Manuel de savoir-prouver à l'usage des roboteux et des distributeux

Thibaut Balabonski, Pierre Courtieu, Robin Pelle, Lionel Rieg, Sébastien Tixeuil, Xavier Urbain

Manuel de savoir-prouver à l'usage des roboteux et des distributeux †

Thibaut Balabonsky[1] et Pierre Courtieu[2] et Robin Pelle[1] et Lionel Rieg[3] et Sebastien Tixeuil[4] et Xavier Urbain[5]

[1] *Université Paris-Sud, LRI, CNRS UMR 8623, Université Paris-Saclay*
[2] *Conservatoire national des arts et métiers,* CÉDRIC, *Paris*
[3] *Univ. Grenoble Alpes, CNRS, Grenoble INP,‡ VERIMAG, 38000 Grenoble, France*
[4] *Sorbonne Université, LIP6, CNRS* [5] *Université de Lyon, Université Claude Bernard Lyon 1, CNRS, LIRIS UMR 5205*

Les essaims de robots mobiles permettent de réaliser des tâches collaboratives complexes par assemblage de tâches plus simples accomplies individuellement par chaque robot. Néanmoins, leur nature massivement distribuée rend la preuve de correction d'un algorithme très difficile. De plus, il existe de nombreuses variantes du modèle en fonction des capacités de perception des robots, de l'espace et de sa topologie, des hypothèses de synchronisation, etc. Nous proposons Pactole, un cadre formel pour l'assistant à la preuve Coq, dans lequel tous ces modèles peuvent être simultanément exprimés. Nous illustrons ce formalisme sur plusieurs études de cas.

Mots-clefs : Méthodes formelles, Assistant à la preuve, Essaims de robots mobiles

1 Introduction

Dans les domaines mêlant criticité des applications et complexité des techniques à l'œuvre, les méthodes formelles contribuent à donner des garanties de sûreté et de correction. Ceci est particulièrement intéressant sur les systèmes distribués, toujours extraordinairement difficiles à vérifier. Dans le contexte des essaims de robots mobiles, les principales approches sont : le model-checking, la synthèse de protocole, et la preuve formelle, c'est-à-dire la preuve dont la correction est mécaniquement certifiée par un assistant à la preuve [PBSTU19]. Ces approches ont montré leur efficacité tant pour découvrir des erreurs dans des protocoles déjà publiés qu'en certifiant la correction d'algorithmes classiques ou originaux. Elles sont complémentaires : model-checking et synthèse offrent un haut niveau d'automatisation mais se limitent à des instances de problèmes ; la preuve formelle manque d'automatisation et requiert une expertise de l'utilisateur mais peut s'avérer plus générale en permettant par exemple de quantifier sur tous les protocoles (et donc de fournir des résultats d'impossibilité) ou de travailler quel que soit le nombre de robots ou la nature continue et infinie de l'espace. Nous présentons ici l'approche Pactole [§], un cadre pour obtenir des preuves formelles défini pour l'assistant à la preuve Coq.

À la différence du model-checking, développer un modèle de calcul distribué comme celui des essaims de robots dans un assistant de preuve consiste à définir *exactement* le modèle de calcul (ici, les modalités de déplacement des robots). Aucune abstraction n'est permise. Les définitions mathématiques écrites dans l'outil doivent représenter fidèlement le modèle de calcul lui-même. Il est bien sûr indispensable que des experts de l'algorithmique distribuée s'assurent que le modèle ainsi défini corresponde exactement à celui accepté par la communauté scientifique.

En général —et notre bibliothèque Pactole ne fait pas exception— la description en Coq d'un modèle de calcul consiste à écrire un *interprète* pour ce modèle : une fonction ou une relation qui décrit tous les comportements possibles du modèle *et uniquement ceux-ci*.

‡ Institute of Engineering Univ. Grenoble Alpes

† This work was partially funded by the CNRS PEPS INS2I project DiDASCaL and the Fédération Informatique de Lyon project COPRAH.

§. https://pactole.liris.cnrs.fr

Une fois le modèle défini et validé, les preuves faites en Coq sur l'interprète sont des résultats mathématiques au sens propre : ce sont des théorèmes sur le modèle au même titre —et même plus fiables— que s'ils étaient prouvés sur papier. Mieux : les preuves elles-mêmes n'ont pas besoin d'être contrôlées par un humain. Seuls les énoncés des théorèmes doivent être vérifiés [Bau13] : signifient-ils bien ce que prétendent les auteurs ?

2 Du modèle informel à son expression formelle

Nous considérons des ensembles de robots *oublieux*, c'est-à-dire sans mémoire des actions précédentes, qui évoluent dans un espace arbitraire, par exemple le plan euclidien ou encore un graphe orienté.

Modèle des robots. Les robots suivent le modèle fondateur de Suzuki et Yamashita [SY99]. Dans ce modèle, les robots sont tous identiques, anonymes, équipés du même algorithme de calcul (robogramme), sans mémoire ni volume et ils ne peuvent pas communiquer directement. Le comportement de chaque robot peut se diviser en *cycles* successifs de trois *phases* : *observation* (Look) où le robot acquiert des informations sur son environnement en fonction de ses capteurs, *calcul* (Compute) où il choisit une destination en fonction de son observation, enfin *mouvement* (Move) où il essaie de se déplacer vers sa destination. Le mouvement peut ou non aboutir en fonction de la variante du modèle : si le robot atteint toujours sa destination, on parle de mouvement *rigide*, dans le cas contraire, le mouvement est *flexible*. D'autres variantes introduisent encore lumières, restrictions de visibilité, problématiques d'autonomie énergétique, etc.

Modèle de l'environnement. Les aléas dus à l'environnement sont représentés par un *démon*, une suite infinie de choix qui influence l'exécution d'un protocole. C'est par exemple le démon qui choisit à chaque étape ce que voient les robots, dans les limites données par le modèle (portée de vision, partage ou non de référentiel, etc.). De même, dans le cas de mouvement flexible, c'est le démon qui choisit jusqu'où se fait le déplacement. Enfin, on peut modéliser différents types d'erreurs dans l'exécution des protocoles, en particulier l'existence de robots *byzantins* au comportement arbitraire et potentiellement malveillant.

Hypothèses de synchronisation. Comme dans tous les systèmes distribués, la synchronisation a une place majeure dans les essaims de robots mobiles. Le modèle le plus général est ASYNC (ASYNChronous) qui ne commande aucune restriction sur les entrelacements des exécutions des robots. Les robots peuvent donc utiliser une vision périmée ou observer un robot pendant son déplacement.

Le modèle SSYNC (Semi-SYNChronous) restreint l'exécution à se faire en *rondes* dans lesquelles seul un sous-ensemble des robots (choisi par le démon) est actif, les autres ne faisant rien. Tous les robots actifs exécutent un cycle complet de manière atomique. Le cas particulier de SSYNC où tous les robots sont actifs à chaque ronde est nommé FSYNC (Fully SYNChronous).

Description formelle en Coq. L'objectif de la bibliothèque Pactole est de regrouper sous un même chapeau toutes les variantes de modèles : le cadre formel doit être suffisament paramétrique et flexible pour s'adapter à toutes ces variantes sans toutefois devenir inutilisable par trop de généralité. Définies formellement, les notions sont *univoques* et partageables sans altération de leur sens. Le maintien du bon contexte est en particulier assuré dans chaque application de théorème : on ne risque aucun glissement sémantique, source de nombreuses erreurs.

Le cœur du modèle réside dans la fonction `round` qui exécute un cycle Look-Compute-Move. Elle prend en paramètre un robogramme (la partie locale du protocole exécutée par chaque robot), les choix du démon à cette étape et la configuration globale initiale. Elle renvoie la nouvelle configuration globale obtenue après que chaque robot (activé) a effectué son déplacement en fonction de son observation, appelée *spectre*, du robogramme et des choix du démon. Une unique définition de `round` permet de représenter toutes les variantes que nous avons considérées jusqu'à maintenant (FSYNC, SSYNC, ASYNC, flexible, rigide, topologie de graphe, droite, plan).

Nous suivons au plus près les définitions mathématiques. Ainsi, une *configuration* est une fonction des identifiants de robots vers leurs états et une *exécution* est une suite infinie de configurations. Un *spectre* (l'observation des robots) est une vue dégradée de la configuration globale qui traduit les limites des capteurs

des robots. Toutes les définitions contiennent des paramètres dont les valeurs définissent quelle variante précise du modèle est considérée (*cf.* Section 3).

3 Développer et prouver un protocole avec Pactole

L'utilisation de la bibliothèque Pactole se fait en plusieurs phases. La première étape est de définir la variante exacte du modèle dans lequel on souhaite se placer. On définit et spécifie ensuite le protocole que l'on souhaite réaliser (et donc le problème auquel il s'attaque). Il reste enfin à prouver que le protocole défini satisfait bien sa spécification. Ces deux dernières étapes se font souvent par allers-retours car les diverses erreurs ou impasses découvertes lors de la preuve peuvent nécessiter de repenser l'algorithme ou de repréciser son comportement attendu.

Cadre formel. Un cadre complet est défini par les huit arguments suivants : (1) les nombres de robots byzantins et de robots corrects, (2) le type d'espace dans lequel évoluent les robots, (3) l'état de chaque robot (qui doit contenir au moins sa position), (4) le spectre, (5) le type de décisions que prennent les robots, (6) les changements de référentiels que peut choisir le démon, (7) les choix autorisés pour le démon concernant la mise à jour des robots (contraints par le modèle), (8) enfin la véritable fonction de mise à jour (suivant les choix du démon). Ces paramètres offrent une grande souplesse au cadre formel, ils suffisent en particulier à exprimer toutes les variantes nécessaires à nos études de cas (*cf.* Section 4).

Notons que les arguments fournis peuvent rester abstraits : on s'intéresse souvent à un problème pour un nombre quelconque de robots ou sur un graphe arbitraire. C'est systématiquement le cas dans nos études pour lesquelles le nombre de robots est arbitraire mais satisfait certaines propriétés (par exemple diviser la taille de l'anneau, ou encore respecter une certaine proportion byzantins/corrects). L'argument concerné fourni n'est alors qu'une variable.

Conception et spécification de protocoles. L'un des intérêts d'utiliser Coq est qu'il combine un langage de programmation fonctionnelle et un assistant à la preuve. Il est ainsi possible de définir le robogramme comme une fonction dans ce langage et de spécifier son comportement et le problème à résoudre avec une syntaxe mathématique familière.

Cette étape ne nécessite pas de compétence particulière en preuves de programme. Elle est *et doit rester* facilement abordable par un non expert.

Preuves. La dernière étape consiste à prouver que l'algorithme défini satisfait effectivement sa spécification. Si elle requiert une certaine expertise dans l'assistant de preuves Coq, cette phase suit néanmoins complètement une preuve « papier ». La rigueur requise par Coq peut toutefois nécessiter quelques modifications des preuves « papier », lesquelles s'avèrent parfois un peu rapides ou imprécises. Suivant la quantité de nouveaux outils mathématiques/conceptuels introduits et la complexité de la preuve, il est possible de passer davantage de temps à développer ces outils et les résultats associés qu'à la preuve du protocole lui-même ; ce n'est pas du temps perdu : une telle bibliothèque sera réutilisée.

4 Résultats

La bibliothèque est disponible à l'adresse `https://pactole.liris.cnrs.fr`. Nous l'avons utilisée avec succès pour prouver correction de protocoles, impossibilité, comparaison entre modèles pour différents problèmes (convergence, rassemblement, exploration) et sur différentes topologies (droite réelle, plan réel, graphes discrets, graphes à arêtes continues).

Convergence. Le problème de la convergence demande de faire se rapprocher les robots jusqu'à ce qu'ils soient tous dans un disque de rayon arbitrairement petit (inconnu des robots). Nous avons étudié ce problème sur la droite réelle et avons formellement montré que s'il y a au moins un tiers de robots byzantins, alors il est impossible de faire converger les robots corrects en modèle SSYNC [ABC+13]. Il s'agit bien d'une preuve d'impossibilité, la quantification est faite sur *tous* les protocoles et *quel que soit* le nombre de robots pourvu qu'une proportion soit respectée.

Rassemblement. Le rassemblement s'intéresse au placement de tous les robots corrects sur *exactement* la même position, y converger ne suffit donc pas. Nous avons étudié ce problème sur la droite et dans le plan réels, sous diverses hypothèses de synchronisation et divers spectres (capacités des capteurs). Nous avons formalisé et généralisé [CRTU15] le résultat d'impossibilité originel de Suzuki et Yamashita [SY99]. Caractérisant les configurations pour lesquelles le rassemblement n'est pas possible, nous avons ensuite conçu un protocole universel (c.-à-d. qui fonctionne dans toutes les situations non couvertes par le résultat d'impossibilité) SSYNC en une puis deux dimensions [CRTU16] pour le résoudre. Ce protocole fonctionne *quel que soit* le nombre de robots (≥ 3) et s'appuie sur l'utilisation du plus petit cercle englobant, pour lequel nous avons développé une bibliothèque. Soulignons que le passage au plan réutilise des résultats développés pour ce problème sur la droite réelle.

Dans le cas des mouvements flexibles FSYNC, nous avons formalisé la correction d'un protocole préexistant [CP05] et avons étendu ce résultat en privant les robots de la détection de multiplicité [BDR+18].

Il faut noter que tous ces travaux utilisent le même fichier de définitions. Ainsi, nous sommes sûrs de toujours parler du même problème et ce, malgré l'utilisation de différents modèles. En particulier, il n'y a avec certitude aucun glissement entre le résultat d'impossibilité et le protocole universel.

Exploration avec arrêt d'un anneau. L'exploration avec arrêt requiert de visiter au moins une fois chaque sommet d'un graphe, ici un anneau, puis de s'arrêter.

Nous avons formalisé dans ce cadre deux résultats classiques pour le modèle SSYNC : l'impossibilité de réaliser cette exploration avec arrêt si le nombre de robots divise la taille de l'anneau et le fait que toute configuration finale contient au moins deux robots sur la même position [BPRT18].

Équivalence entre modèles. Dans un schéma de synchronisation asynchrone (ASYNC), nous avons montré l'équivalence entre deux modèles de graphes : un modèle où les robots ne se trouvent que sur les sommets du graphe et un autre où les robots peuvent se trouver sur les arêtes (intuitivement, ils sont en déplacement) tout en maintenant l'observation par les autres sur les sommets (par exemple sur le sommet le plus proche) [BCP+18].

Nous avons enfin montré des résultats généraux sur les démons (suites infinies de décisions) reliant les hypothèses d'équité (fairness/k-fairness) ainsi que les mouvements rigides/flexibles sous certaines contraintes.

Références

[ABC+13] C. Auger, Z. Bouzid, P. Courtieu, S. Tixeuil, and X. Urbain. Certified Impossibility Results for Byzantine-Tolerant Mobile Robots. In T. Higashino, Y. Katayama, T. Masuzawa, M. Potop-Butucaru, and M. Yamashita, editors, *SSS 2013*, vol. 8255 of *LNCS*, pp. 178–186, 2013. Springer-Verlag.

[Bau13] A. Bauer. How to review formalized mathematics. http://math.andrej.com/2013/08/19/how-to-review-formalized-mathematics/, August 2013.

[BCP+18] T. Balabonski, P. Courtieu, R. Pelle, L. Rieg, S. Tixeuil, and X. Urbain. Brief Announcement : Continuous vs. Discrete Asynchronous Moves : a Certified Approach for Mobile Robots. In T. Izumi and P. Kuznetsov, editors, *SSS 2018*, vol. 11201 of *LNCS*, 2018. Springer-Verlag.

[BDR+18] T. Balabonski, A. Delga, L. Rieg, S. Tixeuil, and X. Urbain. Synchronous gathering without multiplicity detection : A certified algorithm. *Theory of Computing Systems*, 2018. https://doi.org/10.1007/s00224-017-9828-z.

[BPRT18] T. Balabonski, R. Pelle, L. Rieg, and S. Tixeuil. A foundational framework for certified impossibility results with mobile robots on graphs. In *Proceedings of International Conference on Distributed Computing and Networking*, 2018.

[CP05] R. Cohen and D. Peleg. Convergence Properties of the Gravitational Algorithm in Asynchronous Robot Systems. *SIAM Journal of Computing*, 34(6) :1516–1528, 2005.

[CRTU15] P. Courtieu, L. Rieg, S. Tixeuil, and X. Urbain. Impossibility of Gathering, a Certification. *Information Processing Letters*, 115 :447–452, 2015.

[CRTU16] P. Courtieu, L. Rieg, S. Tixeuil, and X. Urbain. Certified universal gathering algorithm in $\mathbb{R}^2$ for oblivious mobile robots. In Cyril Gavoille and David Ilcinkas, editors, *Distributed Computing - 30th International Symposium, (DISC 2016)*, volume 9888 of *Lecture Notes in Computer Science*, 2016. Springer-Verlag.

[PBSTU19] M. Potop-Butucaru, N. Sznajder, S. Tixeuil, and X. Urbain. Formal methods for mobile robots. In P. Flocchini, G. Prencipe, and N. Santoro, editors, *Distributed Computing by Mobile Entities*, volume 11340 of *LNCS, Theoretical Computer Science and General Issues*, pp. 278–313. Springer Nature, 2019.

[SY99] I. Suzuki and M. Yamashita. Distributed Anonymous Mobile Robots : Formation of Geometric Patterns. *SIAM Journal of Computing*, 28(4) :1347–1363, 1999.