Verification of Asynchronous Mobile-Robots in Partially-Known Environments*

Benjamin Aminof[1] and Aniello Murano[2] and Sasha Rubin[2] and Florian Zuleger[1]

[1] Technische Universität Wien
[2] Università degli Studi di Napoli "Federico II"

Abstract. This paper establishes a framework based on logic and automata theory in which to model and automatically verify that multiple mobile robots, with sensing abilities, moving asynchronously, correctly perform their tasks. The motivation is from practical scenarios in which the environment is not completely know to the robots, e.g., physical robots exploring a maze, or software agents exploring a hostile network. The framework shows how to express tasks in a logical language, and exhibits an algorithm solving the *parameterised* verification problem, where the graphs are treated as the parameter. The main assumption that yields decidability is that the robots take a bounded number of turns. We prove that dropping this assumption results in undecidability, even for robots with very limited ("local") sensing abilities.

1 Introduction

Autonomous mobile robots are designed to achieve some task in an environment without a central control. Foundational tasks include, for example, rendezvous (gather all robots in a single position) and reconfiguration (move to a new configuration in a collision-free way) [25, 14, 15]. This paper studies robots in *partially known* environments, i.e., robots do not have global information about the environment, but may know some (often topological) information (e.g., whether the environment is connected, or that it is a ring of some unknown size) [14]. The motivation for studying partially known environments is that in many practical scenarios the robots are unaware of the exact environment in which they are operating, e.g., mobile software exploring a hostile computer network, or physical robots that rendezvous in an environment not reachable by humans.

To illustrate, here is an *example reconfiguration problem.* Suppose that k robots find themselves on different internal nodes of a binary tree, and each robot has to reach a different leaf in a collision free way. Each robot can sense

* Benjamin Aminof and Florian Zuleger were supported by the Austrian National Research Network S11403-N23 (RiSE) of the Austrian Science Fund (FWF) and by the Vienna Science and Technology Fund (WWTF) through grant ICT12-059. Aniello Murano was supported by FP7 EU project 600958-SHERPA. Sasha Rubin is a Marie Curie fellow of the Istituto Nazionale di Alta Matematica.

if its left (or right) child is occupied by a robot. One protocol for solving this problem (assuming a large enough tree) is for each robot to execute 'go to the left child, then repeatedly go the right child' until a leaf is reached. Each move is guarded by a test that the child it wants to move to is not currently occupied.

As the example illustrates, we make the following modeling choices: environments are discrete (rather than continuous), robots have finite memory (rather than being oblivious, or being Turing powerful), robots are nondeterministic (rather than probabilistic), robots move asynchronously (rather than synchronously), and robots can sense the positions and the internal states of each robot no matter where they are, i.e., they can perform "remote" tests (but cannot leave information at visited positions).[3] The assumption that robots move asynchronously is motivated as follows: processes in distributed systems have no common notion of time since they may be running with different internal clocks, and thus a standard abstraction is to assume that processes' actions are interleaved, see [26]. There are two main ways to interleave the robots: an adversary tries to make sure that the robots do not succeed [16], and a co-operator tries to help the robots succeed (reminiscent of schedulers in strategic reasoning [7]).

In this paper we provide a framework for modeling and verifying that multiple mobile robots achieve a task (under adversarial and co-operative interleavings) in a partially-known environment. We now explain how we model the fact that environments are partially-known. Fix a class $\mathcal{G}$ of environments (e.g., $\mathcal{G}$ is the set of all lines, or all grids, or all rings). The *parameterised verification problem* states: Given robots $\overline{R}$, decide if they solve the task T on all graphs $G \in \mathcal{G}$. Requiring the robots to solve the task T on all graphs from a class $\mathcal{G}$ is how we model that the robots operate in partially-known environments — the robots know they are in a graph from $\mathcal{G}$, but they do not know which one. In contrast, the classic (non-parameterised) verification problem states: Given robots $\overline{R}$ and a graph $G \in \mathcal{G}$, decide if robots $\overline{R}$ solve the task T on G. In that setting, the robots can be designed to exploit the exact structure (size, etc.) of the graph.

Aims and Contributions. The aim of this work it to provide a formal framework in which one can reason (both mathematically and computationally) about multiple mobile-robots, with sensing abilities, moving asynchronously in a discrete, static, partially-known environment. We prove that parameterised verification is undecidable already for line-environments, and even for robots with limited tasks (e.g., halting) which can only detect collisions. I.e., a robot can only sense which other robots share its current position. This undecidability result also holds for robots that move synchronously. On the other hand, we prove that parameterised verification is decidable for scenarios in which the number of times that the robots take turns is bounded.[4] This decidability result is very robust:

[3] The ability to sense positions is a form of vision, while the ability to sense internal states is a form of communication

[4] In the example reconfiguration problem, there is some ordering of the robots that switches turns at most k times in which the stated robot-protocol succeeds. Also, for every ordering of the robots that switches turns a sufficiently large number of times, the protocol succeeds ("ordering" and "switching" are formalised in Section 3).

it holds on a very general class of graphs called *context-free sets of graphs* which include e.g., rings, trees, series-parallel graphs, but not grids; it also holds with very powerful abilities called *position-tests* which allow each robot to remotely test the positions of all the robots using formulas which include, e.g., the ability to test connectivity, as well as *state-tests* that allow each robot to remotely test the internal state of the other robots; it holds for tasks that are expressible using a new logic MRTL (Multiple-Robot Task Logic), which can express many natural tasks, e.g., reconfiguration, gathering, and "safe" variations (i.e., complete the task without touching dangerous positions).

Related Work. The work closest to ours is [30] which also considered the parameterised verification problem for multi-robot systems. However, in that paper, the decidability result (and the corresponding logic RTL) was only for one-robot systems (i.e., $k = 1$), and the undecidability result for $k = 2$ was for multiple robots that move *synchronously* and have *remote* tests.

The distributed-computing community has proposed and studied a number of models of robot systems, e.g., recently [24, 18, 10, 13, 8, 17]. This literature is mostly mathematical, and *theorems from this literature are parameterised*, i.e., they may involve graph-parameters (e.g., the maximum degree, the number of vertices, the diameter), memory parameters (e.g., the number of internal states of the robot protocol), and the number of robots may be a parameter. Only recently has there been emphasis on formal analysis of correctness of robots in a parameterised setting [22, 4, 27, 23, 30, 29]. In these formal analyses, typically it is the *number of agents* that is treated as the parameter [22, 4, 27, 23]. In contrast, in this paper (as in [30]) we apply formal methods to the parameterised verification problem in which it is the *environment that is parameterised*.

Also, the formal-verification community has placed emphasis on distributed models in which processes are stationary and interact by sending messages (e.g., in a broadcast or rendezvous manner) or by using guarded commands. The parameterised verification problem for such distributed processes is, in general, undecidable [31]. By simplifying the systems (e.g., restricting the mode of communication, the abilities of the processes, the specification languages, etc.) one can get decidable parameterised verification, recently e.g., [12, 1–3].

We refer the reader to [30, Section 7] for an up-to-date and detailed discussion of the connections between multi-robot systems, classic automata theory (i.e., graph-walking automata) and distributed systems (in particular, token-passing systems). Finally, we mention that there is a smattering of work on parameterised *synthesis* (called *generalised planning* in the AI literature) [11, 19–21].

2 Background: Automata Theory

Write B^* and B^ω for the sets of finite and infinite sequences over alphabet B, respectively. The empty sequence is denoted ϵ. Write $[n]$ for the set $\{1, 2, \cdots, n\}$.

Graphs and Trees. A *Σ-graph*, or *graph*, G, is a tuple (V, E, Σ, λ) where V is a finite set of *vertices*, $E \subseteq V \times V$ is the *edge relation*, Σ is a finite set of *edge*

labels, and $\lambda : E \to \Sigma$ is the *edge labeling function*. A *Δ-ary tree* (for $\Delta \in \mathbb{N}$) is a Σ-graph (V, E, Σ, λ) where (V, E) is a tree, $\Sigma = [\Delta] \cup \{up\}$, and λ labels the edge leading to the node in direction i (if it exists) by i, and the edge leading to the parent of a node (other than the root) is labelled by up. We may rename the labels for convenience, e.g., for binary trees ($\Delta = 2$) we let $\Sigma = \{lc, rc, up\}$ where lc replaces 1 and rc replaces 2.

Monadic Second-order Logic. Formulas are interpreted in Σ-graphs G. Define the set of monadic second-order formulas $\mathsf{MSOL}(\Sigma)$ as follows. Formulas of $\mathsf{MSOL}(\Sigma)$ are built using *first-order variables* $x, y, \cdots$ that vary over vertices, and *set variables* $X, Y, \cdots$ that vary over sets of vertices. The *atomic formulas* (when interpreted over Σ-graphs) are: $x = y$ (denoting that vertex x is the same as vertex y), $x \in X$ (denoting that vertex x is in the set of vertices X), $edg_\sigma(x, y)$ (denoting that there is an edge from x to y labeled $\sigma \in \Sigma$) and true (the formula that is always true). The formulas of $\mathsf{MSOL}(\Sigma)$ are built from the atomic formulas using the Boolean connectives (i.e., $\neg, \vee, \wedge, \to$) and variable quantification (i.e., $\forall, \exists$ over both types of variables). A variable that is not quantified is called *free*. The fragment of $\mathsf{MSOL}(\Sigma)$ which does not mention set variables is called *first-order logic*, denoted $\mathsf{FOL}(\Sigma)$. Write $\mathsf{MSOL}_k(\Sigma)$ for formulas with at most k many free first-order variables and no free set-variables. We abbreviate $z_1, \cdots, z_k$ by $\overline{z}$. We write $\phi(x_1, \cdots, x_k)$ to mean that the free variables from the formula ϕ are amongst the set $\{x_1, \cdots, x_k\}$ – note that the formula $\phi(x_1, \cdots, x_k)$ does not need to use all of the variables $x_1, \cdots, x_k$. For a graph G, and $v_1, \cdots, v_k \in V$, we write $G \models \phi(v_1, \cdots, v_k)$ to mean that ϕ holds in G with variable x_i simultaneously substituted by vertex v_i (for all $i \in [k]$ for which x_i occurs free in ϕ). Here are some examples of formulas and their meanings: The formula $\forall x(x \in X \to x \in Y)$ means that $X \subseteq Y$. Similarly, there are formulas for the set operations $\cup, \cap, =$, and relative complement $X \setminus Y$. The formula $edg(x, y) := \bigvee_{\sigma \in \Sigma} edg_\sigma(x, y)$ means that there is an edge from x to y (here Σ is assumed to be finite). The formula $E^*(x, y) := \forall Z[(closed_E(Z) \wedge x \in Z) \to y \in Z]$ where $closed_E(Z)$ is $\forall a \forall b[(a \in Z \wedge E(a, b)) \to b \in Z]$ defines the transitive closure of E. Generally, MSOL can express the 1-ary transitive closure operator (e.g., [30]).

The Validity Problem and Courcelle's Theorem. A *sentence* is a formula with no free variables. Let Φ be a set of sentences, and let $\mathcal{G}$ be a set of graphs. The *Φ-validity problem of* $\mathcal{G}$ is to decide, given $\phi \in \Phi$, whether for all graphs $G \in \mathcal{G}$, it holds that $G \models \phi$. Unfortunately, the $\mathsf{MSOL}(\Sigma)$-validity problem for the set $\mathcal{G}$ of all Σ-graphs is undecidable. However, Courcelle's Theorem states that MSOL-validity of context-free sets of graphs is uniformly decidable, i.e., there is an algorithm that given a description of a context-free set of graphs $\mathcal{G}$ and an MSOL-sentence ϕ decides if every graph in $\mathcal{G}$ satisfies ϕ [9]. Context-free sets of graphs are the analogue of context-free sets of strings, and can be described by graph grammars, equations using certain graph operations, or MSOL-transductions of the set of trees. Formally, $\mathcal{G}$ is *context-free* if it is MSOL-definable and of bounded clique-width [9]. Examples include, for a fixed alphabet, the set of labeled lines, rings, trees, series-parallel graphs, cliques, but not the set of grids.

Automata and Regular Expressions. Ordinary *regular-expressions* over a finite alphabet B are built from the sets $\emptyset$, $\{\epsilon\}$, and $\{b\}$ ($b \in B$), and the operations union $+$, concatenation $\cdot$, and Kleene-star *. Kleene's Theorem states that the languages definable by regular expressions over alphabet B are exactly those recognised by finite automata over alphabet B. An *ω-regular expression* over alphabet B is inductively defined to be of the form: exp^ω, $exp \cdot r$, or $r + r'$, where exp is an ordinary regular-expression over B, and r, r' are ω-regular expressions over B. An ω-regular language is one defined by an ω-regular expression. A variation of Kleene's Theorem says that the languages definable by ω-regular expressions over alphabet B are exactly the languages recognised by Büchi automata over alphabet B (which are like finite automata except they take infinite words as input, and accept if some accepting state occurs infinitely often).

3 The Model of Robot Systems

In this section we provide a framework for modeling multi-robot systems parameterised by their environment. Environments are modeled as Σ-graphs G and robots are modeled as regular languages of instructions. An instruction either tells the robot to move along an edge, or to test robot positions (e.g., a robot can learn which other robots are at the same vertex as it is, or if there is a robot north of it). Tests are formalised as logical formulas.

Instructions for Robots. Fix a number k of robots and a set of edge labels Σ. A *command* is a symbol from $\{\uparrow_\sigma : \sigma \in \Sigma\} \cup \{\circlearrowleft\}$. The command $\uparrow_\sigma$ tells the robot to move from its current vertex along the edge labeled σ, and the command $\circlearrowleft$ tells the robot to stay at its current vertex. A *position-test* is a formula from $\mathsf{MSOL}_k(\Sigma)$. A *state-test* is an expression of the form "robot i is in state q", where $i \in [k]$ and q is a symbol denoting a state of a robot (formally we may assume that all robots have states from $\mathbb{N}$, and that $q \in \mathbb{N}$)[5]. A position test $\tau(x_1, \cdots, x_k)$ allows the robot to test that $\tau(x_1, \cdots, x_k)$ holds in G, where x_i is the current vertex of robot R_i in G. Simple tests include "$x_i = x_j$" which tests if robots i and j are in the same vertex (i.e., collision detection), and "$edg(x_i, x_j) \vee edg(x_j, x_i)$" which tests if robots i and j are adjacent. A *test* is a state-test or a position-test. The *instruction set* $\textsc{ins}_{\Sigma,k}$ consists of all expressions of the form $\tau \to \kappa$ where τ is a test and κ is a command.

Robots, Configurations, Runs. A *k-robot ensemble* is a vector of robots $\langle R_1, \cdots, R_k \rangle$ where each *robot* $R_i = \langle Q_i, \delta_i \rangle$, each Q_i is a finite set of *states*, and each $\delta_i \subset Q_i \times \textsc{ins}_{\Sigma,k} \times Q_i$ is a finite *transition* relation. For technical convenience, we assume that robot i does not test its own state, i.e., no *ins* in a transition $(p, ins, q) \in \delta_i$ contains any occurrences of state-tests of the form "robot i is in state j". We designate a subset $I_i \subseteq Q_i$ of the states of robot i

[5] Note that, for ease of exposition, we do not explicitly allow Boolean combinations of state-tests. However, these can be indirectly performed by chaining state-tests (remembering the previous test results in the local state) to perform conjunctions, and using nondeterminism for disjunctions.

as *initial* states, and a subset $A_i \subseteq Q_i$ of its states as *accepting* states. A state $p \in Q_i$ is called *halting* if the only transition the robot has from p is (p, true, p). Thus we model a halting robot as one that forever stays in the same state and does not move. The halting states are denoted $H_i \subseteq Q_i$.

Fix a Σ-graph G. A *configuration* c of $\langle R_1, \cdots, R_k \rangle$ on graph G is a pair $\langle \overline{v}, \overline{q} \rangle \in V^k \times \prod_{i \in [k]} Q_i$. A configuration is *initial* if $\overline{q} \in \prod I_i$. For a test τ and a configuration $c = \langle \overline{u}, \overline{p} \rangle$, define $c \Vdash \tau$ to mean that configuration c makes τ true in G. Formally, if τ is a position test then define $c \Vdash \tau$ iff $G \models \tau(\overline{u})$, and if τ is a state-test, say "robot i is in state j", then define $c \Vdash \tau$ iff $p_i = j$. The following definition of $\vdash_i$ expresses that one configuration results from another after robot i successfully executes an instruction, while the rest are idle: for $i \in [k]$ and configurations $c = \langle \overline{w}, \overline{q} \rangle, d = \langle \overline{v}, \overline{p} \rangle$, write $c \vdash_i d$ if $p_j = q_j$ and $w_j = v_j$ for all $j \neq i$, and there exists a transition $(p_i, \tau \to \kappa, q_i) \in \delta_i$ (i.e., of robot R_i) such that $c \Vdash \tau$ (i.e., the current configuration satisfies the test τ) and, if $\kappa = \uparrow_\sigma$ then $\lambda(w_i, v_i) = \sigma$, and if $\kappa = \circlearrowleft$ then $w_i = v_i$.

Schedules and Runs. A *schedule* is a finite or infinite sequence $\mathcal{S} = s_1 s_2 s_3 \cdots$ where $s_i \in [k]$. A *run ρ of $\langle R_1, \cdots, R_k \rangle$ on G starting with an initial configuration c according to schedule $\mathcal{S}$* is a finite or infinite sequence $c_1 c_2 c_3 \cdots$ of configurations such that $c_1 = c$ and for all i, $c_i \vdash_{s_i} c_{i+1}$. The *set (resp. sequence) of positions of a run* $\alpha = \langle \overline{v}_1, \overline{p}_1 \rangle \langle \overline{v}_2, \overline{p}_2 \rangle \cdots$ is the set of positions $\{\overline{v}_1, \overline{v}_2, \cdots\}$ (resp. sequence $\overline{v}_1 \overline{v}_2 \cdots$ of positions) of its configurations. In a similar way define the *set (resp. sequence) of positions of robot i on a run.*

Orderings. A *(finite) k-ordering* is a string $\alpha \in [k]^+$, say of length $N+1$, such that $\alpha_i \neq \alpha_{i+1}$ for $1 \leq i \leq N$. Write $||\alpha|| = N$ to mean that $|\alpha| = N+1$, and say that α *is N-switching.* E.g., 171 is 2-switching. Say that a schedule $\mathcal{S}$ *follows* α if $\mathcal{S}$ is in ${\alpha_1}^* {\alpha_2}^* \cdots {\alpha_N}^* {\alpha_{N+1}}^*$ or ${\alpha_1}^* {\alpha_2}^* \cdots {\alpha_N}^* {\alpha_{N+1}}^\omega$, i.e., robot α_1 is scheduled for some (possibly no) time, then robot α_2 is scheduled, and so on, until α_{N+1} which can be scheduled forever. Similarly, an *infinite k-ordering* of k robots is a string $\alpha \in [k]^\omega$ such that $\alpha_i \neq \alpha_{i+1}$ for all $i \in \mathbb{N}$. In this case write $||\alpha|| = \infty$. A schedule *follows* α if the schedule is in the set ${\alpha_1}^* {\alpha_2}^* \cdots$.

Robot Tasks. Robots should achieve some task in their environment. We give some examples of foundational robot tasks [25]: A robot ensemble *deploys* or *reconfigures* if they move, in a collision-free way, to a certain target configuration. A robot ensemble *gathers* if, no matter where each robot starts, there is a vertex z, such that eventually every robot is in z. A robot ensemble *collaboratively explores* a graph if, no matter where they start, every node is eventually visited by at least one robot. All of these tasks have *safe* variations: the robots complete their task without entering certain pre-designated "bad" nodes.

Multi-Robot Task Logic — MRTL. We now define MRTL, a logic for formally expressing robot tasks. We first define the syntax and semantics, and then we give some example formulas. Later (in Lemma 1) we prove that, when restricted to bounded-switching orderings, MRTL formulas (and therefore many interesting natural tasks) can be converted into MSOL formulas over graphs.

MRTL Syntax. Fix $k \in \mathbb{N}$ and Σ. Formulas of MRTL_k are built, as in the definition of $\mathsf{MSOL}(\Sigma)$ from Section 2, from the following atomic formulas: $x = y$, edg_σ (for $\sigma \in \Sigma$), $x \in X$, true, and the following additional atomic formulas (with free variables $\overline{X}, \overline{x}, \overline{y}$ each of size k) $Reach_Q, Halt_Q^K, Infty_Q$ and $Rept_Q^K$ where $Q \in \{\exists, \forall\}$ and $\emptyset \neq K \subseteq [k]$. Denote by MRTL the set of formulas $\cup_k \mathsf{MRTL}_k$.

MRTL Semantics. Formulas of MRTL_k are interpreted over graphs G, and with respect to k-robot ensembles $\overline{R}$ and a set of orderings Ω. Define the satisfaction relation $\models_{\overline{R},\Omega}$:

- $G \models_{\overline{R},\Omega} Reach_\exists(\overline{X}, \overline{x}, \overline{y})$ iff there is an ordering $\alpha \in \Omega$ and there is a finite run of $\overline{R}$ on G that uses a schedule that follows α, such that the run starts with some initial configuration of the form $\langle \overline{x}, \overline{p} \rangle$ (i.e., $\overline{p} \in \prod I_i$), ends with a configuration of the form $\langle \overline{y}, \overline{q} \rangle$ (i.e., $\overline{q} \in \prod Q_i$), and for each $i \in [k]$, the set of positions of robot i on this run is contained in X_i.
- $G \models_{\overline{R},\Omega} Halt_\exists^K(\overline{X}, \overline{x}, \overline{y})$ means the same as $Reach_\exists$ except that the last tuple of states $\overline{q}$ has the property that $i \in K$ implies that $q_i \in H_i$ (i.e., every robot in K is in a halting state).
- $G \models_{\overline{R},\Omega} Infty_\exists(\overline{X}, \overline{x}, \overline{y})$ means the same as $Reach_\exists$ except that the run is infinite and, instead of ending in $\overline{y}$, it visits $\overline{y}$ infinitely often.
- $G \models_{\overline{R},\Omega} Rept_\exists^K(\overline{X}, \overline{x}, \overline{y})$ means the same as $Reach_\exists$ except that the run is infinite, and infinitely often it reaches a configuration of the form $\langle \overline{y}, \overline{q} \rangle$ such that $i \in K$ implies that q_i is an accepting state (i.e., $q_i \in A_i$).
- $G \models_{\overline{R},\Omega} Reach_\forall(\overline{X}, \overline{x}, \overline{y})$ is the same as $Reach_\exists$ except replace "there is an ordering $\alpha \in \Omega$ and there is a finite run..." by "for every ordering $\alpha \in \Omega$ there is a finite run ...". In a similar way, define $Halt_\forall^K, Infty_\forall$ and $Rept_\forall^K$.

Extend the satisfaction relation to all formulas of MRTL_k in the natural way.

Example 1. The statement $G \models_{\overline{R},\Omega} (\forall \overline{x})(\exists \overline{y})(\exists \overline{X}) Reach_\exists(\overline{X}, \overline{x}, \overline{y}) \wedge (\wedge_{i,j} y_i = y_j)$ means that, no matter where the robots start in G, there is an ordering $\alpha \in \Omega$, and a run according to a schedule that follows α, such that the robots $\overline{R}$ gather at some vertex of the graph G. Replacing $Reach_\exists$ by $Reach_\forall$ means, no matter where the robots start in G, for every ordering $\alpha \in \Omega$, the robots have a run according to a schedule that follows α such that the robots gather at a vertex of the graph. Note that by conjuncting with $\wedge_i X_i \cap B = \emptyset$ where B is an MSOL-definable set of "bad" vertices, one can express "safe gathering".

Example 2. Consider the statement $G \models_{\overline{R},\Omega} (\forall \overline{x})(\exists \overline{y})[\textsc{nonleaf}(\overline{x}) \wedge \textsc{diff}(\overline{x}) \rightarrow (\textsc{leaf}(\overline{y}) \wedge \textsc{diff}(\overline{y}) \wedge Reach_\forall(V^k, \overline{x}, \overline{y})]$ where G is a tree, $\textsc{nonleaf}(\overline{x})$ is an MSOL-formula expressing that every x_i is not a leaf, $\textsc{leaf}(\overline{y})$ is an MSOL-formula expressing that every y_i is a leaf, and $\textsc{diff}(\overline{z})$ is an MSOL-formula expressing that $z_i \neq z_j$ for $i \neq j$. The statement says that, as long as the robots start on different internal nodes of the tree G, for every ordering $\alpha \in \Omega$ there is a run of the robots $\overline{R}$ according to a schedule that follows α in which the robots reconfigure and arrive at different leaves.

4 Reasoning about Robot Systems

We formalise the parameterised verification problem for robot protocols and then study its decidability. The parameterised verification problem depends on a (typically infinite) set of graphs $\mathcal{G}$, a set of k-robot ensembles $\mathcal{R}$, a k-robot task written as an MRTL_k formula T, and a set of k-orderings Ω.

Definition 1. *The* ***parameterised verification problem*** $\mathsf{PVP}_{\mathrm{T},\Omega}(\mathcal{G},\mathcal{R})$ *is: given a robot ensemble* $\overline{R}$ *from* $\mathcal{R}$*, decide whether for every graph* $G \in \mathcal{G}$*,* $G \models_{\overline{R},\Omega} \mathrm{T}$ *(i.e., the robots* $\overline{R}$ *achieves the task* T *on* G *with orderings restricted to* Ω*).*

Example 3. Let $\mathcal{G}$ be the set of all binary trees, $\mathcal{R}$ be the set of all k-robot ensembles, let $\Omega_b := \{\alpha \in [k]^* : ||\alpha|| = b\}$ be the set of *b-switch orderings*, and let T be the task expressing that if the robots start on different internal nodes of a tree then they eventually reconfigure themselves to be on different leaves of the tree, no matter which ordering from Ω_b is chosen (cf. Example 2). We will see later that one can decide $\mathsf{PVP}_{\mathrm{T},\Omega_b}(\mathcal{G},\mathcal{R})$ given $b \in \mathbb{N}$. So, one can decide, given b, whether the protocol from the reconfiguration example (in the Introduction) succeeds for every ordering with b switches.

In Section 4.1 we show that the PVP is undecidable even on lines, for simple tasks, and allowing the robots very restricted testing abilities, i.e., a robot can sense which of the other robots shares the same position with it, called "local collision tests". In Section 4.2 we show that we can guarantee decidability merely by restricting the scheduling regime while allowing the robots full testing abilities, including testing positions and states of other robots "remotely".

4.1 Undecidability of Multi-Robot Systems on a Line

Our undecidability proof proceeds by reducing the halting problem of two counter machines to the parameterised verification problem. An *input-free 2-counter machine* (2CM) [28] is a deterministic program manipulating two nonnegative integer counters using commands that can increment a counter by 1, decrement a counter by 1, and check whether a counter is equal to zero. We refer to the "line numbers" of the program code as the "states" of the machine. One of these states is called the *halting state*, and once it is entered the machine *halts*. Observe that a 2CM has a single computation, and that if it halts then the values of both counters are bounded by some integer n. The *non-halting problem for 2CMs* is to decide, given a 2CM $\mathfrak{M}$, whether it does not halt. This problem is known to be undecidable [28], and is usually a convenient choice for proving undecidability of problems concerning parameterised systems due to the simplicity of the operations of counter machines.

Let $\mathcal{G}$ be the set of all graphs that are finite lines. Formally, for every $n \in \mathbb{N}$ there is a graph $L_n = (V_n, E_n, \Sigma, \lambda_n) \in \mathcal{G}$, where $\Sigma = \{l, r\}$, $V_n = [n]$, $E_n = \cup_{i<n}\{(i, i+1), (i+1, i)\}$, and the label λ_n of an edge of the form $(i, i+1)$ is r, and of the form $(i+1, i)$ is l. We now describe how, given a 2CM $\mathfrak{M}$, one

can construct a robot ensemble $\overline{R}$ which can, on long enough lines, simulate the computation of $\mathfrak{M}$. Our robots have very limited sensing abilities: a robot can only sense if it at one of the two ends of the line or not, and it can sense which of the other robots are in the same node as it is ("collision detection"). Note that a robot does not know that another robot has collided with it (and then moved on) if it is not scheduled while they both occupy the same node.

The basic encoding uses two counter robots C_1 and C_2. The current position of C_i on the line corresponds to the current value of counter i, and it moves to the right to increment counter i and to the left to decrement it. Each of these robots also stores in its finite memory the current state of the 2CM. One difficulty with this basic encoding is how to ensure that the two counter robots always stay synchronised in the sense that they both agree on the next command to simulate, i.e., we need to prevent one of them from "running ahead". A second difficulty is how to update the state of the 2CM stored by a counter robot when it simulates a command that is a test for zero of the other counter. Note that both of these difficulties are very easy to overcome if one robot can remotely sense the state/position of the other robot. Since we disallow such powerful sensing these difficulties become substantially harder to overcome. The basic idea used to overcome the first difficulty is to add synchronisation robots and have a counter robot move only if it has collided with the appropriate synchronization robot. Thus, by arranging that the synchronization robots collide with the counter robots in a round-robin way the latter alternate their simulation turns and are kept coordinated. In order to enforce this round-robin behavior we have to change the encoding such that only every other position on the line is used to encode the counter values. Thus, an increment or a decrement is simulated by a counter robot moving two steps (instead of one) in the correct direction. The basic ingredient in addressing the second difficulty is to add a *zero-test* robot that, whenever one counter is zero, moves to the position of the other counter's robot, thus signaling to it that the first counter is zero.

Theorem 1. *For every 2CM $\mathfrak{M}$, there is a robot ensemble $\overline{R}$ which, for every $n \geq 5$, simulates on the line L_n any prefix of the computation of $\mathfrak{M}$ in which the counters never exceed the value $(n-3)/2$.*

Proof. The ensemble $\overline{R}$ consists of 9 robots: the *counter* robots C_0, C_1, four *synchronisation* robots R_0, R_1, R_2, R_3, a *zero-test* robot T, a *zero* robot Z that marks the zero position of the counters, and a *mover* robot M whose role is to ensure that the robots can simulate more than one command of $\mathfrak{M}$ only if their starting positions on the line are as in the *initialised configuration* escribed below ((‡)). The value of a counter is encoded as half the distance between the corresponding counter robot and the Z robot (e.g., if Z is in node 3 and C_1 is in node 7 then the value of counter 1 is 2).

(‡) (*initialised configuration*): robots R_2, R_3 are in node 1, robots R_0, R_1 are in node 2, and the rest are in node 3.

The definition of the transitions of the robots has the important property that there is only one possible run starting from the initialised configuration,

i.e., at each point in time exactly one robot has exactly one transition with a test that evaluates to true. We assume that each robot remembers if it is at an odd or even node. This can be done even without looking at the node by storing the parity of the number of steps taken since the initialised configuration (‡).

Each command of the 2CM $\mathfrak{M}$ is simulated by the ensemble using 4 phases. For every $i \in \{0, 1, 2, 3\}$, phase i has the following internal stages: (1) the synchronization robots arrange themselves to signal to robot R_i that it can start moving to the right (this mechanism is described below ($\star$)). (2) robot R_i moves to the right until it collides with robot C_j (where $j = i \bmod 2$). It is an invariant of the run that this collision is at an even node if i is odd, and vice-versa. (3) robot C_j moves one step to the left or to the right, in order to simulate the relevant half of the current command of $\mathfrak{M}$, as described below (†). (4) robot R_i moves to the right until it reaches the end of the line. Observe that if during this stage R_i collides with C_j then (unlike in stage 2) it is on a node with the same parity as i (by the invariant, and since C_j moved one step in stage 3). This parity information is used by C_j to know that it should not move, and by R_i to know that it can continue moving to the right. (5) robot R_i moves to the left until it reaches the beginning of the line (see ($\star$)), which ends the phase (here, as in the previous stage, the parity information is used to ignore collisions with C_j). In case the other counter (i.e., counter $1 - j$) is zero, stage (2) of phases $0, 1$ are modified as follows: when robot R_i enters node 3 from the left it collides with Z, C_{1-j} and T; then, T and R_1 move to the right together, where T always goes first, and then R_i follows in lock-step; at the end of stage 2 both T and R_i collide with C_j, thus signalling to the latter that counter $1 - j$ is zero. A similar modification to stages (4) and (5) makes T and R_i move in lock-step fashion all the way to the right and then back to the left depositing T back in node 3.

(†): The operation performed by C_j in stage (3) of each phase is as follows. In phase 0 robot C_0 simulates the first half of the command, in phase 1 robot C_1 simulates the first half of the same command, in phase 2 robot C_0 simulates the second half of the command and in phase 3 C_1 does so. For example, if the command is "increment counter 0" then in phase 0 robot C_0 moves right one step (and updates its simulated state of $\mathfrak{M}$ to be the next command of $\mathfrak{M}$), in phase 1 robot C_1 moves right one step (and updates its simulated state of $\mathfrak{M}$), in phase 2 robot C_0 moves again one step to the right (thus encoding an incremented counter), and in phase 3 robot C_0 moves left one step (thus, returning to its previously encoded value). Simulating the other three increment and decrement commands is done similarly. The only other command we need to simulate is of the form "if counter i is zero goto state p else goto state q". Since this command does not change the value of any counter it is simulated by each counter robot going right in the first half of the simulation and left in the second half. The internal state of $\mathfrak{M}$ is updated to p or q depending on the value of the counter. When simulating the first half of the command, robot C_j knows that counter j (resp. $1 - j$) is zero iff it sees Z (resp. T) with it.

($\star$): We now show that every arrangement of the synchronization robots uniquely determines which one of them its turn it is to move. Let $\textsc{next}(i) :=$

$i+1 \bmod 4, \textsc{prev}(i) := i-1 \bmod 4$. An *initial arrangement for phase* i is of the following form: $R_{\textsc{prev}(\textsc{prev}(i))}, R_{\textsc{prev}(i)}$ are in node 1, and $R_i, R_{next(i)}$ are in node 2. Note that the initialised configuration (‡) contains the initial arrangement for phase 0. We let the initial arrangement for phase i signal that the next robot to move is $R_{\textsc{prev}(\textsc{prev}(i))}$, which moves to the right, thus completing stage (1) of phase i. Hence, at the beginning of stage 2 the arrangement is such that only $R_{\textsc{prev}(i)}$ is left in node 1, which signals that R_i is the next robot to move, as needed for stage (2). Just before the end of stage (5), robot R_i returns to node 2 from the left, and the above arrangement repeats itself. Hence, again it is R_i that moves, however, this time to the left (as indicated by its now different internal memory). The resulting arrangement at the end of phase i is thus: $R_{prev(i)}, R_i$ are in node 1 and $R_{\textsc{next}(i)}, R_{\textsc{prev}(\textsc{prev}(i))}$ are in node 2. Observe that this is exactly the initial arrangement for phase $\textsc{next}(i)$, as required. Note that since robots have collision tests a robot can tell by sensing which other robots are with it (and which are not) exactly which arrangement of the ones described above it is in, and thus if it is allowed to move or not.

Finally, we describe how to amend the construction above by incorporating the robot M to ensure that robots can only simulate the 2CM if they happen to begin in the initialised configuration (‡), and otherwise the system deadlocks after a few steps without any robot entering a halting state.[6] Add to every transition of robot R_i, for $i \in \{0,1,2,3\}$, the additional guard that M is on the same node with it. Thus, M enables the synchronisation robots to move, and if M ever stops, then so does the simulation. Robot M behaves as follows. It first verifies that the rest of the robots are in the initialised configuration by executing the following sequence (and stopping forever if any of the conditions in the sequence fail to hold): check that it is alone on the right-hand side of the line, move left until it collides with C_0, C_1, Z, T, move one step left and check that it collides with R_0, R_1, move one step left and check it is on the left-hand side of the line and collides with R_2, R_3. Once it verified that the robots are on the nodes specified by (‡), it starts "chasing after" the currently active synchronisation robot, i.e., it remembers which robot is active and the direction it moves in, and moves in that direction (if it does not currently collide with that robot). □

From the previous theorem we can easily deduce that $\mathfrak{M}$ halts iff there is a run of the ensemble $\overline{R}$ (on a long enough line, and that fully simulates the run of $\mathfrak{M}$) and in which the robots C_0, C_1 halt. We thus get:

Corollary 1. *Let $k = 9$, $\mathcal{G}$ be the set of lines, $\mathcal{R}$ the set of k-robot ensembles consisting of robots whose only tests are local collision tests and the ability to test the left (resp. right) end of the line, Ω the set of all k-orderings, and* T *the* MRTL *formula $(\forall \overline{x})(\forall \overline{y})(\forall \overline{X})\neg Halt_{\exists}^{\{1\}}(\overline{X}, \overline{x}, \overline{y})$.*[7] *Then $\mathsf{PVP}_{\mathrm{T},\Omega}(\mathcal{G}, \mathcal{R})$ is undecidable.*

[6] One can modify the construction to remove the need for the M robot, however we find the exposition with M clearer.

[7] The formula expresses"for every initial configuration, and every scheduling of the robots, robot 1 never enters a halting state".

Suitable changes to the construction in Theorem 1 yield that other tasks, such as "certain robots gather" or "certain robots reconfigure", are also undecidable.

Remark 1. Note that in the construction, starting from the initialised configuration, at most one robot can move at any time. Thus, allowing all robots that can act to act, as in the synchronous model, does not change anything. So, with minor modifications to deal with the initialisation phase, the theorem also holds for the synchronous model. This strengthens the previously known fact that the PVP is undecidable for synchronous robots on a line with remote testing abilities (i.e., robot l can test if "robots i and j are in the same node") [30].

4.2 Decidability of Multi-Robot Systems with Bounded Switching

The previous section shows that decidability cannot be achieved in very limited situations. However, we now suggest a limitation on the *orderings* which guarantees decidability without requiring any other restrictions. Thus it works on many classes of graphs, robots, and tasks. We first describe, at a high level, the approach we use to solve (restricted cases of) the parameterised verification problem $\mathsf{PVP}_{\mathrm{T},\Omega}(\mathcal{G},\mathcal{R})$, cf. [30]. Suppose we can build, for every k-ensemble $\overline{R}$ of robots, a formula $\phi_{\overline{R},\mathrm{T},\Omega}$ such that for all graphs $G \in \mathcal{G}$ the following are equivalent: i) $G \models \phi_{\overline{R},\mathrm{T},\Omega}$ and ii) $\overline{R}$ achieves task T on G with orderings restricted to Ω. Then, for every $\mathcal{R}$ and $\mathcal{G}$, we would have reduced the parameterised verification problem $\mathsf{PVP}_{\mathrm{T},\Omega}(\mathcal{G},\mathcal{R})$ to the $\Phi_{\mathcal{R},\mathrm{T},\Omega}$-validity problem for $\mathcal{G}$ where $\Phi_{\mathcal{R},\mathrm{T},\Omega}$ is the set of formulas $\{\phi_{\overline{R},\mathrm{T},\Omega} : \overline{R} \in \mathcal{R}\}$. We now show how to build an MSOL-formula $\phi_{\overline{R},\mathrm{T},\Omega}$ in case T is a formula of MRTL and Ω is a finite set of finite orderings.

We begin with a lemma that will be used as a building block. In the simplest setting, the lemma says that for every $i \in [k]$ there is an MSOL formula with free variables $\overline{x},\overline{y}$ that holds on a graph G if and only if robot i can move in G from x_i to y_i while all the other robots are frozen, i.e., $x_j = y_j$ for $j \neq i$.

Lemma 1 (From Robots to MSOL). *Fix k, and let $\overline{R}$ be a k-robot ensemble over instruction set $\mathrm{INS}_{\Sigma,k}$. For every $\overline{p},\overline{q} \in \prod Q_i$ (k-tuples of states) and ordering $\alpha \in [k]^+$, one can effectively construct an $\mathsf{MSOL}(\Sigma)$ formula $\psi_{\alpha,\overline{p},\overline{q}}(\overline{X},\overline{x},\overline{y})$ with free variables X_i, x_i, y_i ($i \in [k]$) such that for every graph G: $G \models \psi_{\alpha,\overline{p},\overline{q}}(\overline{X},\overline{x},\overline{y})$ if and only if there exists a run c of $\overline{R}$ on G according to a schedule that follows α, starting from configuration $c_1 = \langle \overline{x},\overline{p}\rangle$ and reaching, for some $T \in \mathbb{N}$, the configuration $c_T = \langle \overline{y},\overline{q}\rangle$, such that for all $i \in [k]$, the set of positions of robot i on $c_1 c_2 \cdots c_T$ is contained in X_i.*

Similarly one can construct $\psi^{\infty}_{\alpha,\overline{p},\overline{q}}(\overline{X},\overline{x},\overline{y})$ so that for every graph G: $G \models \psi^{\infty}_{\alpha,\overline{p},\overline{q}}(\overline{X},\overline{x},\overline{y})$ if and only if there exists a run c of $\overline{R}$ on G according to a schedule that follows α, starting from configuration $c_1 = \langle \overline{x},\overline{p}\rangle$ and reaching the configuration $\langle \overline{y},\overline{q}\rangle$ infinitely often, and such that the set of positions of robot i on the run is contained in X_i ($i \in [k]$).

Proof. Fix k and $\overline{R}$. We start with an auxiliary step. For $i \in [k]$, states $p_i, q_i \in Q_i$, and $\overline{s} = (s_1, \cdots, s_{i-1}, s_{i+1}, \cdots, s_k)$ with $s_j \in Q_j$, we define an MSO-formula

$\phi_{i,p_i,q_i,\overline{s}}$ with free variables $X, x, y, \overline{z}$ where $\overline{z} = (z_1, \cdots, z_{i-1}, z_{i+1}, \cdots, z_k)$ such that $G \models \phi_{i,p_i,q_i,\overline{s}}$ if and only if the k-ensemble robot $\overline{R}$ has a run according to a schedule in i^* in which robot i starts in position x and state p_i, and reaches position y and state q_i while only visiting vertices in X, and for $j \neq i$, robot j is in vertex z_j and state s_j and does not move or change state. This is done as follows. Each robot $R_i = \langle Q_i, \delta_i \rangle$ is a finite automaton (without initial or final states) over the finite alphabet $\text{INS}_{\Sigma,k}$. By Kleene's theorem, we can build a regular expression exp_i (that depends on $p_i, q_i, \overline{s}$) over $\text{INS}_{\Sigma,k}$ for the language of the automaton R_i with initial state p_i and final state q_i. By induction on the regular expressions we build MSOL formulas (with free variables $X, x, y, \overline{z}$):

- $\varphi_\emptyset :=$ false and $\varphi_\epsilon := x = y \wedge x \in X$,
- if $\tau \in \text{MSOL}_k(\Sigma)$ is a position-test then
  - $\varphi_{\tau \to \uparrow_\sigma} := \tau(z_1, \cdots, z_{i-1}, x, z_{i+1}, \cdots, z_k) \wedge edg_\sigma(x, y) \wedge x, y \in X$,
  - $\varphi_{\tau \to \circlearrowleft} := \tau(z_1, \cdots, z_{i-1}, x, z_{i+1}, \cdots, z_k) \wedge x = y \wedge x \in X$,
- if τ is a state-test, say "robot j is in state l" (for $j \neq i$ and $l \in Q_j$) then, if $s_j = l$ then
  - $\varphi_{\tau \to \uparrow_\sigma} := edg_\sigma(x, y) \wedge x, y \in X$,
  - $\varphi_{\tau \to \circlearrowleft} := x = y \wedge x \in X$,

  and otherwise if $s_j \neq l$, then $\varphi_{\tau \to \uparrow_\sigma}$ and $\varphi_{\tau \to \circlearrowleft}$ are defined to be false,
- $\varphi_{r+s} := \varphi_r \vee \varphi_s$,
- $\varphi_{r \cdot s} := \exists w \, [\varphi_r(X, x, w, \overline{z}) \wedge \varphi_s(X, w, y, \overline{z})]$,
- $\varphi_{r^*} := \forall Z [(cl_{\varphi_r}(X, Z, \overline{z}) \wedge x \in Z) \to y \in Z]$ where $cl_{\varphi_r}(X, Z, \overline{z})$ is defined as $\forall a, b \, [(a \in Z \wedge \varphi_r(X, a, b, \overline{z})) \to b \in Z]$.

Then, define $\phi_{i,p_i,q_i,\overline{s}}$ to be $\varphi_{exp_i}(X, x, y, \overline{z})$. To prove the lemma proceed by induction on the length l of α. Base case: For $\alpha = i \in [k]$, define $\psi_{i,\overline{p},\overline{q}}(\overline{X}, \overline{x}, \overline{y})$ by $\bigwedge_{j \neq i} x_j = y_j \wedge X_j = \{x_j\} \wedge \phi_{i,p_i,q_i,\overline{s}}(X_i, x_i, y_i, x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_k)$, where $\overline{s} = (p_1, \cdots, p_{i-1}, p_{i+1}, \cdots, p_k)$, if $q_j = p_j$ for all $j \neq i$, and otherwise the formula is defined as false. Inductive case: For $\alpha \in [k]^+, i \in [k]$, define $\psi_{\alpha \cdot i,\overline{p},\overline{q}}(\overline{X}, \overline{x}, \overline{y})$ by $\exists \overline{z} \bigvee_{\overline{r}} \left[\psi_{\alpha,\overline{p},\overline{r}}(\overline{X}, \overline{x}, \overline{z}) \wedge \psi_{i,\overline{r},\overline{q}}(\overline{X}, \overline{z}, \overline{y})\right]$ and $\overline{r}$ varies over $\prod Q_i$. This completes the construction of ψ_α. The construction of ψ_α^∞ is similar. □

In Lemma 1, a variable X_i designates a set containing – but not necessarily equal to – the positions of robot i along the run. If one wishes X_i to designate the exact set of positions visited by robot i (in order to express, e.g., "exploration"), then one needs to modify the construction of $\phi_{i,p_i,q_i,\overline{s}}$ in the proof of the lemma.[8] The required modifications are straightforward except for those to the definition of φ_{r^*}, which are more complicated.[9]

[8] In [30] it is wrongly stated that one can transform an MSOL formula that says that there is a run (satisfying some property) that stays within a set X, to one that says that it also visits all of X, by simply requiring that X be a minimal set for which a run satisfying the property exits.

[9] Recall that φ_{r^*} has free variables $X, x, y, \overline{z}$, and its semantic in this case is that robot i can reach y from x (with the other robots' positions being $\overline{z}$), visiting exactly X, using a concatenation of sub-paths each satisfying φ_r. Intuitively, φ_{r^*} existentially quantifies over the stitching points of these sub-paths and uses appropriate sub-formulas that are all satisfied iff one can find sub-paths that can be stitched to lead from x to y and that cover all the positions in X.

Observe that this lemma can be used to express both collaborative and adversarial scheduling. For instance, if Ω is a finite set of orderings, the formula $\bigvee_{\alpha\in\Omega}\psi_{\alpha,\overline{p},\overline{q}}(\overline{X},\overline{x},\overline{y})$ says that there is an ordering $\alpha\in\Omega$ that the robots can follow to go from $\overline{x}$ to $\overline{y}$ while staying in $\overline{X}$, i.e., the ordering is chosen collaboratively, while $\bigwedge_{\alpha\in\Omega}\psi_{\alpha,\overline{p},\overline{q}}(\overline{X},\overline{x},\overline{y})$ expresses that the ordering is chosen adversarially.

Putting everything together, we solve the PVP for finite sets of orderings (and thus for adversarial or co-operative b-switch orderings $\Omega_b := \{\alpha : ||\alpha|| = b\}$).

Theorem 2. *There is an algorithm that given an edge-label set Σ, a number of robots $k\in\mathbb{N}$, a formula* T *of* MRTL_k*, a finite set Ω of finite k-orderings, and a description of a context-free set of Σ-graphs $\mathcal{G}$, decides* $\mathsf{PVP}_{\mathrm{T},\Omega}(\mathcal{G},\mathcal{R})$*, where $\mathcal{R}$ is the set of all k-ensembles of robots over* $\mathrm{INS}_{\Sigma,k}$.

Proof. Given $\overline{R}\in\mathcal{R}$ build the formula $\phi_{\overline{R},\mathrm{T},\Omega}$ by replacing every atomic formula in T by its definition with respect to $\overline{R}$. E.g., $Reach_{\exists}(\overline{X},\overline{x},\overline{y})$ is replaced by $\bigvee_{\alpha\in\Omega}\bigvee_{\overline{p}}\bigvee_{\overline{q}}\psi_{\alpha,\overline{p},\overline{q}}(\overline{X},\overline{x},\overline{y})$, where $\overline{p}$ varies over $\prod_{i\in[k]}I_i$ and $\overline{q}$ varies over $\prod_{i\in[k]}Q_i$. Now, a routine induction on the structure of the formula T shows that $G\models_{\overline{R},\Omega}\mathrm{T}$ if and only if $G\models\phi_{\overline{R},\mathrm{T},\Omega}$. By Lemma 1 the formula $\phi_{\overline{R},\mathrm{T},\Omega}$ is in $\mathsf{MSOL}(\Sigma)$. Finally, apply the fact that the MSOL-validity problem for context-free sets of graphs $\mathcal{G}$ is uniformly decidable [9]. □

5 Discussion

In [6, 30] (see also the discussion before Theorem 1) it was shown that the PVP is undecidable for two synchronous robots on a line, reachability tasks, and allowing the robots "remote" position-tests. In Section 4.1 we substantially strengthen this result and prove that the problem is still undecidable even if we only allow robots "local" position-tests or even just local "collision tests", both for robots that move synchronously and asynchronously. The fact that the proof works for both the synchronous and asynchronous models (Remark 1), strongly suggests that limiting the robots' sensing capabilities may not be a very fruitful direction for decidability. In Section 4.2 we showed that for asynchronous robots, if one imposes a bound on the number of times the robots can switch, then PVP is decidable for very general tasks (i.e., those expressible in a new logic called MRTL), large classes of graphs (i.e., the context-free sets of graphs), and allowing robots very powerful testing abilities (i.e., MSOL position-tests *and* state-tests). This is the first parameterised decidability result of the PVP for *multiple* robots where the environment is the parameter. Thus, our work indicates that if practitioners want formal guarantees on the correctness of the robot protocols they design, then they could design them in the framework given in this paper (i.e., finite-state, bounded-switching, powerful testing abilities).

A main limitation of our decidability result is the fact that the set of grids is not context-free — grids are the canonical workspaces since they abstract 2D and 3D real-world scenarios. However, this limitation is inherent and not

confined to our formalisation since the parameterised verification problem even for one robot ($k = 1$) on a grid with only "local" tests is undecidable [6, 30]. A second limitation is that robots do not have a rich memory (e.g., they cannot remember a map of where they have visited). Extending the abilities to allow for richer memory and communication will result in undecidability, unless it is done in a careful way. Also, the complexity of the decision procedure we gave is very high. Again this is inherent in the problem since, e.g., already for one robot on trees the PVP with the "explore and halt" task is ExpTime-complete [30]. We leave for future research the problem of finding decidability results with reasonable complexity for multi-robot systems that are rich enough to capture protocols found in the distributed computing literature, e.g., [5, 24, 14, 15].

References

1. B. Aminof, S. Jacobs, A. Khalimov, and S. Rubin. Parameterized model checking of token-passing systems. In *VMCAI*, volume 8318 of *LNCS*, pages 262–281. Springer, 2014.
2. B. Aminof, T. Kotek, F. Spegni, S. Rubin, and H. Veith. Parameterized model checking of rendezvous systems. In *CONCUR*, volume 8704 of *LNCS*, pages 109–124. Springer, 2014.
3. B. Aminof, S. Rubin, F. Zuleger, and F. Spegni. Liveness of parameterized timed networks. In M. M. Halldórsson, K. Iwama, N. Kobayashi, and B. Speckmann, editors, *ICALP*, volume 9135 of *LNCS*, pages 375–387. Springer, 2015.
4. C. Auger, Z. Bouzid, P. Courtieu, S. Tixeuil, and X. Urbain. Certified impossibility results for byzantine-tolerant mobile robots. In *SSS*, volume 8255 of *LNCS*, pages 178–190. Springer, 2013.
5. M. A. Bender and D. K. Slonim. The power of team exploration: Two robots can learn unlabeled directed graphs. Technical report, MIT, 1995.
6. M. Blum and C. Hewitt. Automata on a 2-dimensional tape. *SWAT (FOCS)*, pages 155–160, 1967.
7. P. Čermák, A. Lomuscio, F. Mogavero, and A. Murano. Mcmas-slk: A model checker for the verification of strategy logic specifications. In *CAV*, volume 8559 of *LNCS*, pages 525–532. Springer, 2014.
8. R. Cohen, P. Fraigniaud, D. Ilcinkas, A. Korman, and D. Peleg. Label-guided graph exploration by a finite automaton. *T. on Algorithms (TALG)*, 4(4):42, 2008.
9. B. Courcelle and J. Engelfriet. Book: Graph structure and monadic second-order logic. a language-theoretic approach. *Bull. EATCS*, 108:179, 2012.
10. S. Das. Mobile agents in distributed computing: Network exploration. *Bull. EATCS*, 109:54–69, 2013.
11. G. De Giacomo, P. Felli, F. Patrizi, and S. Sardiña. Two-player game structures for generalized planning and agent composition. In M. Fox and D. Poole, editors, *AAAI*, pages 297–302, 2010.
12. G. Delzanno. Parameterized verification and model checking for distributed broadcast protocols. In *ICGT*, volume 8571 of *LNCS*, pages 1–16. Springer, 2014.
13. K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc. Tree exploration with little memory. *Journal of Algorithms*, 51(1):38–63, 2004.
14. P. Flocchini, G. Prencipe, and N. Santoro. Computing by mobile robotic sensors. In S. Nikoletseas and J. D. Rolim, editors, *Theoretical Aspects of Distributed Computing in Sensor Networks*, EATCS, pages 655–693. Springer, 2011.

15. P. Flocchini, G. Prencipe, and N. Santoro. *Distributed Computing by Oblivious Mobile Robots*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool, 2012.
16. P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Hard tasks for weak robots: The role of common knowledge in pattern formation by autonomous mobile robots. In *Algorithms and Computation*, volume 1741 of *LNCS*, pages 93–102. Springer, 1999.
17. P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science*, 345:331 – 344, 2005.
18. L. Gasieniec and T. Radzik. Memory efficient anonymous graph exploration. In H. Broersma, T. Erlebach, T. Friedetzky, and D. Paulusma, editors, *Graph-Theoretic Concepts in Computer Science*, volume 5344 of *LNCS*, pages 14–29. Springer, 2008.
19. Y. Hu and G. De Giacomo. Generalized planning: Synthesizing plans that work for multiple environments. In T. Walsh, editor, *IJCAI*, pages 918–923. AAAI, 2011.
20. A. Khalimov, S. Jacobs, and R. Bloem. PARTY parameterized synthesis of token rings. In N. Sharygina and H. Veith, editors, *CAV*, volume 8044 of *LNCS*, pages 928–933. Springer, 2013.
21. A. Khalimov, S. Jacobs, and R. Bloem. Towards efficient parameterized synthesis. In R. Giacobazzi, J. Berdine, and I. Mastroeni, editors, *VMCAI*, volume 7737 of *LNCS*, pages 108–127. Springer, 2013.
22. P. Kouvaros and A. Lomuscio. Automatic verification of parameterised multi-agent systems. In M. L. Gini, O. Shehory, T. Ito, and C. M. Jonker, editors, *AAMAS*, pages 861–868, 2013.
23. P. Kouvaros and A. Lomuscio. A counter abstraction technique for the verification of robot swarms. In B. Bonet and S. Koenig, editors, *AAAI*, pages 2081–2088, 2015.
24. E. Kranakis, D. Krizanc, and S. Rajsbaum. Mobile agent rendezvous: A survey. In P. Flocchini and L. Gasieniec, editors, *SIROCCO*, volume 4056 of *LNCS*, pages 1–9. Springer, 2006.
25. E. Kranakis, D. Krizanc, and S. Rajsbaum. Computing with mobile agents in distributed networks. In S. Rajasekaran and J. Reif, editors, *Handbook of Parallel Computing: Models, Algorithms, and Applications*, CRC Computer and Information Science Series, pages 8–1 to 8–20. Chapman Hall, 2007.
26. N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
27. L. Millet, M. Potop-Butucaru, N. Sznajder, and S. Tixeuil. On the synthesis of mobile robots algorithms: The case of ring gathering. In P. Felber and V. K. Garg, editors, *SSS*, volume 8756 of *LNCS*, pages 237–251. Springer, 2014.
28. M. L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., 1967.
29. A. Murano and L. Sorrentino. A game-based model for human-robots interaction. In *Workshop "From Objects to Agents" (WOA)*, volume 1382 of *CEUR Workshop Proceedings*, pages 146–150. CEUR-WS.org, 2015.
30. S. Rubin. Parameterised verification of autonomous mobile-agents in static but unknown environments. In G. Weiss, P. Yolum, R. H. Bordini, and E. Elkind, editors, *AAMAS*, pages 199–208, 2015.
31. I. Suzuki. Proving properties of a ring of finite-state machines. *Inf. Process. Lett.*, 28(4):213–214, July 1988.