Du discrètement continu au continûment discret

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

Du discrètement continu au continûment discret †

Thibaut Balabonski[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, CNRS, LIP6* [5] *Université de Lyon, Université Claude Bernard Lyon 1, CNRS, LIRIS UMR 5205*

Les robots mobiles oublieux ont été étudiés à la fois dans des espaces euclidiens continus et dans des espaces discrets (c'est-à-dire des graphes). Cependant, l'état de l'art actuel forme des ensembles de résultats distincts pour les deux modèles. À notre avis, le modèle continu reflète bien la physicalité des robots fonctionnant dans un environnement réel, tandis que le modèle discret reflète bien la nature numérique des robots autonomes, dont les capteurs et les capacités de calcul sont intrinsèquement finis.

Nous explorons la possibilité de faire le pont entre les deux modèles. Notre approche est certifiée à l'aide de l'assistant de preuve Coq et du framework Pactole, que nous étendons au modèle asynchrone (le plus général) sans compromettre sa généricité. Notre cadre étendu est ensuite utilisé pour prouver formellement l'équivalence entre les mouvements atomiques dans un espace discret (le modèle classique des «robots sur les graphes») et les mouvements non atomiques dans un espace unidimensionnel continu lorsque les capteurs de vision du robot sont discrets (les robots se déplacent en ligne droite entre les positions, mais leurs observations ne se traduisent que par les positions de départ ou d'arrivée), quel que soit le problème résolu. Notre effort consolide l'intégration entre le modèle, la spécification du problème et sa preuve prônée par le framework Pactole.

Mots-clefs : Essaims de robots mobiles, Espace Euclidien, Graphe, Équivalence, Assistant de preuve, Pactole, Méthodes formelles

1 Introduction

Les réseaux de robots autonomes ont capté l'attention de la communauté de l'algorithmique distribuée, de l'industrie et des grands média par les nouvelles applications et la rupture technologique qu'ils promettent. Ils permettent de se passer d'agents très onéreux et spécialisés en visant la réalisation de différentes tâches par la collaboration de capteurs mobiles déployés en essaims. Exploration d'espaces dévastés par une catastrophe naturelle ou d'origine humaine et recherche de survivants, décontamination et démantèlement de centrales dangereuses sont autant d'exemples qui montrent l'intérêt d'essaims de robots autonomes et mobiles ; ils soulignent en outre l'extrême dynamicité et la complexité de ce modèle émergent. De nombreuses variantes du modèle introduit en 1999 par Susuki et Yamashita [SY99] sont apparues au fur et à mesure de sa popularité grandissante ; le lecteur intéressé pourra en trouver les descriptions dans de récents ouvrages [FPS19]. Du point de vue théorique, il est intéressant de savoir déterminer, dans chacune de ces variantes, quelles sont les tâches réalisables ou non, et ce sous quelles conditions.

Nous considérons le cas de robots anonymes, équipés du même programme (le protocole) et opérant selon un cycle Perception-Calcul-Déplacement (Look-Compute-Move). Durant la première phase, le robot reçoit *une perception* d'un instantané de l'environnement, exprimée dans son propre référentiel auto-centré et qui n'a pas forcément les mêmes échelles ou orientations que ceux de ses congénères (il peut même être

‡ Institute of Engineering Univ. Grenoble Alpes

† This work was partially funded by the CNRS PEPS INS2I project DiDASCaL and the ANR project SAPPORO (2019-CE25-0005-1).

différent d'un cycle au suivant). À partir de cette perception *uniquement* (et donc *sans mémoire des actions passées*), le robot calcule à l'aide du programme embarqué une destination (toujours exprimée dans son propre référentiel). Il se déplace enfin vers celle-ci au cours de la dernière phase.

La topologie de l'espace dans lequel les robots évoluent est simplement un paramètre du modèle général et peut être la droite réelle, un plan euclidien, un espace discret (un graphe), etc. On peut observer à ce jour deux axes de recherches indépendants s'intéressant respectivement *(i)* aux espaces continus euclidiens et *(ii)* aux graphes ; ils étudient des problèmes de natures différentes avec des techniques algorithmiques distinctes.

Enfin, on distingue plusieurs niveaux de synchronisation : le plus faible [FPS19] est le mode asynchrone (ASYNC) où chaque robot exécute son cycle à sa propre allure. Si ASYNC est le modèle le plus pertinent en pratique, il est aussi notoirement difficile à appréhender : plusieurs résultats récents montrent que certains algorithmes écrits pour le modèle ASYNC et publiés dans des revues jugées fiables sont erronés, malgré les relectures multiples. Selon nous, le modèle ASYNC est donc une cible prioritaire pour l'application de méthodes formelles assistant à la conception et à la vérification de résultats. Dans cet article, nous poursuivons l'effort de recherche entrepris via la bibliothèque Pactole [§], développée pour l'assistant à la preuve Coq, en y introduisant le modèle ASYNC et en illustrant son utilisation pour établir un résultat d'équivalence entre les espaces continus et les espaces discrets.

2 La formalisation du modèle asynchrone

Cette section décrit l'utilisation d'un aspect ajouté récemment dans Pactole : le modèle asynchrone. Le lecteur peu familier avec l'architecture globale de Pactole consultera avec profit le résumé qui en est fait par Balabonski *et al.* [BCP+19]. La difficulté du modèle ASYNC réside dans le fait que les observations faites par les robots peuvent être réalisées pendant que d'autres robots sont en cours de déplacement. L'observation d'un robot est cependant toujours supposée donner un *instantané*, c'est-à-dire qu'elle doit traduire une position existante à un instant donné. On ne considère pas ici la possibilité d'observer des robots différents à des instants différents, ce qui donnerait une observation qui ne correspond à aucun état réel du système. De fait, avec ASYNC, il n'est plus possible d'ignorer complètement le temps que prennent les différentes phases d'un cycle. Pour rendre compte de cela, notre interpréteur du cycle Look-Compute-Move a été raffiné : la fonction `round` qui effectue un pas d'exécution du système considère maintenant que certains robots sont « en mouvement ». La représentation de leur état est enrichie en conséquence.

Au moment où un robot effectue une observation (supposée atomique), les positions des autres robots sont mises à jour (par un démon qui représente l'environnement et est vu comme un adversaire) en fonction de leur déplacement en cours afin de refléter la position effectivement observée.

Pour chaque robot `r`, un pas d'exécution (étant donnée une décision `da` du démon) s'effectue suivant [¶] :

1. Si `r` est en mouvement (destination $\neq$ position), alors sa position est mise à jour en fonction d'une décision du démon. Ceci peut entraîner une arrivée à destination.
2. Sinon `r` est à une position qui est sa destination (sur un graphe : un sommet) et une nouvelle destination est calculée de la manière suivante (qui simule la décision du programme embarqué sur la base des seules informations dont le robot dispose) :
   (a) Le démon fournit le référentiel local à `r` qui va permettre de déterminer son observation (locale).
   (b) Le programme embarqué calcule la nouvelle destination à partir (uniquement) de l'observation.
   (c) La destination obtenue est retraduite dans le référentiel global.

   L'état du robot est mis à jour avec la nouvelle destination, et sa nouvelle position si le démon le laisse commencer son déplacement.

La modélisation formelle des exécutions ASYNC, introduisant son lot de difficultés, n'a de sens que si elle peut effectivement être utilisée dans un développement formel. Nous illustrons ci-après la pertinence de notre approche en rapprochant graphes discrets et graphes continus à observation discrète.

§. La bibliothèque Pactole est disponible à l'adresse suivante : `https://pactole.liris.cnrs.fr/`.
¶. Par soucis de clarté, on a volontairement omis de cette description les parties concernant les robots byzantins.

3 Graphes discrets et continus

Deux visions coexistent quant aux robots sur les graphes : se téléportent-ils d'un nœud à l'autre (dans un graphe discret) ou bien se déplacent-ils continûment le long des arêtes (dans un graphe cette fois continu) ?

Le modèle discret. Le modèle discret est défini par un graphe orienté $G = (V, E)$ qui dénote l'espace dans lequel évoluent les robots. Les robots ne peuvent être présents que sur les sommets, ce qui implique que lors d'un déplacement, le démon n'a que deux choix possibles : ou bien le robot reste à son point de départ, ou bien il atteint sa destination ; il n'y a pas de position intermédiaire. Pour représenter des graphes non orientés, on impose que pour chaque arête $e \in E$, il y ait une arête dans le sens contraire.

Le modèle continu. Le modèle continu s'appuie sur le même graphe orienté $G = (V, E)$ que le modèle discret. Cependant, à l'inverse de ce dernier, les robots peuvent être situés sur des sommets ou sur des arêtes. La position d'un robot sur une arête est représentée par un pourcentage p de son parcours. Les observations des robots sont aussi précises que leurs positions : un robot peut différencier deux autres robots à deux niveaux différents d'une même arête. Bien entendu le modèle continu est strictement différent, et plus puissant, que le modèle discret si aucune restriction supplémentaire n'est imposée.

Le modèle continu avec « capteur discrets ». Nous proposons ici une unique et simple restriction : les capteurs des robots produisent une version « discrétisée » de l'environnement qui les entoure. Plus précisément nous nous plaçons dans l'hypothèse où les capteurs transforment une position intermédiaire entre deux sommets s_1 et s_2 du graphe en soit s_1 soit s_2, le choix entre les deux étant décidé en fonction d'un *seuil* fixé pour chaque arête du graphe (*cf.* formalisation ci-dessous). Ce seuil, constant pour chaque arête, fait partie de la description de G. Seule l'observation change, les positions des robots restent continues.

Comme nous allons le voir, cette unique restriction suffit à rendre les deux modèles équivalents.

Formalisation des modèles. La bibliothèque Pactole a été conçue de façon très paramétrable afin de s'adapter au maximum de modèles. En particulier sont laissés en paramètres : l'espace dans lequel les robots évoluent, le type des états des robots, le type des observations qu'ils effectuent (pour représenter différents types de capteurs), les espaces de choix du démon pour le changement de référentiel ou les mises à jour de l'état des robots, etc.

Ici, l'espace est un graphe $G = (V, E, \text{threshold})$ où la fonction $\text{threshold} : E \to]0, 1[$ représente le seuil de chaque arête. Elle n'est pas utilisée dans le modèle discret.

Les positions des robots (classe `Location` de Pactole) sont, pour le modèle discret : les sommets V du graphe G, et pour le modèle continu : les sommets ou les arêtes avec un pourcentage (type `strict_ratio`).

```
Inductive locationG := OnVertex (l:locationV) | OnEdge (e:E) (p:strict_ratio).
```

Nous définissons la notion d'*observation discrète* en utilisant la projection suivante dans le calcul de l'observation, qui convertit une position continue en une position discrète :

```
Definition LocC2D (locC : locationG) : locationV :=
  match locC with
  | OnVertex l ⇒ l                          (* Exactement sur un noeud       *)
  | OnEdge e p ⇒ if  (G.(threshold) e) ≤ p  (* Ou suivant threshold et ratio *)
                 then G.(tgt) e             (* - Vu a la fin de l'arete e    *)
                 else G.(src) e             (* - Ou bien vu a son origine    *)
  end.
```

Dans le modèle discret, l'état d'un robot contient sa position et sa destination actuelle, la direction (l'arête) le long de laquelle il se déplace, avec la contrainte que la position doit être la source ou la destination (lorsqu'il est déjà arrivé) de l'arête. Dans le modèle continu, l'état est soit le même que dans le cas discret (si le robots est sur un sommet), soit ne contient que la position. En effet, dans le cas où le robot se trouve sur une arête, cette arête est nécessairement celle le long de laquelle il se déplace, et aucune contrainte supplémentaire n'est nécessaire pour indiquer son intention.

Le type des observations des robots est un paramètre laissé abstrait puisque notre résultat n'en dépend pas.

Les changements de référentiel fixés par le démon sont donnés par un isomorphisme du graphe G préservant les seuils. Les choix du démon concernant les déplacements sont un booléen dans le cas discret

(a-t-on atteint la destination ?) et un pourcentage dans le cas continu qui sera ajouté au pourcentage de la position du robot sur son arête (dans la limite de 100%).

4 Le pont formel entre les modèles

Les deux théorèmes suivants établissent une bisimulation entre les exécutions ASYNC sur graphes discrets et sur graphes continus à observation discrète, lorsque les robots ne peuvent pas être interrompus lors des déplacements (rigidité). Un corollaire est que toute solution à un problème dans l'un des deux modèles peut être traduite en une solution dans l'autre.

Ces résultats se fondent sur une traduction systématique de tous les composants d'un modèle dans l'autre : positions des robots, états des robots, configurations, programme embarqué, démons et, enfin, exécutions. Ces transformations entre modèles sont formellement montrées inverses l'une de l'autre. Notons en particulier qu'on sait traduire le programme embarqué d'un modèle vers l'autre.

Pour toute configuration, tout programme et toute action du démon dans le cadre discret, la configuration résultant d'un pas d'exécution, transposée dans le cadre continu, est le résultat d'un pas des traductions dans le cadre continu/observation discrète du programme et de l'action depuis la traduction de la configuration.

```
Theorem graph_equivD2C : ∀ (c : DGF_config) (rbg : robogramV) (da : DGF_da),
  config_V2G (round rbg da c) ≡ round (rbg_V2G rbg)(da_D2C da)(config_V2G c).
```

Pour toute configuration, tout programme et toute action du démon dans le cadre continu/observation discrète avec déplacements rigides, la traduction vers le modèle discret d'un pas d'exécution est le résultat d'un pas des traductions respectives du programme et de l'action du démon sur la traduction de la configuration.

```
Theorem graph_equiv_C2D : ∀ (c : CGF_config) (rbg : robogramG) (da : CGF_da),
  config_G2V (round rbg da c) ≡ round (rbg_G2V rbg)(da_C2D da c)(config_G2V c).
```

5 Conclusion

Nous avons étendu la bibliothèque Pactole du modèle le plus délicat à appréhender pour des humains : le modèle ASYNC. Nous avons ensuite utilisé cette modélisation pour obtenir une preuve certifiée que des déplacements discrets sont équivalents à des déplacements continus quand les observations des positions des robots se font de manière discrète. Ces observations discrètes correspondent à des plate-formes d'expérimentations réelles où l'environnement dans lequel évoluent les robots est doté de capteurs de présence binaires dont l'observation est retransmise aux robots par l'environnement [OMDT19]. Notre travail rend possible l'utilisation d'algorithmes déjà obtenus dans un cadre discret dans ce nouveau contexte plus réaliste.

Par nature les preuves faites dans un assistant de preuve sont dignes de confiance (si elles ne contiennent pas d'axiome). Il est en revanche parfois difficile de vérifier *de quoi* elles sont la preuve. Pour cela le lecteur, et notamment le relecteur, doit vérifier que les *énoncés* des théorèmes (et donc également les définitions que ces énoncés utilisent) sont bien conformes à ce que les auteurs prétendent. Le lecteur intéressé pourra lire le travail de Bauer [Bau13]. Pactole vise aussi à rendre ces spécifications et énoncés les plus lisibles possibles.

Références

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

[BCP+19] Thibaut Balabonski, Pierre Courtieu, Robin Pelle, Lionel Rieg, Sébastien Tixeuil, and Xavier Urbain. Manuel de savoir-prouver à l'usage des roboteux et des distributeux. In *ALGOTEL 2019 - 21èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, pages 1–4, Saint Laurent de la Cabrerisse, France, 2019.

[FPS19] Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro, editors. *Distributed Computing by Mobile Entities*, volume 11340 of *Lecture Notes in Computer Science, Theoretical Computer Science and General Issues*. Springer Nature, 2019.

[OMDT19] Keisuke Okumura, Manao Machida, Xavier Défago, and Yasumasa Tamura. Priority inheritance with backtracking for iterative multi-agent path finding. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 535–542. ijcai.org, 2019.

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