Synthèse du rapport de thèse en français

Les tâches qui peuvent être effectuées par des robots autonomes sont de plus en plus nombreuses et complexes, à la fois dans notre vie de tous les jours et dans l'industrie. De nombreuses études se sont intéressées récemment aux réseaux de robots mobiles et autonomes qui doivent coopérer pour réaliser une tâche complexe commune qu'ils ne peuvent accomplir seuls. On peut trouver des exemples de tels travaux dans [Pre13, PRT11, FPS12]. Les applications concernées par ce type de systèmes comprennent l'exploration d'environnements inconnus, la surveillance de zones à risque, la construction de cartes pour de telles zones, etc. Par exemple, ces robots peuvent patrouiller les quais dans un port, comme décrit dans la Figure 1. Ils

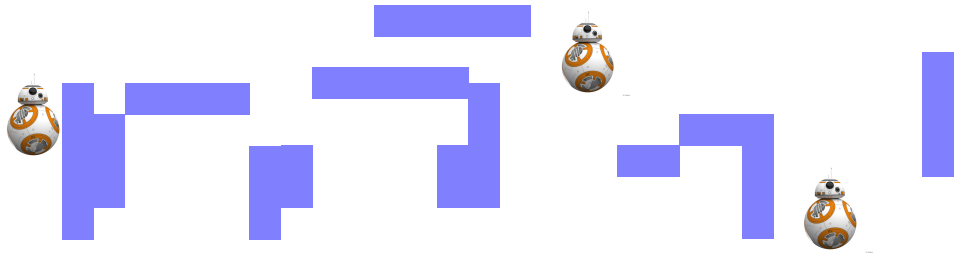

FIGURE 1 – Exemple d'applications pour robots mobiles

doivent se coordonner pour contrôler des conteneurs de marchandises, en vérifiant régulièrement qu'aucun d'eux n'a été ouvert. Si un intrus est repéré, les robots doivent le repousser à l'extérieur du périmètre de sécurité, puis retourner surveiller les conteneurs.

Un système constitué d'une telle équipe est appelé *système distribué* ou *réparti* [Lyn96a, Tel01]. Ce qualificatif de "distribué" provient du fait que le système est composé d'un ensemble d'entités de calcul autonomes dotés de capacités de communication qui leur permettent de résoudre une tâche commune. Traditionnellement, les entités sont immobiles et communiquent les unes avec les autres par passage de messages. Le modèle de robot que nous étudions dans cette thèse diffère du modèle classique par deux aspects : les entités sont mobiles et ne communiquent pas par passage de messages. Par ailleurs, ces entités mobiles peuvent être dotés de capacités limitées.

Les travaux menés sur les réseaux de robots mobiles s'intéressent aux caractéristiques de bases qui sont nécessaires afin que l'équipe de robots puisse accomplir une tâche donnée, en l'absence de toute autorité centrale. Ces travaux considèrent des déplacements des robots dans un plan euclidien ou dans un environnement

discret. Les tâches données peuvent être critiques, il faut donc veiller à ce que les robots les accomplissent correctement, en dépit de leurs limitations.

Jusqu'à présent, les réseaux de robots ont été étudiés de manière empirique et la plupart des résultats ont été validés seulement par des preuves faites "à la main". Ces preuves visent à vérifier que l'objectif des robots, appelé *spécification* du problème et décrit par un ensemble de propriétés, est satisfait. Elles sont difficiles à lire et à écrire, car le raisonnement sur des systèmes complexes est à la fois lourd et source d'erreurs. L'utilisation de preuves automatiques, basées sur un modèle mathématique abstrait du système, pourrait simplifier ce processus de vérification. Pour ces techniques de vérification formelle, plusieurs approches existent :

— La *génération automatique de tests* prend en entrée une description formelle des comportements souhaités du système et génère un ensemble de tests à effectuer, qui doit couvrir un ensemble de comportements le plus grand possible. Le système est ensuite exécuté conformément à ces tests, et l'on peut vérifier si les sorties sont identiques à celles attendues. Cependant il n'est pas toujours possible de fournir un ensemble exhaustif de tests, et la conception de tests est difficile pour les systèmes concurrents ou distribués.

— La *démonstration automatique* laisse l'ordinateur prouver automatiquement des théorèmes décrivant les propriétés du système, en se basant sur une description mathématique du programme, et sur un ensemble de règles de déduction et d'axiomes. Cette méthode n'est cependant pas totalement automatisable, et l'assistant de preuves a besoin d'être plus ou moins guidé par une aide humaine pendant le processus. D'ailleurs, quand une preuve échoue, il est difficile d'en connaitre la raison. Divers assistants de preuve ont émergé depuis les années 60, par exemple PVS [1], Coq [2] et Isabelle/HOL [3] sont parmi les plus largement utilisés.

— Le *model-checking* part d'une représentation abstraite (un *modèle*) du programme et d'une spécification formelle des comportements souhaités, et vérifie de façon exhaustive et automatique que tous les comportements du modèle satisfont cette spécification. Bien sûr, la recherche n'est exhaustive que sur l'abstraction du système considérée. Un avantage de cette technique est sa complétude et sa capacité à extraire des comportements invalidant les propriétés. Ainsi, le *model-checking* peut être utilisé dès la phase de conception pour valider les prototypes du système. L'explosion combinatoire est l'inconvénient majeur de cette méthode : explorer symboliquement toutes les exécutions (l'ensemble peut être infini) dans un modèle de grande taille est

1. http ://pvs.csl.sri.com
2. https ://coq.inria.fr/
3. https : //isabelle.in.tum.de

très consommateur de temps et de mémoire. Les models-checkers UPPAAL [4] et SPIN [5] sont parmi les plus connus.

Chacune de ces approches a ses avantages et ses inconvénients : alors que les tests peuvent être faciles à mettre en œuvre, ils ne peuvent pas être appliqués dans la phase de conception, et la génération de tests exhaustifs est difficile. Les preuves sont difficiles à mettre en œuvre car elles nécessitent la présence d'un expert, mais elles sont exhaustives, applicables dés la phase de conception, et peuvent s'appliquer à des systèmes dits *paramétrés*, où certaines variables ne sont pas spécifiées, comme par exemple le nombre de processus. Enfin, le *model checking* est une approche globale et automatique, mais limitée par la taille du modèle. En outre, le problème du *model checking* de systèmes paramétrés est indécidable en général [AK86].

Alors que le *model checking* vérifie qu'un modèle satisfait une propriété, la *synthèse* consiste à se donner une spécification et à générer, si possible, un modèle qui la satisfasse. L'avantage de cette méthode est qu'elle se situe en amont de la conception et que les protocoles ainsi générés sont corrects par construction. Les premiers travaux sur la synthèse remontent à Church [Chu63] et ont été développés ensuite dans [BL69, CE81, MW84, PR90]. Formellement, le problème de décision est le suivant : étant donnée une spécification, existe-t-il un modèle qui la satisfasse ? Le problème de synthèse proprement dit consiste, lorsque la réponse est positive, à construire effectivement le modèle.

Dans le cas de systèmes ouverts, qui interagissent avec un environnement, Buchi et Landweber [BL69] ont montré que le problème était décidable dans le cadre de la théorie des jeux. Cette approche consiste à considérer le problème de synthèse comme un *jeu* entre le système et l'environnement. Le système et son environnement sont deux joueurs qui s'affrontent, et le système gagne s'il satisfait la spécification initiale. Ainsi, trouver un modèle qui satisfait la spécification revient à trouver une stratégie gagnante pour le système, quel que soit les stratéges de l'environnement. Toutefois, ce problème est aussi indécidable en général pour les systèmes distribués, la décidabilité étant obtenue avec des hypothèses supplémentaires sur l'architecture [PR90].

Dans ce travail, notre objectif est d'étudier comment les méthodes formelles peuvent être appliquées dans le contexte d'algorithmes de robots mobiles. Après une présentation de ces algorithmes, nous montrons les avantages de ces méthodes par rapport aux approches traditionnelles.

4. http ://www.uppaal.org
5. http ://spinroot.com/spin/whatispin.html

1 Robots Mobiles

Dans cette thèse nous nous sommes intéressés à un modèle théorique [SY99, FPS12] dans lequel les robots qui coopèrent pour atteindre un objectif commun ont des capacités limitées. Les robots fonctionnent selon un cycle composé de trois phases : *Look, Compute* et *Move*. Quand un robot exécute la première phase, il examine l'environnement qui l'entoure et s'inscrit dans un repère constitué par l'ensemble des autres robots. Dans la deuxième phase, il calcule un futur mouvement en fonction de sa position relative aux autres robots, et enfin, dans la dernière phase, il met en œuvre le mouvement calculé précédemment.

Ce modèle, appelé SYm (ou ATOM), correspond à un comportement synchrone des robots, qui exécutent ensemble chacune des phases. Il a été étendu par Prencipe [Pre00] afin de prendre en compte un comportement asynchrone plus réaliste pour des systèmes distribués. Ce nouveau modèle porte le nom de CORDA. Ces modèles diffèrent par leurs degrés d'atomicité :

— Dans le modèle historique, ATOM [SY99], un sous ensemble de robots exécute chacune des trois phases de manière atomique. Il existe deux variantes : Fsync (Fully-synchronous) où tous les robots sont synchronisés, et Ssync (Semi-synchronous) où à chaque cycle, seul un sous-ensemble de robots s'exécute de manière synchrone.

  Dans ce modèle, le comportement du système correspond à l'exécution de toutes les opérations instantanément, la conséquence est qu'aucun robot ne pourra jamais être observé alors qu'il se déplace, et la compréhension de l'univers par les robots actifs est toujours cohérente.

— Le second modèle, CORDA [Pre00], ou Async, est une variante moins contrainte et donc plus réaliste, où chaque robot peut être activé de manière asynchrone, indépendamment des autres robots. La durée de chaque phase, et le temps entre les phases successives d'un même cycle sont limitées mais inconnues. En conséquence, les calculs peuvent être basés sur des observations totalement obsolètes, prises arbitrairement loin dans le passé.

Notons qu'en terme d'exécutions, Fsync est inclus dans Ssync et Ssync est inclus dans Async.

La capacité d'une équipe à réaliser une tâche assignée dépend principalement de ses membres et de leurs capacités : plus les robots sont puissants, plus le problème est résolu facilement. Les robots du modèle considéré ici disposent quant à eux de capacités très limitées. Nous allons maintenant détailler les hypothèses minimales généralement faites sur les capacités de ces robots.

1.1 Des robots restreints

Les robots sont identiques et anonymes, ils exécutent le même algorithme et ils ne peuvent pas être distingués par leur apparence, mais ils peuvent avoir des vitesses de calcul et de déplacement différentes (dans le cas de Async). De plus ces robots peuvent avoir des identités mais ni eux ni les autres robots n'ont connaissance ou accès à ces identités.

Les robots sont amnésiques *i.e.*, ils n'ont aucun souvenir de leurs actions passées. Cette capacité implique que tout état peut être considéré comme initial. Par conséquent, les algorithmes de robots sont auto-stabilisants. Un algorithme distribué est dit auto-stabilisant s'il assure qu'un comportement correct peut être retrouvé en un temps fini sans aucune aide exterieure.

Les robots n'ont pas un sens commun de l'orientation. Chaque robot a sa propre unité de longueur, et possède une boussole locale définissant son propre système de coordonnées cartésiennes locales. Ce système de coordonnées local est auto-centré, *i.e.*, l'origine est la position du robot. De plus, le système de coordonnées local des robots peut complètement changer à chaque cycle, mais il reste invariant au cours d'un cycle.

Les robots sont silencieux : il ne communiquent pas entre eux de manière explicite, mais seulement en observant les positions des autres robots, et prennent leurs décisions en conséquence. En d'autres termes, le seul moyen pour un robot d'envoyer des informations à un autre robot est de se déplacer et de laisser les autres robots observer son mouvement avant de bouger à nouveau.

Bien que dans la littérature des robots plus faibles soient étudiés, nous nous sommes intéressés dans cette thèse aux robots décrit par le modèle original de Suzuki et Yamashita [SY99].

1.2 Les capacités des robots

Pour exécuter leurs cycles Look-Compute-Move, les robots peuvent observer leur environnement, prendre des décision et se mouvoir. Deux types d'environnements ont été étudiés :

— le plan euclidien (continu) [SY99, FPS12],

— un environnement discret [KMP06, FIPS13], représenté par un graphe, dans lequel les noeuds correspondent aux différentes positions possibles et les arcs aux routes permettant aux robots d'aller d'une position à une autre.

La représentation discrète est motivée par des aspects pratiques liés au manque de fiabilité des dispositifs d'observation utilisés par les robots ainsi qu'à l'imprécision de leur motorisation [CDD+08]. Cela permet notamment de simplifier la conception des modèles en raisonnant sur des structures finies. Cependant, ce cadre rend le

modèle plus sensible à la taille des constantes, ce qui peut augmenter de manière significative le nombre de configurations symétriques lorsque le graphe sous-jacent est également symétrique (par exemple un anneau) et donc la taille des preuves de correction [DSN11, KLOT11, KLK+12].

1.2.1 Observation

Les robots sont dotés de capteurs de vision fournissant les emplacements des autres robots. L'emplacement est obtenu soit à grain fin (avec un certain degré de précision) ou à gros grain. Dans le premier cas, la littérature se réfère principalement au modèle où l'espace est continu, tandis que dans le second cas l'environnement est discret.

Les robots n'ont pas de dimension et donc leur visibilité ne peut être obstruée : si trois robots r_1, r_2, et r_3 sont alignés, avec r_2 au milieu, r_1 peut quand même voir r_3. Les robots peuvent partager la même position : cette formation est appelée une *tour* [FPS12]. La capacité pour un robot de détecter la multiplicité est essentielle pour réaliser certaines tâches particulières. Parmi les détecteurs de multiplicité, nous distinguons :

— les détecteurs *faibles*, capables de déterminer si il y a zéro, un ou plusieurs robots à une position particulière,

— les détecteurs *forts* qui fournissent le nombre exact de robots à une position particulière.

Le capteur correspondant peut être local ou global : dans le contexte local un robot ne détecte que la multiplicité à sa position courante, alors que dans le cadre global le nombre de robots sur chaque position est connu. Dans le modèle que nous étudions il n'y a aucune restriction sur la distance de visibilité des robots.

1.2.2 Calcul

Comme dans les systèmes distribués classiques, les robots sont supposés être en mesure d'effectuer toute suite finie de calculs en temps négligeable. Comme les robots sont amnésiques, la mémoire volatile est utilisée pour effectuer le cycle Look-Compute-Move, le contenu de la mémoire est effacé à la fin de chaque cycle. Le calcul prend en entrée l'observation faite dans la phase Look et retourne un movement. Lorsque deux robots sont sur la même position ou sont symétriques, ils calculent le même mouvement.

1.2.3 Mouvement

Les robots peuvent se déplacer seulement à l'emplacement déterminé par la phase de calcul du cycle en cours. Dans certains cas, en raison de la symétrie, la

position calculée peut être ambiguë : elle correspond alors à un mouvement non déterministe, qui peut être résolu par un ordonnanceur. Dans le modèle discret, un robot peut se déplacer seulement vers un emplacement adjacent à sa position courante. Dans le modèle continu, un robot se déplace vers sa destination quelle qu'elle soit.

1.2.4 Ordonnancement

Lorsque plusieurs processus veulent s'exécuter simultanément, il est nécessaire de determiner quel processus sera exécuté et à quel moment. Les ordonnanceurs sont des abstractions utilisées pour caractériser le degré d'asynchronisme des robots [DGMP06, FPS12]. Un ordonnanceur *équitable* est généralement utilisé, il active tous les robots infiniment souvent, mais certains robots peuvent être activés arbitrairement plus que les autres.

Il existe d'autres types d'équité qui dépendent de l'ordonnancement des actions plutôt que des processus. Un ordonnanceur *fort* garantit que chaque action tirable infiniment souvent possible sera exécutée infiniment souvent. Un ordonnanceur *faible* garantit que chaque action tirable continûment à partir d'un certain moment sera exécutée une infinité de fois.

Dans cette thèse, nous considérons des robots dans un univers discret. Nous nous sommes principalement intéressés à deux problèmes largement étudiés pour les robots mobiles et autonomes : le premier est celui du rassemblement [KKM10], où les robots doivent se retrouver sur une unique position, le second est celui de l'exploration [FIPS13, DPT13], où les robots doivent visiter tous les noeuds du graphe.

Le problème du rassemblement est le premier à avoir été étudié dans la littérature, aussi bien historiquement [SY99, KMP06, KKN08, Pel11] que par le nombre de publications. Quelles que soient leurs positions initiales, les robots doivent se déplacer afin d'être finalement tous regroupés sur une même position, non connue à l'avance, et y rester par la suite. Comme le problème de consensus dans les systèmes distribués classiques, où toutes les entités doivent être d'accord sur une même valeur, le rassemblement a une définition simple, mais l'existence d'une solution dépend du degré de synchronisme du système.

Un équipe de robot a exploré un graphe si chaque position de celui-ci est visité par au moins un robot. Le problème de l'exploration a plusieurs variantes, parmi elles nous avons étudié l'exploration avec arrêt et l'exploration perpétuelle exclusive. L'exploration avec arrêt est la version historique du travail sur l'exploration [FIPS13, LPT10, DPT13]. Les robots doivent atteindre une configuration dans laquelle ils sont tous inactifs et où chaque noeud du graphe a été visité par au moins un robot. La difficulté de cette tâche réside dans le fait que les robots

doivent s'arrêter une fois l'exploration faite. En l'absence de mémoire persistante, cela signifie qu'ils doivent être en mesure de faire la distinction entre les différentes étapes du processus d'exploration. L'exploration perpétuelle exclusive a été étudiée plus récemment [BMPT10, DSN+13]. Chaque noeud du graphe doit être visité infiniment souvent et aucun point de multiplicité ne doit apparaître.

2 Méthodes formelles et algorithmes de robots

L'utilisation de méthodes formelles exige des représentations mathématiques du système et de ses spécifications, données comme un ensemble de propriétés. Un système distribué est souvent décrit comme un système de transition [Tel01, Lyn96b], composé des modèles de ses sous-systèmes. Les propriétés peuvent être classées en différents types, parmi lesquels les propriétés de sûreté et de vivacité. Les propriétés de sureté exigent que « quelque chose de mauvais ne se produise jamais », comme l'absence de blocage dans l'exécution d'un système. Les invariants forment une sous-classe importante des propriétés de sûreté, exprimant que « quelque chose est vrai à chaque étape de chaque exécution ». Les propriétés de vivacité exigent que « quelque chose de bon finisse par arriver », par exemple que chaque processus ait finalement accès aux ressources critiques. Toute spécification peut être écrite comme la conjonction de propriétés de sécurité et de vivacité [AS85].

2.1 Model-checking

Un *model-checker* prend en entrée un modèle M, souvent sous la forme d'un système de transitions, décrivant toutes les exécutions possibles du système, et la propriété à vérifier, exprimée par une formule logique φ. Il détermine si le modèle satisfait ou non la formule. Lorsque la propriété n'est pas satisfaite, le *model-checker* fournit un contre-exemple, *i.e.*, une exécution du modèle qui invalide la propriété. Ce contre-exemple est utile pour trouver des erreurs dans les systèmes complexes, c'est un avantage majeur du model-checking par rapport aux autres méthodes formelles, comme la démonstration de théorèmes, qui peuvent infirmer une propriété, mais sans fournir systématiquement un tel contre-exemple.

L'approche par automates pour le *model-checking* a été introduite par Vardi et Wolper [VW86], afin de fournir un cadre unifié et extensible, d'abord appliqué à une classe de formules logiques appelée LTL. Cette approche comprend trois étapes :

— Tout d'abord le système et ses spécifications doivent être modélisés. Soit M le modèle du système, et φ la formule représentant l'ensemble des spécifications du système. Le langage $\mathcal{L}(M)$ associé à M représente toutes les exécutions

de M. La négation de la formule φ est traduite en un automate $\mathcal{A}_{\neg\varphi}$ dont le langage, $\mathcal{L}(A_{\neg\varphi})$, est l'ensemble des exécutions qui invalident φ.

— Les automates M et $\mathcal{A}_{\neg\varphi}$ sont synchronisés afin d'obtenir un automate $M \times \mathcal{A}_{\neg\varphi}$ dont le langage $\mathcal{L}(M \times \mathcal{A}_{\neg\varphi}) = \mathcal{L}(M) \cap \mathcal{L}(\mathcal{A}_{\neg\varphi})$,est l'ensemble des exécutions de M qui invalident φ.

— Enfin, le *model-checker* effectue un test de vacuité sur le produit. Le modèle M satisfait φ si et seulement si $\mathcal{L}(M \times \mathcal{A}_{\neg\varphi}) = \emptyset$. Si le test de vacuité est positif, cela veut dire qu'aucune exécution n'invalide φ, les propriétés exprimées par φ sont donc satisfaites par le modèle M. Sinon, une exécution de M qui invalide φ est retournée comme contre-exemple.

L'inconvénient de cette méthode est le problème bien connu d'*explosion combinatoire* [Val96], lorsque le produit $M \times \mathcal{A}_{\neg\varphi}$ est de grande taille. En particulier, lorsque le modèle M du système est lui-même obtenu par produit de plusieurs composants, sa taille eest xponentielle en le nombre de processus. Ainsi, dans l'approche par automates, le produit est souvent trop grand pour que le test de vacuité puisse être réalisé dans un délai raisonnable en terme de temps d'exécution ou de mémoire utilisée.

Les systèmes distribués sont naturellement structurés comme un ensemble de plusieurs processus, parmi lesquels certains présentent des comportements similaires. De tels composants sont dits symétriques, et connaître le comportement d'un de ces composants est souvent suffisant pour connaître le comportement de ses pairs. Plus formellement, les symétries du système définissent une relation d'équivalence sur ses états, qui permet de construire un espace d'états réduit, par quotient, où au moins un état par classe d'équivalence est maintenu. Si un seul représentant par classe est maintenu, la réduction maximale est atteinte. La définition de symétries garantit que l'espace d'états réduit préserve les propriétés, si les symétries sont respectés [CJEF96, ES96]. Cette réduction est généralement exponentiellement plus petite que l'espace d'états initial, ce qui réduit le temps d'exécution et la mémoire utilisée lors la procédure de vérification.

À notre connaissance, dans le contexte des réseaux de robots mobiles opérant dans l'espace discret, une seule tentative, par Devismes *et al.* [DLP$^+$12], a étudié la possibilité d'utiliser le model-checking comme méthode de vérification. Les auteurs utilisent LUSTRE [HCRP91] pour décrire et vérifier le problème de l'exploration avec arrêt sur une grille de taille 3×3, par 3 robots, dans le modèle Ssync. Ils considèrent un type particulier de configurations avec une tour de 2 robots et un robot isolé, où seul le robot isolé souhaite se déplacer. Ils vérifient l'invariant : *nœuds visités* ≤ 4, ce qui leur permet de compléter leurs preuves manuelles par de la vérification formelle pour ce cas précis.

2.2 Preuves

En utilisant un assistant de preuves, un utilisateur peut exprimer des données, des programmes, des théorèmes et des preuves. Les assistants de preuves fournissent une garantie supplémentaire en vérifiant mécaniquement la solidité de la preuve une fois qu'un expert l'a développée de manière interactive. Ils ont été utilisés avec succès pour diverses tâches telles que la formalisation de la sémantique des langages de programmation [Ler09], la certification d'un noyau de l'OS [KAE+10], ou la vérification de protocoles cryptographiques [ABB+12]. Au cours des vingt dernières années, l'utilisation d'assistants de preuves automatiques s'est étendue à la validation de systèmes distribués.

L'inconvénient majeur de cette méthode est qu'elle nécessite un fort niveau d'expertise pour l'écriture de la preuve. De plus elle n'est pas complètement automatique, car les preuves sont souvent obtenues à partir de nombreux lemmes.

Pour le modèle de robots, Courtieu *et al.* ont utilisé COQ afin de certifier des résultats d'impossibilité, en présence de processus byzantins [ABC+13]. Une preuve certifiée du résultat de [SY99] est proposé dans [CRTU15], confirmant l'impossibilité de regrouper deux robots. Les auteurs fournissent également un résultat plus général d'impossibilité : rassembler un nombre pair de robots, lorsque deux robots sont initialement sur la même position, est impossible.

2.3 Synthèse d'algorithmes

Les techniques de synthèse sont situées plus en amont, puisqu'elles permettent, dans certains cas, de générer automatiquement un algorithme correct. Pour cela, soit φ la spécification que le système doit satisfaire, et soit E un modèle de l'environnement. Le problème de synthèse cherche s'il existe un programme P tel que $P \times E$ satisfait φ. Ainsi, le système atteint son objectif quel que soit le comportement de l'environnement. Lorsque la réponse est positive, le programme doit alors être construit. Une réponse négative donne une preuve qu'il y a toujours une façon pour l'environnement d'empêcher le système d'atteindre son objectif. Une telle stratégie de l'environnement peut être vue comme une preuve d'impossibilité.

Au premier abord on pourrait croire que le modèle réellement nécessaire est celui des *jeux distribués*, dans lequel chaque robot représente un joueur distinct, tous ces joueurs coopérant contre un environnement hostile. Dans les jeux distribués, l'existence d'une stratégie gagnante pour l'équipe de joueurs est indécidable [PR79]. Cependant, le fait que les robots soient capables de voir leur environnement, et donc de toujours connaître la configuration du système, nous permet de rester dans le cadre de jeux à 2 joueurs. Il est donc possible de coder l'ensemble des robots comme un seul joueur. Bien sûr, la stratégie obtenu sera centralisée, le jeu devra être conçu afin de n'obtenir que des stratégies qui peuvent être distribuées

afin d'être implémentées sur les robots.

À notre connaissance, dans le contexte des réseaux de robots mobiles, une seule tentative [BDP+12] examine la possibilité de synthétiser automatiquement des protocoles de robots mobiles. Ce travail considère l'exploration perpétuelle exclusive par k robots d'anneaux de tailles n quelconque dans le modèle Ssync. L'approche choisie est brute force : tous les protocoles possibles sont générés mécaniquement sans qu'aucune propriété à satisfaire ne soit spécifiée. Les protocoles ainsi obtenus ne contiennent aucune *ambiguïté*, c'est à dire qu'ils ne contiennent pas de configurations symétriques, ni de de multiplicité. Une fois ces protocoles construit chacun est étudié manuellement afin de voir s'il permet ou non une exploration perpétuelle exclusive. Tous les protocoles qui le permettent sont ensuite comparés afin d'obtenir une étude qualitative.

Dans cette thèse nous avons voulu donner suite à ces travaux, en démontrant que le model-checking pouvait être utilisé dans les réseaux de robots afin de vérifier des protocoles entiers et non seulement comme aide pour des preuves manuelles. Nous avons voulu montrer le pouvoir de la synthèse qui permet de générer automatiquement des algorithmes corrects par construction sans se restreindre sur les mouvements ou les configurations possibles. Notre approche diffère des précédentes, car notre modèle est suffisant général pour contenir tous les modèles d'atomicité, alors que les travaux antérieurs ne gèrent que les modèles synchrones.

3 Contributions

Nous avons construit un modèle formel représentant un réseau de robots mobiles comme un produit d'automates. Ce modèle peut représenter les trois hypothèses d'exécution décrites ci-dessus, à savoir Fsync, Ssync et Async. Nous en proposons une implémentation qui permet de réduire la taille de l'espace d'états en utilisant les symétries du modèles.

Grâce à ce modèle, deux contributions ont été réalisées : nous avons tout d'abord vérifié formellement deux protocoles existants, qui permettent de résoudre les deux variantes de l'exploration d'anneau avec des robots asynchrones. Afin de vérifier les algorithmes de robots, nous avons créé un générateur de modèles qui, sur un anneau et quelle que soit la tâche à effectuer, prend en entrée la taille de l'anneau, le nombre de robots, le modèle d'exécution du système et un algorithme à vérifier sous la forme de transitions gardées, puis construit le modèle du système. Une fois les spécifications du système formellement énoncées, l'utilisation d'un model-checker nous permet de vérifier l'algorithme pour une taille d'anneau et un nombre de robots fixés. Nous avons d'abord vérifié le protocole de Flocchini *et al* qui résout le problème de l'exploration avec arrêt [FIPS13], puis nous avons étudié le problème de l'exploration perpétuelle exclusive avec l'algorithme de Blin

et al [BMPT10]. Dans les travaux d'origines, les deux protocoles ont été donnés par une suite de descriptions informelles, que nous avons du formaliser.

Dans le cas de l'exploration avec arrêt [FIPS13], il a été prouvé correct grâce à une preuve manuelle lorsque le nombre de robots est $k > 17$ et n (la taille de l'anneau) et k sont premiers entre eux. La nécessité de la borne $k > 17$ n'ayant pas été prouvée dans le papier d'origine, notre méthodologie démontre que pour de nombreux cas de k et n non couverts dans le document original, le protocole est toujours correct. Nous offrons une conjecture pour les cas avec $k \leq 17$ robots.

Nous avons ensuite étudié l'algorithme de Blin *et al* qui permet à un minimum de robots d'explorer perpétuellement un anneau sans que des collisions ne surviennent [BMPT10]. Dans ce cas notre méthodologie nous permet d'obtenir un contre-exemple relevant d'un défaut subtil dans l'algorithme, lorsque celui est exécuté dans le modèle Async. Nous proposons une correction du protocole original et vérifions par model-checking plusieurs instances du nouvel algorithme. Une fois après avoir vérifié la correction de l'algorithme pour ces petits modèles, nous proposons une preuve inductive de ce protocole pour des anneaux de tailles quelconques.

La deuxième contributions de cette thèse porte sur la synthèse automatique de protocoles pour les réseaux de robots. Notre but est de montrer comment la synthèse pouvait être appliquée au réseaux de robots évoluant dans un environnement discret. Comme étude de cas nous nous sommes intéressées au problème du rassemblement, tout d'abord dans le modèle synchrone puis dans le modèle asynchrone. Nous avons proposé un encodage du problème de gathering par un jeu d'accessibilité avec deux joueurs. Les joueurs étant l'algorithme de robot d'une part et l'ordonnanceur étant son adversaire, ce dernier en plus de son rôle d'ordonnanceur décide le sens de l'orientation des robots à chaque activation. Notre encodage est assez fort pour englober les deux modèles synchrone Ssync et Fsync y compris lorsque plusieurs robots sont situés dans le même noeud et lorsque des situations symétriques se produisent, contrairement à la solution ad-hoc existante [BDP$^+$12]. Cela nous a permit de générer automatiquement un algorithme distribué *optimal*, pour trois robots évoluant sur des anneaux de tailles fixes. Notre critère d'optimalité se réfère au nombre de mouvements nécessaires pour que les robots se regroupent. En étudiant les algorithmes donnés nous avons pu extraire un *motif*, et en déduire un algorithme paramètré. Une preuve par induction nous a permis de prouver que cet algorithme est correct pour toute taille d'anneau dans le modèle synchrone mais aussi dans le modèle asynchrone.

Dans le cas asynchrone, nous montrons comment la recherche d'un algorithme de rassemblement de robots peut être vu comme un jeu avec informations partielles : une extension des jeux à deux joueurs. Dans ce type de jeux, contrairement aux précédents, les joueurs ont une vue incomplète du système. Afin de

lutter contre l'explosion combinatoire due au modèle asynchrone, nous proposons un algorithme récursif qui permet d'obtenir un protocole de rassemblement, grâce à la synthèse d'algorithmes dans le modèle synchrone combinée avec une utilisation dans le modèle asynchrone d'un model-checker sur les algorithmes produits par la synthèse.

Références

[ABB+12] José Bacelar Almeida, Manuel Barbosa, Endre Bangerter, Gilles Barthe, Stephan Krenn, and Santiago Zanella Béguelin. Full proof cryptography : verifiable compilation of efficient zero-knowledge protocols. In *ACM Conference on Computer and Communications Security, CCS'12*, pages 488–500, 2012.

[ABC+13] Cédric Auger, Zohir Bouzid, Pierre Courtieu, Sébastien Tixeuil, and Xavier Urbain. Certified impossibility results for byzantine-tolerant mobile robots. In *15th International Symposium on Stabilization, Safety, and Security of Distributed Systems - 15th International Symposium, SSS'13*, pages 178–190, 2013.

[AK86] Krzysztof R. Apt and Dexter Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 22(6) :307–309, 1986.

[AS85] Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21(4) :181–185, 1985.

[BDP+12] François Bonnet, Xavier Défago, Franck Petit, Maria Potop-Butucaru, and Sébastien Tixeuil. Brief announcement : Discovering and assessing fine-grained metrics in robot networks protocols. In *14th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'12)*, volume 7596 of *Lecture Notes in Computer Science*, pages 282–284. Springer, 2012.

[BL69] J. Richard Büchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Trans. Amer. Math. Soc.*, 138 :295–311, 1969.

[BMPT10] Lélia Blin, Alessia Milani, Maria Potop-Butucaru, and Sébastien Tixeuil. Exclusive perpetual ring exploration without chirality. In *24th International Symposium in Distributed Computing (DISC'10)*, volume 6343 of *Lecture Notes in Computer Science*, pages 312–327. Springer, 2010.

[CDD+08] A. Clerentin, M. Delafosse, L. Delahoche, B. Marhic, and A. Jolly-Desodt. Uncertainty and imprecision modeling for the mobile robot localization problem. *Autonomous Robots*, 24(3) :267–283, 2008.

[CE81] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *IBM Workshop on Logics of Programs*, 1981.

[Chu63] Alonzo Church. Logic, arithmetic, and automata. In *International Proceedings Congr. Mathematicians (Stockholm, 1962)*, pages 23–35. Inst. Mittag-Leffler, Djursholm, 1963.

[CJEF96] Edmund M. Clarke, Somesh Jha, Reinhard Enders, and Thomas Filkorn. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design*, 9(1) :77–104, 1996.

[CRTU15] Pierre Courtieu, Lionel Rieg, Sébastien Tixeuil, and Xavier Urbain. Impossibility of gathering, a certification. *Information Processing Letters*, 115(3) :447–452, 2015.

[DGMP06] Xavier Défago, Maria Gradinariu, Stéphane Messika, and Philippe Raipin Parvédy. Fault-tolerant and self-stabilizing mobile robots gathering. In *20th International Symposium on Distributed Computing, DISC'06*, volume 4167 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 2006.

[DLP+12] Stéphane Devismes, Anissa Lamani, Franck Petit, Pascal Raymond, and Sébastien Tixeuil. Optimal grid exploration by asynchronous oblivious robots. 7596 :64–76, 2012.

[DPT13] Stéphane Devismes, Franck Petit, and Sébastien Tixeuil. Optimal probabilistic ring exploration by semi-synchronous oblivious robots. *Structural Information and Communication Complexity*, 498 :10–27, 2013.

[DSN11] Gianlorenzo D'Angelo, Gabriele Di Stefano, and Alfredo Navarra. Gathering of six robots on anonymous symmetric rings. In *Proc. of 18th Int. Coll. on Structural Information and Communication Complexity (SIROCCO'11)*, volume 6796 of *Lecture Notes in Computer Science*, pages 174–185. Springer, 2011.

[DSN+13] Gianlorenzo D'Angelo, Gabriele Di Stefano, Alfredo Navarra, Nicolas Nisse, and Karol Suchan. A unified approach for different tasks on rings in robot-based computing systems. In *27th International Symposium on Parallel & Distributed Processing Workshops, IPDPSW'13*, pages 667–676. IEEE, 2013.

[ES96] E. Allen Emerson and A. Prasad Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9(1) :105–131, 1996.

[FIPS13] Paola Flocchini, David Ilcinkas, Andrzej Pelc, and Nicola Santoro. Computing without communicating : Ring exploration by asynchronous oblivious robots. *Algorithmica*, 65(3) :562–583, 2013.

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

[HCRP91] Nicolas Halbwachs, Paul Caspi, Pascale Raymond, and Daniel Pilaud. The synchronous dataflow programming language lustre. *Proceedings of the IEEE*, 79(9) :1305–1320, 1991.

[KAE+10] Gerwin Klein, June Andronick, Kevin Elphinstone, Gernot Heiser, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. sel4 : formal verification of an operating-system kernel. *Commun. ACM*, 53(6) :107–115, 2010.

[KKM10] Evangelos Kranakis, Danny Krizanc, and Euripides Markou. *The Mobile Agent Rendezvous Problem in the Ring*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool, 2010.

[KKN08] Ralf Klasing, Adrian Kosowski, and Alfredo Navarra. Taking advantage of symmetries : Gathering of asynchronous oblivious robots on a ring. *Theoretical Computer Science*, 5401 :446–462, 2008.

[KLK+12] S. Kamei, A. Lamani, S. Kamei, F. Ooshita, and Sébastien Tixeuil. Gathering an even number of robots in an odd ring without global multiplicity detection. In *37th International Symposiumon Mathematical Foundations of Computer Science, MFCS'12*, volume 7464 of *Lecture Notes in Computer Science*, pages 542–553. Springer, 2012.

[KLOT11] S. Kamei, A. Lamani, F. Ooshita, and Sébastien Tixeuil. Asynchronous mobile robot gathering from symmetric configurations without global multiplicity detection. In *18th International Colloquium on Structural Information and Communication Complexity, SIROCCO'11*, volume 6796 of *Lecture Notes in Computer Science*, pages 150–161. Springer, 2011.

[KMP06] Ralf Klasing, Euripides Markou, and Andrzej Pelc. Gathering asynchronous oblivious mobile robots in a ring. In *Algorithms and Computation, 17th International Symposium, ISAAC 2006, Kolkata, India, December 18-20, 2006, Proceedings*, volume 4288 of *Lecture Notes in Computer Science*, pages 744–753. Springer, 2006.

[Ler09] Xavier Leroy. A formally verified compiler back-end. *J. Autom. Reasoning*, 43(4) :363–446, 2009.

[LPT10] Anissa Lamani, Maria Gradinariu Potop-Butucaru, and Sébastien Tixeuil. Optimal deterministic ring exploration with oblivious asynchronous robots. In *Proc. of 17th Int. Coll. in Structural Information and Communication Complexity (SIROCCO'10)*, volume 6058 of *Lecture Notes in Computer Science*, pages 183–196. Springer, 2010.

[Lyn96a] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[Lyn96b] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[MW84] Zohar Manna and Pierre Wolper. Synthesis of communicating processes from temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 6(1) :68–93, 1984.

[Pel11] Andrzej Pelc. DISC 2011 invited lecture : Deterministic rendezvous in networks : Survey of models and results. In *Distributed Computing - 25th International Symposium, DISC 2011, Rome, Italy, September 20-22, 2011. Proceedings*, volume 6950 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2011.

[PR79] Gary L. Peterson and John H. Reif. Multiple-person alternation. In *20th Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 29-31 October 1979*, pages 348–363, 1979.

[PR90] Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *31st Annual Symposium on Foundations of Computer Science*, pages 746–757. IEEE Computer Society, 1990.

[Pre00] Giuseppe Prencipe. A new distributed model to control and coordinate a set of autonomous mobile robots : The corda model, 2000.

[Pre13] Giuseppe Prencipe. Autonomous mobile robots : A distributed computing perspective. In Paola Flocchini, Jie Gao, Evangelos Kranakis, and Friedhelm Meyer auf der Heide, editors, *Algorithms for Sensor Systems - 9th International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics, ALGOSENSORS 2013, Sophia Antipolis, France, September 5-6, 2013, Revised Selected Papers*, volume 8243 of *Lecture Notes in Computer Science*, pages 6–21. Springer, 2013.

[PRT11] Maria Potop-Butucaru, Michel Raynal, and Sébastien Tixeuil. Distributed computing with mobile robots : An introductory survey. In Leonard Barolli, Fatos Xhafa, and Makoto Takizawa, editors, *The 14th International Conference on Network-Based Information Systems, NBiS 2011, Tirana, Albania, September 7-9, 2011*, pages 318–324. IEEE Computer Society, 2011.

[SY99] Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots : Formation of geometric patterns. *SIAM Journal on Computing*, 28(4) :1347–1363, 1999.

[Tel01] Gerard Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, New York, NY, USA, 2nd edition, 2001.

[Val96] Antti Valmari. The state explosion problem. In Wolfgang Reisig and Grzegorz Rozenberg, editors, *Lectures on Petri Nets I : Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996*, volume 1491 of *Lecture Notes in Computer Science*, pages 429–528. Springer, 1996.

[VW86] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. pages 322–331, 1986.