont de plus en plus nombreuses et en complexité croissante. Jusqu'à présent, les réseaux de robots ont été étudiés de manière empirique et la plupart des résultats ont été validés principalement par des simulations ou des preuves partielles. Nous proposons la première méthode automatique de modélisation et de vérification formelle de tels systèmes. L'utilisation de méthodes automatiques de vérification permet de s'assurer du bon fonctionnement d'algorithmes quand la mise en place de preuves est très complexe notamment dans un environnement asynchrone.

Keywords: Robots mobiles, Exploration de graphes, Vérification, *Model-Checking*

1 Introduction

De nombreuses applications utilisent des groupes de robots qui s'auto-organisent pour résoudre un objectif commun, en l'absence d'agent de coordination.

Bien que ces robots évoluent dans un espace euclidien à deux dimensions, il est plus facile de représenter une version discrétisée de cet espace. La représentation consiste alors en un graphe où l'espace est partitionné, chaque noeud du graphe représentant un emplacement, et chaque arc représentant la possibilité pour un robot de se déplacer d'un noeud vers un autre.

Nous considérons un système distribué de k robots dont les capacités sont restreintes : ces robots sont anonymes et identiques, ils ne peuvent être distingués les uns des autres et exécutent le même protocole. Ils ne mémorisent pas les actions précédemment exécutées, n'ont aucun sens de l'orientation et ne peuvent communiquer entre eux. Cependant, ils ont la capacité d'observer leur environnement et de voir l'agencement des autres robots. Ils fonctionnent selon un cycle composé de trois phases : *Look*, *Compute* et *Move*. Pendant la première phase, les robots examinent le monde qui les entoure. Les informations ainsi collectées leur permettent de calculer, lors de la phase *Compute*, un futur mouvement qui sera rendu effectif dans la dernière phase. Ce modèle synchrone, appelé SYm (ou ATOM), a été introduit par Suzuki & Yamashita puis amélioré par Prencipe & al. en un modèle asynchrone, appelé CORDA, afin de prendre en compte le comportement asynchrone des systèmes distribués.

Parmis les nombreuses tâches qui ont été étudiées dans le cadre d'un environnement discret, nous nous intéressons à l'exploration perpétuelle d'un environnement inconnu : chaque noeud du graphe doit être visité infiniment souvent. Plus spécifiquement, nous étudions ici l'exploration d'un anneau.

De nombreux travaux traitent de l'exploration d'un anneau [FPS12]. Cependant, les preuves des résultats présentés ne sont pas automatiques, ce qui peut être source d'erreurs, en particulier dans le cadre asynchrone. Nous examinons ici l'article de Blin & al. [BMPT10], où les auteurs démontrent que trois robots déterministes sont nécessaires et suffisants pour explorer un anneau de taille $n \geq 10$, et que le nombre maximal de robots pour explorer un anneau est $k = n-5$ si k et n sont premiers entre eux. Les auteurs fournissent des algorithmes pour explorer un anneau avec des nombres minimaux et maximaux de robots.

Afin de vérifier automatiquement le bon fonctionnement d'algorithmes répartis, nous proposons la première méthode formelle de modélisation et de vérification de tels protocoles, en utilisant les algorithmes de [BMPT10] comme étude de cas. La modélisation appropriée du système et la formalisation du problème permettent l'utilisation du *model-checking* comme technique automatique de vérification. Cette technique a permis de mettre en évidence la correction de ces algorithmes dans un environnement synchrone et a révélé un contre-exemple dans un environnement asynchrone. Ceci a permis d'effectuer une correction éclairée.

2 Modélisation du système

Le système est modélisé par un ensemble fini d'automates, représentant le fonctionnement des robots dans les modèles d'exécution SYm et CORDA. Les composants du système sont les robots et les degrés d'asynchronie de l'environnement (les modèles d'exécution), ce dernier étant représenté par un *ordonnanceur*.

Modélisation des robots. On rappelle que les robots sont identiques et exécutent le même algorithme déterministe. Un même automate, représenté dans la Figure 1, peut donc décrire le comportement de chacun des robots. Le robot commence par observer son environnement (transition *Look*), puis il calcule son futur emplacement (transition *Compute*), et il finit par effectuer le mouvement pré-calculé (transition *Move*). En pratique, l'état *Ready to move* est divisé en autant d'états que de mouvements possibles, selon le protocole par lequel le robot calcule sa future position.

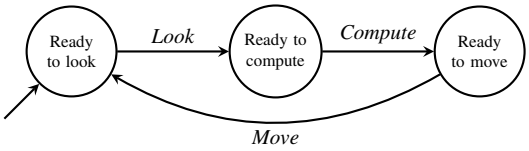


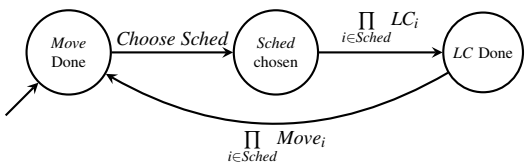
FIGURE 1: L'automate modélisant le comportement d'un robot

Dans ce modèle les contraintes de temps sont abstraites (temps de calcul, vitesse d'un robot). On suppose que les actions sont instantanées et que chaque cycle se termine en un temps fini. On s'intéresse tout particulièrement aux phases *Look* et *Move*, le calcul d'une future position pouvant être comprise comme une réinterprétation de la vue prise par un robot. La transition *Compute* a donc été combinée à son prédecesseur pour obtenir une transition *LC*.

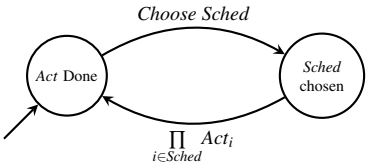
Modélisation de l'ordonnanceur. L'ordonnanceur choisit l'ordre avec lequel les robots vont exécuter leurs actions afin de respecter les modèles d'exécution SYm ou CORDA. Il est également modélisé par un automate fini. Contrairement aux robots qui ont toujours le même comportement, l'automate de l'ordonnanceur varie en fonction du modèle d'exécution choisi et du nombre de robots, comme le montre la Figure 2.

Notons *Rob* l'ensemble des robots du système. La phase *LC* (resp. *Move*) du *i*ème robot est représentée par l'action LC_i (resp. $Move_i$). Pour un sous-ensemble $Sched \subseteq Rob$, on définit :
$\prod_{i \in Sched} LC_i$ (respectivement $\prod_{i \in Sched} Move_i$) la synchronisation des actions LC_i (resp. $Move_i$).

Le modèle d'exécution SYm a deux variantes, SYm synchrone (tous les robots sont synchronisés à chaque phase) et SYm semi-synchrone (un sous-ensemble de robots est synchronisé à chaque phase). L'automate correspondant à l'ordonnanceur pour le modèle semi-synchrone est décrit dans la Figure 2a, il contient un cycle pendant lequel un sous-ensemble *Sched* des robots *Rob* est choisi afin d'être synchronisé sur les phases suivantes. L'état *Sched chosen* est divisé en 2^k états, afin de représenter tous les sous ensembles $Sched \subseteq Rob$. Pour représenter la variante totalement synchrone, le sous-ensemble choisi est toujours $Sched = Rob$. Avec cette variante tous les cycles sont identiques.



(a) SYm Semi-synchrone



(b) CORDA

FIGURE 2: Les automates des ordonnanceurs

Dans le modèle d'exécution CORDA, totalement asynchrone, il n'y a pas de règle d'ordonnancement, ce qui peut produire des exécutions où un robot change de position selon une observation qui n'est plus à jour.

L'ordonnanceur correspondant est décrit par l'automate de la Figure 2b. À chaque cycle, un sous ensemble $Sched \subseteq Rob$ est ordonnancé, puis chaque robot de ce sous-ensemble exécute la prochaine action selon son état local (Act_i peut être LC_i ou $Move_i$).

Modélisation du système. Le système est obtenu par un produit synchronisé entre les différents automates, auquel on ajoute les configurations des robots sur le graphe. Dans un graphe de n noeuds il y a $\binom{n}{k}$ configurations possibles, le nombre d'états du système résultant est donc multiplié par le nombre de configurations. De plus, le nombre de transitions partant d'un état dépend du nombre de mouvements réalisables sur la configuration de cet état.

On s'aperçoit rapidement que, même en choisissant un fort niveau d'abstraction, grâce auquel seules les spécificités comportementales des robots qui permettent de vérifier le protocole sont exprimées, l'automate du système est sujet à une forte explosion combinatoire.

3 Vérification et étude de cas

L'intérêt de notre démarche est illustré par la vérification du *Min-Algorithm* présenté dans [BMPT10]. Pour cette étude de cas, il faut extraire de la spécification du problème, puis formaliser, les propriétés qui doivent être satisfaites. Nous avons choisi la logique LTL pour cette formalisation.

Spécification du problème. Le problème traité dans [BMPT10] est *L'exploration perpétuelle exclusive* défini comme suit : pour tout graphe G de taille n et une configuration initiale où plusieurs robots n'occupent pas le même noeud, un protocole permet d'effectuer une exploration perpétuelle exclusive s'il satisfait les deux propriétés suivantes : (i) *exclusion*, plusieurs robots ne visiteront pas le même noeud et ne traverseront pas le même arc au même moment et (ii) *vivacité*, tous les noeuds du graphe seront visités infiniment souvent.

Dans les modèles d'exécution partiellement asynchrones, il est possible que l'ordonnanceur choisisse de ne jamais sélectionner un ou plusieurs voire tous les robots. Afin de palier ces cas où aucune progression n'est possible pour l'algorithme, une propriété d'équité doit être ajoutée. Cette propriété demande que tous les robots soit ordonnancés infiniment souvent. La propriété de vivacité ne peut être satisfaite que sur les exécutions satisfaisant la propriété d'équité.

Min-Algorithm. Les auteurs de cet algorithme garantissent qu'il permet à trois robots d'effectuer une exploration perpétuelle exclusive sur tout anneau de taille au moins 10, lorsque le nombre de robots et la taille de l'anneau sont premiers entre eux. L'algorithme repose sur une classification des configurations et une reconnaissance des classes de configurations. Il est présenté dans les tableaux 1 and 2, repris de l'article original. Une configuration est définie par une liste non orientée de symboles R et F indexés par des entiers : R_i représente i noeuds consécutifs occupés, et F_j représente j noeuds consécutifs libres.

Phase légitime : $z \neq \{0,1,2,3,4\}$			
$RL1::$	(R_2,F_2,R_1,F_z)	$\rightarrow$	$(R_1,F_1,R_1,F_2,R_1,F_{z-1})$
$RL2::$	$(R_1,F_1,R_1,F_2,R_1,F_z)$	$\rightarrow$	(R_2,F_3,R_1,F_z)
$RL3::$	(R_2,F_3,R_1,F_z)	$\rightarrow$	(R_2,F_2,R_1,F_{z+1})

TABLE 1: Règles de la phase légitime du *Min-Algorithm*

Vérification. L'implantation de l'algorithme s'est effectuée avec le langage DiVinE, en ajoutant des gardes sur les transitions, afin de restreindre les mouvements des robots. Pour chaque configuration observée, l'algorithme autorise certains robots à se déplacer selon des règles préétablies (l'ajout de gardes permettant de reconnaitre ces configurations).

Pour vérifier le bon fonctionnement de l'algorithme, nous avons utilisé le *model-checker* associé à DiVinE [BBvR10], avec une taille d'anneau égale à 10, qui est la plus petite taille d'anneau pour lequel cet

Phase de convergence :				
$RC1::$	(R_2,F_y,R_1,F_z)	$\rightarrow$	$(R_2,F_{min(y,z)},R_1,F_{max(y,z)+1})$	avec $y \neq z \neq \{1,2,3\}$
$RC2::$	$(R_1,F_x,R_1,F_y,R_1,F_y)$	$\rightarrow$	$(R_1,F_x,R_1,F_{y-1},R_1,F_{y+1})$	avec $x \neq y \neq 0$
$RC3::$	$(R_1,F_x,R_1,F_y,R_1,F_z)$	$\rightarrow$	$(R_1,F_{x-1},R_1,F_{y+1},R_1,F_z)$	avec $x < y < z$
$RC4::$	(R_3,F_z)	$\rightarrow$	(R_2,F_1,R_1,F_{z-1})	si 1 seul robot se déplace
		$\rightarrow$	$(R_1,F_1,R_1,F_1,R_1,F_{z-2})$	si 2 robots se déplacent
$RC5::$	(R_2,F_1,R_1,F_z)	$\rightarrow$	(R_2,F_2,R_1,F_{z-1})	

TABLE 2: Règles de la phase de convergence du *Min-Algorithm*

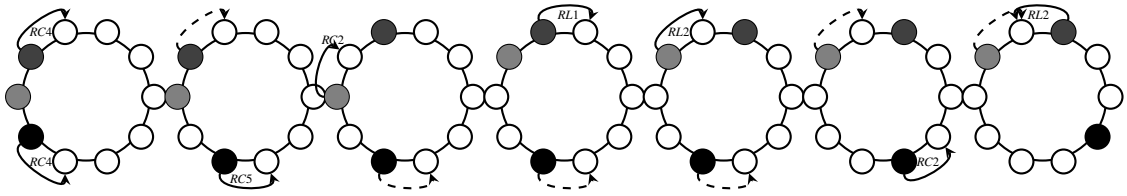


FIGURE 3: Contre-exemple

algorithme doit fonctionner. Alors que l'algorithme est correct lorsque les robots sont synchrones, DiVinE a produit un contre-exemple pour CORDA, parmi les 13.10^6 combinaisons de mouvements possibles. Ce contre exemple est une exécution aboutissant à une collision, comme le montre la Figure 3. Dans cette figure, chaque anneau représente une configuration. Un changement de configuration apparait lorsqu'un robot se déplace. Pour chaque configuration, une flèche pleine représente un calcul, une flèche en pointillé représente un calcul basé sur une vue obsolète.

Grâce à ce contre-exemple l'algorithme a pu être corrigé en modifiant la règle RC5 par :

$$RC5 \;::\; (R_2,F_1,R_1,F_z) \;\rightarrow\; (R_1,F_1,R_1,F_1,R_1,F_{z-1})$$

4 Conclusion

Nous avons proposé la première méthode de modélisation et de vérification formelle pour des réseaux de robots synchrones et asynchrones dans un espace discretisé. Nous avons mis en pratique cette technique en vérifiant les algorithmes proposés dans [BMPT10]. Nous avons prouvé que le Min-algorithm ne fonctionnait qu'avec des robots synchrones, alors qu'il était prouvé correct dans un modèle synchrone. Notre technique a permis de corriger cet algorithme pour qu'il fonctionne aussi dans le cadre asynchrone. À l'avenir, nous souhaiterions vérifier d'autres algorithmes (déterministes ou probabilistes), sur différents graphes, et pour différentes tâches. De plus, plutôt que de vérifier des stratégies existantes, il serait intéressant de considérer aussi un problème de synthèse, en cherchant à produire automatiquement une stratégie certifiée correcte.

Références

[BBvR10] J. Barnat, L. Brim, M. Češka, and P. Ročkai. DiVinE : Parallel Distributed Model Checker (Tool paper). In *Parallel and Distributed Methods in Verification and High Performance Computational Systems Biology (HiBi/PDMC 2010)*, pages 4–7. IEEE, 2010.

[BMPT10] L. Blin, A. Milani, M. Potop-Butucaru, and S. Tixeuil. Exclusive perpetual ring exploration without chirality. *Distributed Computing*, pages 312–327, 2010.

[FPS12] Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. *Distributed Computing by Oblivious Mobile Robots*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2012.