# Parameterised Verification of Autonomous Mobile-Agents in Static but Unknown Environments

Sasha Rubin*
Università degli Studi di Napoli "Federico II"
Naples, Italy
sasha.rubin@gmail.com

## ABSTRACT

Automata walking on graphs are a mathematical formalisation of autonomous mobile agents with limited memory operating in discrete environments. This paper establishes a framework in which to model and automatically verify that autonomous mobile agents correctly perform their tasks. The framework consists of a logical language tailored for expressing agent tasks, and an algorithm solving the *parameterised* verification problem, where the graphs are treated as the parameter. We reduce the parameterised verification problem to classic questions in automata theory and monadic second order logic, i.e., universality and validity problems.

We illustrate the framework by instantiating it to a popular model of robot from the distributed computing literature.

This work clarifies the border between classes of mobile-agent systems that have decidable parameterised verification problem, and those that do not.

## Categories and Subject Descriptors

D.2.4 [**Software/Program Verification**]: Formal Methods; F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*Automata*; C.2.2 [**Computer Communication Networks**]: Network Protocols—*Protocol Verification*; C.2.4 [**Computer Communication Networks**]: Distributed Systems; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

## Keywords

Model Checking; Logic; Automata Theory; Distributed Networks; Autonomous Mobile Agents; Robots; Parameterised Verification

## 1. INTRODUCTION

Autonomous mobile agents are designed to achieve some task in an environment, e.g., exploration, or rendezvous. They are in an *unknown* environment if they do not have global information about the environment, e.g., mobile software exploring hostile computer networks, or physical robots that rendezvous in an environment not reachable by humans. This paper studies a mathematical formalisation of autonomous mobile agents in discrete environments, i.e., robots on finite graphs.

The distributed computing community has recently proposed and studied a number of models of robot systems [5, 33, 28, 18, 21, 16, 27]. 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.

However, until recently [31, 4, 35, 32] there has been little emphasis on formal analysis of correctness of robots in a parameterised setting. In this paper we apply formal methods to the parameterised verification problem in which it is the *environment that is parameterised*. This is how we model that robots operate in unknown environments. Formally, we address the following decision problem.

**Parameterised Verification Problem**: Given robots $\overline{R}$, decide if they solve the task T on all graphs $G \in \mathcal{G}$.

In contrast, the classic (non-parameterised) verification problem states:

**Verification Problem**: Given robots $\overline{R}$ and a graph $G \in \mathcal{G}$, decide if robots $\overline{R}$ solve the task T on $G$.

The non-parameterised verification problem is often handled using model-checking [14]. The parameterised verification problem corresponds to a family of model-checking problems (one for each $G \in \mathcal{G}$), or equivalently, a single infinite-state model-checking problem. A variety of techniques have been applied to such systems, fully discussed in Section 7 (Related Work), however none seem suited to the problem in which the environment is parameterised. Fortunately, the theory of automata and logic has produced a rich understanding of the expressive power of certain logics, e.g., monadic second-order logic MSOL, over certain classes of graphs, e.g., context-free sets of graphs [44, 45, 29, 17]. Our framework reduces the parameterised verification problem to classic questions in automata theory and MSOL, i.e., universality and validity problems. The key observations are that: i) robots are graph-walking automata, and can be defined in MSOL, and ii) robot tasks correspond to properties of runs of automata which are often definable in MSOL.

### *Modeling Choices.*

There are a number of different modeling choices for mo-

*Marie Curie fellow of the Istituto Nazionale di Alta Matematica.

**Appears in:** *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum (eds.), May 4–8, 2015, Istanbul, Turkey.*

bile agents [38]. We list them, and emphasise (using underlining) the choices made in this paper: (i) is the environment continuous (e.g., the plane) or discrete (e.g. a finite graph whose edges are labeled by directions or port numbers)? (ii) do agents act synchronously or asynchronously? (iii) are agents probabilistic or non-deterministic? (iv) is there one agent or are there multiple agents, and are the number of agents known or unknown? (v) is the environment static, or can agents affect their environment (e.g., by marking the nodes of a graph)? Moreover, (vi) how much information about the environment is known, a priori, to the agents? We assume agents do not have global knowledge of the environment, and in particular the size is not known and nodes in the environment do not have unique identifiers. And finally, (vii) how do agents communicate and sense their environment? We assume agents can *sense their positions in the graph* as well as those of the other agents (in the case of multiple agents), i.e., robots acquire information about the current state of the environment solely by vision (we use logical formulas, which we call tests, to define these sensing abilities).

### *Aim and Contributions.*

The main aim of this work it to provide a clearer understanding of the border between classes of autonomous mobile-agent systems that have decidable parameterised verification problem, and those that do not.

In particular, we provide a quite general formalisation of robot systems (using the modeling choices above) in which all components (i.e., environments, robots, and tasks) are modeled separately. We formalise reasoning about robot systems as the parameterised verification problem (PVP) where the environment is treated as a parameter. We show how to reduce the PVP to problems about automata and logic, i.e., universality and validity problems. We then prove that the PVP is decidable for suitable restrictions of the components, notably the case of a single robot on context-free sets of graphs. We illustrate the power of the framework by instantiating it to a model of robot systems from the distributed computing literature. This instantiation falls into our decidable restrictions.

## 2. BACKGROUND: AUTOMATA THEORY

In order to make this paper self-contained, this section contains the necessary background. In particular, we will use basic notions from mathematical logic and automata theory to formalise the robot systems (a full logical background can be found in [22], and a smaller discussion of monadic second-order logic $\mathsf{MSOL}$ over graphs can be found in [17, Section 1.3.1]), and easy-to-state theorems (e.g., Kleene's Theorem about the equivalence of regular expressions and automata, and Courcelle's Theorem on satisfiability of $\mathsf{MSOL}$ over context-free sets of graphs [17]) to give an algorithm for solving the parameterised verification problem.

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

### *Graphs.*

A $\Sigma$-*graph*, or *graph*, $G$, is a tuple $(V, E, \Sigma, \lambda)$ where $V$ is a finite set of *vertices*, $E \subseteq V \times V$ is a relation called the *edge relation*, $\Sigma$ is a finite set of *edge labels*, and $\lambda : E \to \Sigma$ is the *edge labeling function*. The *out-degree* of a vertex $v$, written $\mathsf{deg}(v)$, is the cardinality of the set $\{(v, w) \in E : w \in V\}$ of *outgoing edges* of $v$.

Sometimes the edge labels give a sense of direction:

EXAMPLE 1. *An* $L$-labeled grid *is a graph with* $V = [n] \times [m]$, $\Sigma = L \times \{N, S, E, W\}$, *and labels* $\lambda((x, y), (x + 1, y)) \in L \times \{E\}$, *etc., where* $L$ *is a finite set and* $n, m \in \mathbb{N}$. *An* $L$-labeled line *is an* $L$*-labeled grid with* $V = [n] \times [1]$. *If* $|L| = 1$ *then we call the grid (or line)* unlabeled.

EXAMPLE 2. *A* $\Delta$-ary tree *(for* $\Delta \in \mathbb{N}$*) is a* $\Sigma$*-graph* $(V, E, \Sigma, \lambda)$ *where* $(V, E)$ *is a tree,* $\Sigma = [\Delta] \cup \{up\}$, *and* $\lambda$ *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.

### *First-order and Monadic Second-order Logic.*

Formulas will be interpreted in $\Sigma$-graphs $G$ (an exception are formulas in Section 4.3). 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 $\Sigma$-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 $X$), and $edg_\sigma(x, y)$ (denoting that there is an edge from $x$ to $y$ labeled $\sigma \in \Sigma$). 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). 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 $k$-many free variables.

Here are some examples of formulas and their meanings:

MF1. 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 $X \setminus Y$.

MF2. The formula $edg(x, y) := \bigvee_{\sigma \in \Sigma} edg_\sigma(x, y)$ means that there is an edge from $x$ to $y$ (here $\Sigma$ is assumed to be finite).

MF3. The formula $\exists x \exists y(x \neq y \wedge edg(z, x) \wedge edg(z, y))$ means that $\mathsf{deg}(z) \geq 2$.

MF4. If $\phi(x, y)$ is an $\mathsf{MSOL}(\Sigma)$ formula then define $\phi^*(x, y)$ by $\forall Z[(closed_\phi(Z) \wedge x \in Z) \to y \in Z]$. Here $closed_\phi(Z)$ is defined as $\forall a \forall b[(a \in Z \wedge \phi(a, b)) \to b \in Z]$. Note that $\phi^*$ is an $\mathsf{MSOL}(\Sigma)$-formula.

Note that $\phi^*(x, y)$ holds in a graph $G$ if and only if there exists a finite sequence of vertices $v_1 v_2 \cdots v_m \in V^*$ (here $m \geq 1$) such that $x = v_1, y = v_m$ and $\phi(v_i, v_{i+1})$ holds in $G$ for all $i < m$. This shows that that if a binary relation is $\mathsf{MSOL}$-definable, then so is its transitive-closure.

Similarly, define $\phi^\omega(x, y) := \phi^*(x, y) \wedge \exists z(\phi(y, z) \wedge \phi^*(z, y))$. Note $\phi^\omega(x, y)$ holds in a graph $G$ if and only if there is an infinite sequence of vertices $v_1 v_2 \cdots \in V^\omega$ with $x = v_1$, $v_i = y$ for infinitely many $i \in \mathbb{N}$, and $\phi(v_i, v_{i+1})$ holds in $G$ for all $i \in \mathbb{N}$. This uses the fact that $V$ is finite.

MF5. The *k-ary transitive-closure operator* is the function that maps a $2k$-ary relation $\rho(\overline{x},\overline{y})$ to the $2k$-ary relation $\rho^*(\overline{x},\overline{y})$ such that, in every graph $G$: $\rho^*(\overline{x},\overline{y})$ holds if and only if there exists a sequence $\overline{v}_1,\overline{v}_2,\cdots,\overline{v}_m$ such that $\overline{x}=\overline{v}_1$, $\overline{y}=\overline{v}_m$, and $\rho(\overline{v}_i,\overline{v}_{i+1})$ holds for every $i<m$. For $k>1$, it is not the case that if a $k$-ary relation $\phi$ is MSOL-definable, then so is its $k$-ary transitive-closure, because, intuitively, this would require having $k$-ary relation variables $Z$.[1]

For a formula $\phi(x_1,\cdots,x_k)$, a graph $G$, and $v_1,\cdots,v_k\in V$, write $G\models\phi(v_1,\cdots,v_k)$ to mean that $\phi$ holds in $G$ with variable $x_i$ substituted by vertex $v_i$ (for $i\le k$).

### *The Validity Problem and Courcelle's Theorem.*

A *sentence* is a formula with no free variables. Let $\Phi$ be a set of sentences, and let $\mathcal{G}$ be a set of graphs. The *$\Phi$-validity problem of $\mathcal{G}$* is to decide, given $\phi\in\Phi$, whether or not for all graphs $G\in\mathcal{G}$, it holds that $G\models\phi$. Unfortunately for us, the FOL($\Sigma$)-validity problem for $\mathcal{G}=$ the set of all $\Sigma$-graphs is undecidable [22]. On the other hand, many interesting cases have decidable validity. One version of Courcelle's Theorem states that the MSOL-validity problem is decidable for every context-free set of graphs $\mathcal{G}$ [17].[2] Context-free sets of graphs are the analogue of context-free sets of strings, and are generated by graph grammars or equations using certain graph operations. Examples include, for a fixed alphabet, the set of labeled lines, the set of rings, the set of trees, the set of series-parallel graphs, the set of cliques, but not the set of all grids.

### *Automata and Regular Expressions.*

Ordinary *regular-expressions* over a finite alphabet $B$ are built from the sets $\emptyset$, $\{\epsilon\}$, and $\{b\}$ for $b\in B$, and the operations union $+$, concatenation $\cdot$, and Kleene-star $^*$. Kleene's Theorem states that the languages recognised by regular expressions over alphabet $B$ are exactly those recognised by finite automata over alphabet $B$.

An *$\omega$-regular expression* over the finite alphabet $B$ is inductively defined to be of the form: $exp^\omega$, $exp\cdot r$, or $r+r'$, where $exp$ is an ordinary regular-expression over $B$, and $r,r'$ are $\omega$-regular expressions over $B$. An $\omega$-regular language is one defined by an $\omega$-regular expression. A variation of Kleene's Theorem says that the languages recognised by $\omega$-regular expressions over alphabet $B$ are exactly the languages recognised by Büchi automata over alphabet $B$ (which are like non-deterministic finite automata except that they take infinite words as input, and accept if some accepting state occurs infinitely often).

## 3. THE MODEL OF ROBOT SYSTEMS

We provide a framework for modeling multi-robot systems parameterised by their environment.

[1]To prove this formally, note that 2-ary transitive closure on finite words (i.e., binary-labeled lines) can define the non-regular language $\{0^n1^n : n\in\mathbb{N}\}$; now use the fact that, over finite words, MSOL can only define the regular languages, part of a result known as the Büchi-Elgot-Trakhtenbrot Theorem, see [45].

[2]Actually, a strong form of the theorem states that there is an algorithm that given a description of a context-free set of graphs $\mathcal{G}$ and an MSOL-formula $\phi$ decides if every graph in $\mathcal{G}$ satisfies $\phi$.

## 3.1 The model of robot systems

In this section we define robot systems, i.e., robot protocols, environments, and tasks.

Environments are modeled as $\Sigma$-graphs,[3] and robots are modeled as regular languages of instructions, as in [7]. An instruction either tells the robot to move along an edge, or it allows the robot to test the positions of the robots (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. As discussed in Section 5, our model is general enough to be able to express a popular model of robot systems found in the distributed computing literature [27, 21, 16, 33, 28, 18].

We first define robot systems consisting of a single robot, and then define multi-robot systems.

### *Robot Instruction Set.*

A robot follows instructions from the *instruction set*

$$\text{INS}_{\Sigma,1} := \{\uparrow_\sigma : \sigma\in\Sigma\}\cup\text{MSOL}_1(\Sigma).$$

Instructions are of two kinds, *moves* and *tests*:[4] $\uparrow_\sigma$ tells the robot to move from its current vertex along the edge labeled $\sigma$; and $\text{MSOL}_1(\Sigma)$ consists of formulas $\tau(x)$ that allow the robot to test that $\tau(x)$ holds in $G$, where $x$ is the current vertex of the robot in $G$. Typical tests include: is the degree of the current vertex at least $d$? is there an edge out of the current vertex labeled $\sigma$? Here is a more complex test: is there a path from the current vertex that only uses edges labeled $\sigma$ to a vertex of degree $d$?

For a sequence $ins\in(\text{INS}_{\Sigma,1})^*$, write $[[ins]]_G\subseteq V^2$ for the set of pairs of vertices $(u,v)$ such that, in $G$, one can reach $v$ from $u$ by following the instructions $ins$. Formally,

1. $(u,v)\in[[\epsilon]]_G$ iff $u=v$,
2. $(u,v)\in[[\uparrow_\sigma]]_G$ iff $\lambda(u,v)=\sigma$,
3. $(u,v)\in[[\tau(x)]]_G$ iff $u=v$ and $G\models\tau(u)$, and
4. $(u,v)\in[[d\cdot e]]_G$ for $d\in(\text{INS}_{\Sigma,1})^*$ and $e\in\text{INS}_{\Sigma,1}$ iff there exists $z\in V$ such that $(u,z)\in[[d]]_G$ and $(z,v)\in[[e]]_G$.

### *Robot Protocols.*

Fix a set of edge-labels $\Sigma$. A 1-*robot*, or *robot*, $R$, is a pair $(Q,\delta)$ where $Q$ is a finite set of *states*, and $\delta\subset Q\times\text{INS}_{\Sigma,1}\times Q$ is a finite *transition* relation.

We sometimes designate certain states of the robot to be *initial*, denoted $I\subseteq Q$ and certain states to be *repeating*, denoted $A\subseteq Q$. A state $p\in Q$ is called *halting* if the robot has the transition $(p,\mathsf{true},p)$, i.e., state $p$ is a sink. Thus we model a halting robot as one that stays in the same state forever. The set of halting states is denoted $H\subseteq Q$.

EXAMPLE 3. *The robot in Figure 1 does a (perpetual) depth-first search of trees in which every node has zero or two children.*[5] *Tests are written $leaf, lc, rc, root$ (whose meanings are "is the current node a leaf?", "left-child?", "right-child?", "the root?"), and move instructions are written $\uparrow_{up}$,*

[3]The fact that $\Sigma$-graphs are finite corresponds to our modeling assumption that environments are *discrete*.

[4]Looking at the instruction set we see that robots cannot alter the graph, i.e., the environment is *static*.

[5]To make the diagram readable, an edge may be labeled by multiple successive actions separated by semicolons.

$\uparrow_{lc}$, *and* $\uparrow_{rc}$ *(whose meanings are "move up to the parent", "to the left-child", "to the right-child").*

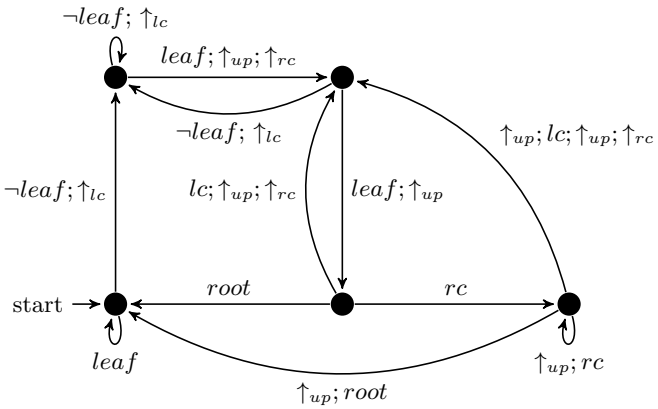


**Figure 1: Robot that does a DFS on binary trees.**

### *Configurations and Runs.*

A robot walks about a $\Sigma$-graph $G$. Let $R = (Q, \delta)$ be a robot with instruction set $\text{INS}_{\Sigma,1}$. A *configuration* $c$ of $R$ on $G$ is a pair $\langle v, q\rangle \in V \times Q$. Say that the configuration $\langle v, q\rangle$ has *position* $v$. A configuration is *initial* (resp. *halting, repeating*) if $q$ is. The following definition expresses that one configuration results from another after the robot executes a single instruction: for configurations $c = \langle v, p\rangle$ and $d = \langle w, q\rangle$, say that $d$ *results from* $c$ iff there is a transition $(p, ins, q)$ of $\delta$ such that $(v, w) \in [[ins]]_G$. A *run* $\alpha$ of $R$ on $G$ starting with an initial configuration $c$ is an infinite sequence $c_1c_2c_3\cdots$ of configurations such that $c_1 = c$ and for all $i$, $c_{i+1}$ results from $c_i$.

The *set of positions of a run* $\alpha = (v_1, q_1)(v_2, q_2)\cdots$ is the set of positions $\{v_1, v_2, \cdots\}$ of its configurations. The *sequence of positions of a run* $\alpha$ is the sequence of positions $v_1v_2\cdots$ of its configurations.

### *Multi-Robot Systems.*

We extend the previous definitions to $k$-many of robots.

A *$k$-robot ensemble* is a sequence $\langle R_1, \cdots, R_k\rangle$ where, writing $R_i = (Q_i, \delta_i)$, each $Q_i$ is a finite set of *states*, and each $\delta_i \subset Q_i \times \text{INS}_{\Sigma,k} \times Q_i$ is a finite *transition* relation, where the *instruction set* is $\text{INS}_{\Sigma,k} := \{\uparrow_\sigma : \sigma \in \Sigma\} \cup \mathsf{MSOL}_k(\Sigma)$. For a sequence $ins \in ((\text{INS}_{\Sigma,k})^k)^*$ define[6] $[[ins]]_G \subseteq V^{2k}$ such that $(\overline{u}, \overline{v}) \in [[ins]]_G$ if and only if, in $G$, one can reach $\overline{v}$ from $\overline{u}$ by following the instructions in $ins$.

Formally,

1. If $ins = \epsilon$, then $(\overline{u}, \overline{v}) \in [[ins]]_G$ if and only if $\overline{u} = \overline{v}$;
2. If $ins = (d_1, \cdots, d_k) \in \text{INS}_{\Sigma,k}$ then $(\overline{u}, \overline{v}) \in [[ins]]_G$ if, for each $i \leq k$:
   (a) if $d_i = \uparrow_\sigma$ then $\lambda(u_i, v_i) = \sigma$,
   (b) if $d_i = \tau(x_1, \cdots, x_k)$ then $u_i = v_i$ and $G \models \tau(u_1, \cdots, u_k)$.
3. If $ins = d \cdot e$ then $(\overline{u}, \overline{v}) \in [[ins]]_G$ if and only if there exists $\overline{z} \in V$ such that $(\overline{u}, \overline{z}) \in [[d]]_G$ and $(\overline{z}, \overline{v}) \in [[e]]_G$.

Fix a $\Sigma$-graph $G$. A *configuration* $c$ of $\langle R_1, \cdots, R_k\rangle$ on graph $G$ is a pair $\langle \overline{v}, \overline{q}\rangle \in V^k \times \prod_{i \leq k} Q_i$. A configuration is *initial* if $q_i$ is the initial state of robot $R_i$ (for all $i \leq k$). The following definition expresses that one configuration results from another after the robots simultaneously execute their own next instruction (which may be to move along an edge labeled $\sigma$, or to test the current position of all the robots):[7] configuration $\langle \overline{w}, \overline{q}\rangle$ *results from* $\langle \overline{v}, \overline{p}\rangle$ iff there are transitions $(p_i, ins_i, q_i) \in \delta_i$ (for $i \leq k$) such that $(\overline{v}, \overline{w}) \in [[(ins_1, \cdots, ins_k)]]_G$. *Runs* and their *sets of positions and sequences* are defined as before.

### *Robot Tasks.*

Robots should achieve some task in their environment: a *$k$-robot task*, or simply a *task*, T, is a function that maps a graph $G$ to a set of sequences of positions of $G$, i.e., $\text{T}(G) \subseteq (V^k)^\omega$. A robot-ensemble $\overline{R}$ *achieves* T on $G$ if for every run $\alpha$ of $\overline{R}$ on $G$ it holds that $\alpha' \in \text{T}(G)$, where $\alpha'$ is the sequence of positions of the run $\alpha$.

We give some examples of foundational robot tasks [34]:

RT1. A robot *explores and halts* if, no matter where it starts, it a) eventually halts, and b) visits every vertex of the graph at least once.

RT2. A robot *perpetually explores* if, no matter where it starts, it visits every vertex of $G$ infinitely often.

RT3. A robot *explores and returns* if, no matter where it starts, it a) eventually stops where it started, and b) visits every vertex of the graph at least once.

RT4. An ensemble of robots *collaboratively explores* a graph if, no matter where they start, every node is eventually visited by at least one robot.

RT5. A $k$-robot ensemble *gathers* if, no matter where each robot starts, there is a vertex $z$, such that eventually every robot is in $z$.

# 4. REASONING ABOUT ROBOT SYSTEMS

We formalise the parameterised verification problem for robot protocols, and then describe our solution to it (Theorem 2) which shows how to reduce parameterised verification in the case of a single robot to the logical validity problem of certain logics.

## 4.1 The Parameterised Verification Problem

The parameterised verification problem depends on a set of graphs $\mathcal{G}$, a set of robot ensembles $\mathcal{R}$, and a robot task $T$. Note that $\mathcal{G}$ is typically infinite.

DEFINITION 1. *The* ***parameterised verification problem*** $\mathsf{PVP}_{\text{T}}(\mathcal{G}, \mathcal{R})$ *states: given a robot ensemble* $\overline{R}$ *from* $\mathcal{R}$*, decide whether or not* $\overline{R}$ *achieves the task* T *on every graph* $G \in \mathcal{G}$.

Unfortunately, this problem is easily undecidable for the types of systems we have:

[6]To improve readability, the notation $[[\ ]]_G$ does not mention $k$.

[7]This is where we model the assumption that robots act *synchronously.*

THEOREM 1. *There exists a task* T *and computable sets* $\mathcal{G}$ *and* $\mathcal{R}$ *such that* $\mathsf{PVP}_{\mathrm{T}}(\mathcal{G},\mathcal{R})$ *is undecidable.*

*In particular, we can choose* $k=1$, T *to be the task "never halt",* $\mathcal{G}$ *to be the set of unlabeled-grids, and* $\mathcal{R}$ *to be all robots; or we can choose* $k=2$, T *to be the task "no robot ever halts",* $\mathcal{G}$ *to be the set of unlabeled-lines, and* $\mathcal{R}$ *to be all robots.*

PROOF SKETCH. Since the proof technique is a standard, we merely sketch it. We reduce the non-halting problem for (Turing-powerful) two-counter machines to $\mathsf{PVP}_{\mathrm{T}}(\mathcal{G},\mathcal{R})$, i.e., given machine $M$ build robot(s) $\overline{R}$ such that $M$ accepts no input if and only if $\overline{R}$ achieves task T on all graphs in $\mathcal{G}$.

Case $k=1$: As observed by [8] for their "2-dimensional automata", one robot on a grid can simulate a two-counter machine: counter values $(n,m)\in\mathbb{N}^2$ are encoded by the robot being at position $(n,m)$ of the grid. The only tests that are needed are to check whether or not the robot is on the boundary (this is to simulate the counter machine's "test for zero", as well as to alert about a counter overflow).

Case $k=2$: The idea is that two distinguishable robots on a line can simulate one robot on a square grid; the position of the $i$th robot on the line gives the $i$th co-ordinate of the single robot on the grid. The robots only need to be able to test if they are at the end points of the line. Alternatively, one can directly code computations of Turing machines, as was done by [40] to show that universality of 2-head one-way finite-state automata is undecidable. □

In light of this negative result, our main task is to understand in what way we can restrict the systems to get decidability.

## 4.2 Reducing Parameterised Verification to Validity

We first describe, at a high level, the approach we use to solve (restricted cases of) the parameterised verification problem $\mathsf{PVP}_{\mathrm{T}}(\mathcal{G},\mathcal{R})$. Suppose we can build, for every $k$-ensemble $\overline{R}$ of robots, a formula $\phi_{\overline{R},\mathrm{T}}$ such that for all graphs $G$ the following are equivalent:

- $G\models\phi_{\overline{R},\mathrm{T}}$
- $\overline{R}$ achieves task T on $G$.

Then, for every $\mathcal{R}$ and $\mathcal{G}$, we would have reduced the parameterised verification problem $\mathsf{PVP}_{\mathrm{T}}(\mathcal{G},\mathcal{R})$ to the $\Phi_{\overline{R},\mathrm{T}}$-validity problem for $\mathcal{G}$ where $\Phi_{\overline{R},\mathrm{T}}$ is the set of formulas $\{\phi_{\overline{R},\mathrm{T}}:\overline{R}\in\mathcal{R}\}$.[8]

We now detail this approach in the case of a single robot, i.e., $k=1$.

LEMMA 1. *Let* $R=(Q,\delta)$ *be a robot over instruction set* $\mathrm{INS}_{\Sigma,1}$*, and let* $p,q\in Q$.

*We can build* $\mathsf{MSOL}(\Sigma)$ *formulas* $\psi_{R,p,q}(X,x,y)$ *so that for every graph* $G$*:* $G\models\psi_{R,p,q}(X,x,y)$ *if and only if there exists a run of* $R$ *on* $G$ *starting from configuration* $\langle x,p\rangle$ *that has a prefix that reaches the configuration* $\langle y,q\rangle$ *and the set of positions on the prefix is exactly* $X$.

*We can build* $\mathsf{MSOL}(\Sigma)$ *formulas* $\psi^{\infty}_{R,p,q}(X,x,y)$ *so that for every graph* $G$*:* $G\models\psi^{\infty}_{R,p,q}(X,x,y)$ *if and only if there is a run of* $R$ *on* $G$ *starting from configuration* $\langle x,p\rangle$ *that reaches the configuration* $\langle y,q\rangle$ *infinitely often and the set of positions on the run is exactly* $X$.

[8]Note that in our approach $\phi_{\overline{R},T}$ does not depend on $\mathcal{R}$ or on $\mathcal{G}$.

PROOF. A robot $R=(Q,\delta)$ is a finite automaton (without initial or final states) over a finite alphabet *Alph* of instructions, i.e., $Alph\subset\{\uparrow_\sigma:\sigma\in\Sigma\}\cup\mathsf{MSOL}_1(\Sigma)$. By Kleene's theorem — which states that every finite-state automaton can be translated into a regular expression — we can build a regular expression *exp* (that depends on $R,p,q$) over alphabet *Alph* for the language of the automaton $R$ with initial state $p$ and final state $q$.

By induction on the structure of regular expressions over alphabet *Alph* we build MSOL formulas:

- $\varphi_\emptyset:=\mathsf{false}$
- $\varphi_\epsilon(X,x,y):=x=y\wedge x\in X$
- $\varphi_{\uparrow_\sigma}(X,x,y):=edg_\sigma(x,y)\wedge x,y\in X$
- $\varphi_\tau(X,x,y):=x=y\wedge\tau(x)\wedge x\in X$
- $\varphi_{r+s}:=\varphi_r\vee\varphi_s$
- $\varphi_{r\cdot s}(X,x,y):=\exists z\,[\varphi_r(X,x,z)\wedge\varphi_s(X,z,y)]$
- $\varphi_{r^*}(X,x,y):=\forall Z[(cl_{\phi_r}(X,Z)\wedge x\in Z)\rightarrow y\in Z]$ where

$$cl_\phi(X,Z):=\forall a,b\,[(a\in Z\wedge\varphi(X,a,b))\rightarrow b\in Z]\,.$$

An easy induction shows that $G\models\varphi_r(X,x,y)$ if and only if there is a sequence of instructions $ins\in Alph^*$ accepted by the regular expression $r$ and a path from $x$ to $y$ that follows instructions $ins$, and that only visits vertices in $X$ (but not necessarily all of $X$). Finally, define the MSOL formula $\psi_{R,p,q}(X,x,y)$ to state that $\phi_{exp}(X,x,y)$ holds and $X$ is minimal: $\phi_{exp}(X,x,y)\wedge\neg\exists Y(\phi_{exp}(Y,x,y)\wedge Y\subset X)$.

Similarly, by a variation of Kleene's Theorem, we can build an $\omega$-regular expression *exp* over alphabet *Alph* for the language consisting of all infinite sequences that label infinite paths in $R$ that start in $p$ and see $q$ infinitely often. Then, inductively build $\varphi_{exp}$, as before, with the following additional rule:

$$\varphi_{r^\omega}(X,x,y):=\exists z\,[\varphi_{r^*}(X,x,y)\wedge\varphi_r(X,y,z)\wedge\varphi_{r^*}(X,z,y)]$$

As before, define the MSOL formula $\psi^{\infty}_{R,p,q}(X,x,y)$ to state that $X$ is minimal such that $\varphi_{exp}(X,x,y)$ holds. □

The idea of the proof of Lemma 1 follows [7] who built a formula expressing that there is a run starting in configuration $\langle x,p\rangle$ that reaches configuration $\langle y,q\rangle$. Our Lemma extends this in two ways, i.e., recording the visited states $X$, and expressing that a configuration occurs infinitely often.

NOTE 1. *The case of multiple robots* ($k>1$) *is more subtle. For instance, the analogue of Lemma 1 does not hold. Indeed, for* $k=2$*, let* $R_1=R_2$ *be robots that have one state* $p$ *and that non-deterministically move in every direction. Then the statement "there is a run of the robots from configuration* $\langle x_1,x_2,p,p\rangle$ *to configuration* $\langle y_1,y_2,p,p\rangle$*" is equivalent to the statement that the* $k$*-ary transitive closure of the edge relation in graph* $G$ *contains the tuple* $\langle x_1,x_2,y_1,y_2\rangle$*. However, the* $k$*-ary transitive closure is not expressible in* MSOL *(as was pointed out in Example MF5 in Section 2).*

### 4.3 Robot Task Logic — RTL

We now define a logic, called RTL, for expressing robot tasks in the case of one robot ($k = 1$).

**Syntax.** Formulas of RTL are built, as in the definition of MSOL from Section 2, from the following atomic formulas: $x = y$, $Reach(X, x, y), Halt(X, x, y), Infty(X, x, y)$, and $Rept(X, x, y)$.

**Semantics.** Formulas of RTL are interpreted with respect to graphs and robots. Thus, given a graph $G$ and a robot $R$ with state set $Q$, initial-state set $I$, repeating-state set $A$, and halting-state set $H$, define the satisfaction relation $\models_{\mathsf{RTL}}$:

$$\langle G, R\rangle \models_{\mathsf{RTL}} Reach(X,x,y) \quad \text{iff } G \models \bigwedge_{p\in I}\bigvee_{q\in Q} \psi_{R,p,q}(X,x,y)$$

$$\langle G, R\rangle \models_{\mathsf{RTL}} Halt(X,x,y) \quad \text{iff } G \models \bigwedge_{p\in I}\bigvee_{q\in H} \psi_{R,p,q}(X,x,y)$$

$$\langle G, R\rangle \models_{\mathsf{RTL}} Infty(X,x,y) \quad \text{iff } G \models \bigwedge_{p\in I}\bigvee_{q\in Q} \psi^{\infty}_{R,p,q}(X,x,y)$$

$$\langle G, R\rangle \models_{\mathsf{RTL}} Rept(X,x,y) \quad \text{iff } G \models \bigwedge_{p\in I}\bigvee_{q\in A} \psi^{\infty}_{R,p,q}(X,x,y)$$

The formulas $\psi_{R,p,q}$ and $\psi^{\infty}_{R,p,q}$ are from Lemma 1. Extend the satisfaction relation to all formulas of RTL in the natural way.

**Examples.** Here are some example RTL formulas and their meanings.

1. The atomic formula $Reach(X, x, y)$ expresses that the robot, starting in position $x$, reaches position $y$, and the set of visited vertices is $X$. The RTL formula $\forall x \exists y Reach(V, x, y)$ expresses that the robot explores the graph, no matter where it starts.

2. The RTL formula $\forall x \exists y Halt(V, x, y)$ expresses "explore and stop".

3. The RTL formula $\forall x Halt(V, x, x)$ expresses "explore and return".

4. The atomic formula $Infty(X, x, y)$ expresses that the robot, starting in position $x$, visits position $y$ infinitely often, and the set of vertices the robot visits along this run is exactly $X$. Thus $\forall x Infty(V, x, x)$ is an RTL formula expressing that the robot "perpetually explores" the graph.

A task T is *captured* by an RTL formula $\vartheta$ if $\langle G, R\rangle \models_{\mathsf{RTL}} \vartheta$ is equivalent to the statement that every run of $R$ on $G$ is in $\mathrm{T}(G)$. The example formulas show that the first three tasks from Section 3.1 are captured by RTL formulas.

Here is the main theorem that solves the parameterised verification problem in the case of a single robot ($k = 1$). It reduces the PVP to MSOL-validity of the set of environments.

THEOREM 2. *Fix an edge-label set $\Sigma$. Let $\mathcal{R}$ be the set of all robots over $\mathrm{INS}_{\Sigma,1}$, let T be a task that is captured by an RTL formula, and let $\mathcal{G}$ be a set of $\Sigma$-graphs with decidable* MSOL*-validity problem. Then $\mathsf{PVP}_{\mathrm{T}}(\mathcal{G}, \mathcal{R})$ is decidable.*

PROOF. Say RTL formula $\vartheta$ captures T. Given $R \in \mathcal{R}$, build the formula $\phi_{R,\mathrm{T}}$ by replacing every atomic formula in $\vartheta$ by its definition with respect to $R$, e.g., $Reach(X, x, y)$ is replaced by $\bigwedge_{p\in I}\bigvee_{q\in Q}\psi_{R,p,q}(X, x, y)$. A routine induction on the structure of the formula $\vartheta$ gives: $\langle G, R\rangle \models_{\mathsf{RTL}} \vartheta$ if and only if $G \models \phi_{R,\mathrm{T}}$. But by Lemma 1 $\phi_{R,\mathrm{T}}$ is a formula in $\mathsf{MSOL}(\Sigma)$. Thus we can apply the fact that the MSOL-validity problem for $\mathcal{G}$ is decidable to decide whether or not $G \models \phi_{R,\mathrm{T}}$ for all $G \in \mathcal{G}$. □

COROLLARY 1. *If $\mathcal{R}$ and T are as in Theorem 2 and $\mathcal{G}$ is a context-free set of graphs, then $\mathsf{PVP}_{\mathrm{T}}(\mathcal{G}, \mathcal{R})$ is decidable.*

NOTE 2. *The case of multiple robots ($k > 1$) is harder to analyse. Indeed, Theorem 1 states that PVP is undecidable already for $k = 2$ on lines (which is a very basic context-free set of graphs) and simple reachability tasks. It is a challenging problem to find natural and useful restrictions on robots which allow the analogue of Corollary 1 (or Lemma 1) to hold in the case of multiple robots.*

## 5. ILLUSTRATION: DISTRIBUTED COMPUTING

We now instantiate our framework to a popular model of autonomous mobile-agent from the distributed computing literature, i.e., [27, 21, 16, 33, 28, 18]. To distinguish their agents from ours, we call theirs *DC-robots*. Here is their model: environments are modeled as undirected graphs, and each vertex is annotated with a local port numbering i.e., for every vertex $v$ the set of labels of the edges of $v$ are in bijection with $\{1, 2, \cdots, \mathsf{deg}(v)\}$. A DC-robot finding itself at vertex $v$ can decide to exit via local port-number $d$ and update its local state based on a) its current state, b) the identification of the robots at the same vertex $v$, c) the degree of $v$, and d) the local-port number of $v$ of the edge it used when arriving at $v$. Thus graphs are assumed to have bounded degree $\Delta$. Typical tasks from this literature are "perpetual exploration", "exploration with return", and "gathering". The main twist is that the robots should perform their task on a graph no matter the local port numbering.

Such robot systems are easily expressible in our framework. The edge-label set $\Sigma$ is defined to be $[\Delta]$, the edge-relation $E$ is assumed to be symmetric, and $\lambda(v, w) = i$ codes that $i$ is $v$'s port number for the undirected edge $\{v, w\}$. Note that $\lambda(v, w)$ need not equal $\lambda(w, v)$. The interesting aspect is how to simulate d) above. Our robot must store in its state both the state of the DC-robot, as well as the local-port number with which it entered the current vertex. It does this as follows: when our robot is at vertex $v$, and the DC-robot says to take exit $i$, our robot first determines the $w$ such that $\lambda(v, w) = i$, and then it determines the $j \in \Delta$ such that $\lambda(w, v) = j$. It can do this with test-instructions in $\mathsf{FOL}(\Sigma)$.

**Rotor-Robot** We now give a simple but important example that can be analysed in our framework. The *rotor robot* operates as follows: when it enters a vertex $v$ by port $i$ it leaves by port $i + 1$ modulo $\mathsf{deg}(v)$. This robot is known by various other names: Abelian mobile robot, rotor walk, ant walk, Eulerian walker, and Propp machine. It is important because it is a viable alternative to probabilistic robots that perform random walks [11]. It has the property that it explores all trees — a result that seems to be folklore, and that could be proved, for fixed degree $\Delta$, using Theorem 2.

**Label-Guided Tasks.** The graphs in this paper are edge-labeled, but our results hold allowing vertex-labels. Moreover, our framework can incorporate questions of the form: given a robot and a task, decide if for every graph $G \in \mathcal{G}$ there exists a labeling of $G$ such that the robot achieves the task on the graph with the help of the labeling. For instance, although there is no DC-robot that perpetually explores all graphs, there is a DC-robot and an algorithm that colours vertices by three colours, such that the robot can perpetually explore every such coloured graph [16]. Since the set of all graphs does not have decidable-validity, we might only hope to verify variations of this fact for restricted classes of graphs, e.g., fix a context-free set of graphs $\mathcal{G}$; then one can decide, given a DC-robot, whether or not for every $G \in \mathcal{G}$ the robot achieves the task "there is a colouring of $G$ using three colours that the robot can use to perpetually explore $G$". The reason this fits into Theorem 2 is that this task is captured by an RTL formula — indeed, the property that $X_1, X_2, X_3 \subseteq V$ colour a graph is easily expressed in MSOL (just say that $\wedge_{i \neq j} X_i \cap X_j = \emptyset$ and $X_1 \cup X_2 \cup X_3 = V$).

# 6. COMPLEXITY CONSIDERATIONS

As discussed in Section 4.2, the framework reduces the parameterised verification problem $\mathsf{PVP}_{\mathrm{T}}(\mathcal{G}, \mathcal{R})$ to the $\Phi_{\overline{R},\mathrm{T}}$-validity problem for $\mathcal{G}$ where $\Phi_{\overline{R},\mathrm{T}}$ is the set of formulas $\{\phi_{\overline{R},\mathrm{T}} : \overline{R} \in \mathcal{R}\}$. Unfortunately, the complexity of the decision procedure for $\mathsf{PVP}_{\mathrm{T}}(\mathcal{G}, \mathcal{R})$ in Corollary 1 may be non-elementary, i.e., not bounded by any tower of exponentials in the size of the input robot $R$. Indeed: although the size of the computed formula $\phi_{R,\mathrm{T}}$ is exponential in the size of the robot $R$ and linear in the size of the RTL formula capturing T, but the complexity of the MSOL-validity problem for $\mathcal{G}$ is non-elementary even taking $\mathcal{G}$ to be the set of binary-labeled lines [42].

Consequently, we illustrate that improved decision procedures can be found for interesting tasks and sets of graphs.

A robot is *deterministic* if for all $p \in Q$, either a) there exists $q, q' \in Q$ and $\tau \in \mathsf{MSOL}(\Sigma)$ and the only transitions out of $p$ are $(p, \tau, q)$ and $(p, \neg\tau, q')$, or b) there exists $\sigma \in \Sigma, q, q' \in Q$ and the only transitions out of $p$ are $(p, \uparrow_\sigma, q)$ and $(p, \neg\exists z(edg_\sigma(x, z)), q')$. In other words, a) says that if test $\tau$ holds goto state $q$ else goto state $q'$, and b) says that if there is an edge in the graph labeled $\sigma$ then traverse it (in case of many such edges, pick one nondeterministically) and go to state $q$, otherwise goto state $q'$. A $\Sigma$-graph is *deterministic* if $\lambda(v, w) = \lambda(v, w')$ implies $w = w'$. For instance, $\Delta$-ary trees are deterministic. Note that deterministic robots have at most one run on deterministic graphs.

THEOREM 3. *Fix $\Delta \in \mathbb{N}$. Let $\mathcal{G}$ be the $\Delta$-ary trees, let $\mathcal{R}$ be all deterministic robots, and let* T *be the "explore and halt" task. Then* $\mathsf{PVP}_{\mathrm{T}}(\mathcal{G}, \mathcal{R})$ *is* EXPTIME-COMPLETE.

PROOF SKETCH. The idea is to inter-reduce the parameterised verification problem with the universality problem for deterministic tree-walking automata (DTWA). A DTWA is a deterministic machine that can recognise sets of trees: the automaton starts at the root, at any given time the automaton sits on a node of the input tree, can test if the current node is a leaf, the root, or the $i$th child (for $i \leq \Delta$), and based on these tests the machine updates its internal state and executes one of the commands: "accept the tree", "reject the tree", "go to the parent" or "go the $i$th child". The universality of DTWA is EXPTIME-COMPLETE. Indeed, from a DTWA $A$ build a DTWA $B$ that simulates $A$ and rejects whenever $A$ runs forever (this causes a quadratic blowup in the number of states [36]); then complement $B$ by swapping accept and reject states to get a DTWA $C$; then convert $C$ into an ordinary frontier-to-root tree automaton $D$ using a subset construction that calculates loops of $C$ [9, Fact 1] (this causes an exponential blowup); then test $D$ for emptiness (which can be done in P).

Here is the reduction that gives the upper bound: given a deterministic robot $R$ build a DTWA $A_R$ that operates on trees $t$ with marked nodes $s$ and $v$, written $(t, s, v)$, as follows: first it does a DFS from the root, and when it reaches $s$ it begins the simulation of $R$; after the simulation begins, $A_R$ remembers if $v$ is visited; if $R$ enters a halt state and $v$ was visited then $A_R$ accepts $(t, s, v)$, and if $R$ enters a halt state and $v$ was not visited then $A_R$ rejects input $(t, s, v)$. Thus: $R$ "explores and halts" iff for all $(t, s, v)$ the run of $R$ on $t$ starting at $s$ visits $v$ and later enters a halt state iff $A_R$ accepts all inputs of the form $(t, s, v)$. The size of $A_R$ is linear in the size of $R$.

Here is the reduction that gives the lower bound: given a DTWA $A$, build a robot $R_A$ which first explores the input tree $t$ (doing, say, a DFS), then simulates $A$ from the root, and halts iff $A$ accepts (thus $R_A$ runs forever if $A$ rejects). Since $R_A$ always explores its input, $A$ accepts $t$ iff $R_A$ explores and halts on $t$. The size of $R_A$ is linear in the size of $A$. □

NOTE 3. *For every $\Delta \geq 2$, there is no robot that "explores and halts" on all local port-numbered $\Delta$-ary trees [21]. The proof above is easily adapted to yield an algorithm that, given a robot $R$ and $\Delta \geq 2$, returns a local-port numbered tree on which the robot does not succeed to "explore and halt".*

# 7. COMPARISON WITH RELATED WORK

Robot systems are considered distributed if they involve autonomous agents with no central control. In light of this, we first describe the relationship of our work to formal verification of distributed systems generally, and then to formal verification of robot protocols in particular.

### *Formal verification of distributed systems.*

Typical parameters that arise in the study of distributed systems are the number of agents, the number of assumed faulty agents, etc. Since parameterised problems of distributed systems are, in general, undecidable [3, 43], the formal methods community developed sound but incomplete methods that require human intervention, e.g., counter- and predicate-abstractions, inductive invariants, regular model checking and acceleration techniques. See for instance [39, references on pages 2-3].

On the other hand, by simplifying the systems one can get decidable PVP. These use techniques from games, automata theory and logic, notably finite-model properties/cutoffs, reduction to Petri nets, and the theory of well-structured transition systems [24, 26, 23, 15, 31, 19, 1, 2].

Of all these models, token-passing systems (i.e., the models in [24, 15, 1, 2]) are the closest to robots — both tokens and robots move along the vertices of graphs. However, we now argue that the model in these cited papers is incomparable with our model. First, the translation of their token-passing systems into robot systems would require that robots can read and write to variables at the vertices (this

is to model the fact that processes have states). However, we assumed environments are static (because otherwise the PVP is quickly undecidable). Conversely, translating robot-systems into the cited token-passing systems requires that the robot-systems satisfy the following unrealistic assumptions (that are used in their decidability proofs): when a robot decides to move, an adversary decides which edge it takes, as well as to which of its internal states to transition (i.e., even the robot's memory may be scrambled). Not even the simplest robots from the distributed computing literature (e.g., the rotor robot, or the DFS robot) satisfy this double restriction.

### *Formal verification of robot systems.*

The formal methods community has only recently [30, 20, 12, 6, 4, 35] begun explicitly verifying and synthesising robot protocols (rather than distributed systems in general). However, half of these papers only treat small values of the parameters. For instance, one such paper concludes [6, page 11]:

> *While our method is parameterised by both k [the number of robots] and n [the size of the graph], it does not permit to verify whether a [robot] protocol is valid for every k and n satisfying a particular predicate.*

The papers that do treat parameterised robot protocols do not give sound and complete decision procedures for their systems, as we do. Indeed, [4] uses a proof assistant to provide certificates (formal proofs) of impossibility results about robot networks; [35] uses the theory of games on graphs to synthesise a robot protocol for gathering $k$ robots on a ring of size $n$ (for small values of $k, n$), and relies on a hand-proven induction to prove that the synthesised robot protocol works for all values of the parameters $k, n$; and [32] presents a sound technique that may identify cutoffs using a counter-abstraction in order to draw conclusions about certain swarm algorithms, independently of the number of swarming agents.

The quotation above continues:

> *Adapting recent advances in parameterised model checking [citation elided] would be a nice way to obtain such results.*

Both the methodology (reducing parameterised verification of robot systems to validity problems in logic) and the results of this paper (algorithms for automatic parameterised verification of robot systems consisting of a single robot in an unknown environment) are novel and have succeeded where other methods and "recent advances" (discussed in the previous subsection) have not.

### *Comparison with graph-walking automata.*

**Graph-walking automata.** There is no canonical definition of automaton on graphs. Our model of a single robot is equivalent to graph-walking automata with MSOL-tests [7]. The proof idea of Lemma 1 (which compiles robots into formulas over graphs) is borrowed from [7, 25]. Other classical notions of automata on graphs (e.g., [8]) are often too expressive and lead to undecidable parameterised verification problems (see the proof of Theorem 1).

**Multi-head automata.** If agents are modeled as finite-state machines, then multi-agent systems are instances of multi-head automata, and the parameterised verification problem for reachability tasks is equivalent to the universality problem of such automata. A technical difference between our robot-ensembles and multi-head automata is that the $k$-many heads of a multi-head automaton are operated from a central control. In the language of robots this would mean that the robots can communicate their current local states to each other. Such communication is disallowed as it quickly results in undecidability. However, the proof of Theorem 1 shows that, similarly, even simple vision-based communication leads to undecidability.

**Tree-walking automata (TWA)** are a natural generalisation to trees of two-way automata on words. Tree-walking automata with a single head, and their corresponding regular expressions, have been studied for their own sake [9], in formal verification, e.g., [46, 10, 37], and as tree pattern-matching tools [13, 41]. TWA are used in Theorem 3 to reason about a single robot walking on unknown trees.

## 8. FUTURE CHALLENGES

This paper takes a step in the following general research agenda: find natural robot systems that have decidable or tractable PVP.

Regarding decidability, much work remains to be done. An important problem is to extend the methodology or results of this paper to multiple robots. Notes 1 and 2 discuss the challenges facing such an investigation.

Similarly, in light of the fact that PVP is quickly undecidable in a dynamic environment (e.g., this is the case for simple reachability tasks of a single robot that can read and write to every vertex on a line, cf. [43, 24]), what restrictions on the robot or the dynamic environment will result in decidable PVP?

On the other hand, we believe it is feasible to extend our framework to quantitative tasks, such as minimising the number of steps to complete a task.

Regarding complexity, we have seen that our general algorithm for solving the PVP (in Theorem 2) has high computational complexity, and we have seen (in Theorem 3) that a certain problem on trees is ExpTime-Complete. For which systems (tasks, classes of graphs, and classes of robots) is the PVP solvable in P, NP, or PSpace?

We believe that tackling these questions will open avenues both in automata theory and in the verification of mobile multi-agent systems in unknown environments.


## Acknowledgements

I thank the reviewers for their careful reading and provocative questions which helped improve the content and presentation of this paper.


## REFERENCES


[1] B. Aminof, S. Jacobs, A. Khalimov, and S. Rubin. Parameterized model checking of token-passing systems. In *VMCAI*, pages 262–281, 2014.

[2] B. Aminof, T. Kotek, F. Spegni, S. Rubin, and H. Veith. Parameterized model checking of rendezvous systems. In *CONCUR*, pages 109–124, 2014.

[3] K. Apt and D. Kozen. Limits for automatic verification of finite-state concurrent systems. *Inf. Process. Lett.*, 15:307–309, 1986.

[4] C. Auger, Z. Bouzid, P. Courtieu, S. Tixeuil, and X. Urbain. Certified impossibility results for byzantine-tolerant mobile robots. In *SSS*, pages 178–190, 2013.

[5] M. A. Bender, A. Fernandez, D. Ron, A. Sahai, and S. Vadhan. The power of a pebble: Exploring and mapping directed graphs. *Information and Computation*, 176(1):1–21, 2002.

[6] B. Berard, L. Millet, M. Potop-Butucaru, Y. Thierry-Mieg, and S. Tixeuil. Formal verification of mobile robot protocols. Report <hal-00834061>, 2013.

[7] R. Bloem and J. Engelfriet. Monadic second order logic and node relations on graphs and trees. In *Structures in Logic and Computer Science*, pages 144–161, 1997.

[8] M. Blum and C. Hewitt. Automata on a 2-dimensional tape. *SWAT (FOCS)*, pages 155–160, 1967.

[9] M. Bojańczyk. Language and automata theory and applications. *Lecture Notes in Computer Science Volume 5196*, pages 1–2, 2008.

[10] P. A. Bonatti, C. Lutz, A. Murano, and M. Y. Vardi. The complexity of enriched $\mu$-calculi. *Logical Methods in Computer Science*, 4(3:11):1–27, 2008.

[11] B. Bond and L. Levine. Abelian networks: foundations and examples. *CoRR*, abs/1309.3445, 2013.

[12] F. Bonnet, X. Defago, F. Petit, M. Potop-Butucaru, and S. Tixeuil. Brief announcement: Discovering and assessing fine-grained metrics in robot networks protocols. In *SSS*, pages 282–284. 2012.

[13] A. Brüggemann-Klein and D. Wood. Caterpillars: A context specification technique. *Markup Languages*, 2:81–106, 2000.

[14] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[15] E. Clarke, M. Talupur, T. Touili, and H. Veith. Verification by network decomposition. In *CONCUR 2004*, volume 3170, pages 276–291, 2004.

[16] R. Cohen, P. Fraigniaud, D. Ilcinkas, A. Korman, and D. Peleg. Label-guided graph exploration by a finite automaton. In *ICALP*, pages 335–346. 2005.

[17] B. Courcelle and J. Engelfriet. Book: Graph structure and monadic second-order logic. a language-theoretic approach. *Bull. EATCS*, 108:179, 2012.

[18] S. Das. Mobile agents in distributed computing: Network exploration. *Bull. EATCS*, 109:54–69, 2013.

[19] G. Delzanno. Parameterized verification and model checking for distributed broadcast protocols. In *ICGT*, pages 1–16, 2014.

[20] S. Devismes, A. Lamani, F. Petit, P. Raymond, and S. Tixeuil. Optimal grid exploration by asynchronous oblivious robots. *CoRR*, abs/1105.2461, 2011.

[21] K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc. Tree exploration with little memory. *Journal of Algorithms*, 51(1):38–63, 2004.

[22] H. Ebbinghaus and J. Flum. *Finite model theory*. Perspectives in Mathematical Logic. Springer, 1995.

[23] E. Emerson and V. Kahlon. Model checking guarded protocols. In *LICS*, pages 361–370, 2003.

[24] E. Emerson and K. Namjoshi. On reasoning about rings. *Int. J. Found. Comput. Sci.*, 14(4):527–550, 2003.

[25] J. Engelfriet and H. J. Hoogeboom. Nested pebbles and transitive closure. In *STACS*, pages 477–488, 2006.

[26] J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *LICS*, pages 352–359, 1999.

[27] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science*, 345:331 – 344, 2005.

[28] 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. 2008.

[29] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *LNCS*. Springer, 2002.

[30] X. Huang, P. Maupin, and R. Van Der Meyden. Model checking knowledge in pursuit evasion games. In *IJCAI*, pages 240–245, 2011.

[31] P. Kouvaros and A. Lomuscio. Automatic verification of parameterised multi-agent systems. In *AAMAS*, pages 861–868, 2013.

[32] P. Kouvaros and A. Lomuscio. A counter abstraction technique for the verification of robot swarms. To appear in *AAAI*, 2015.

[33] E. Kranakis, D. Krizanc, and S. Rajsbaum. Mobile agent rendezvous: A survey. *SIROCCO*, pages 1–9, 2006.

[34] 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*, pages 8–1 to 8–20. Chapman Hall/CRC Computer and Inf. Sci. Series, 2007.

[35] L. Millet, M. Potop-Butucaru, N. Sznajder, and S. Tixeuil. On the synthesis of mobile robots algorithms: The case of ring gathering. In *SSS*, pages 237–251, 2014.

[36] A. Muscholl, M. Samuelides, and L. Segoufin. Complementing deterministic tree-walking automata. *Inf. Process. Lett.*, 99(1):33–39, 2006.

[37] J. Obdržálek. Fast mu-calculus model checking when tree-width is bounded. In *CAV*, pages 80–92, 2003.

[38] A. Pelc. Disc 2011 invited lecture: Deterministic rendezvous in networks: Survey of models and results. In *DISC*, pages 1–15, 2011.

[39] A. Pnueli, J. Xu, and L. Zuck. Liveness with $(0,1,\infty)$-counter abstraction. In *CAV*, pages 93–111, 2002.

[40] A. Rosenberg. On multi-head finite automata. *IBM Journal of Research and Development*, 10(5):388–394, Sep 1966.

[41] T. Schwentick. Foundations of xml based on logic and automata: A snapshot. In *FoIKS*, pages 23–33, 2012.

[42] L. J. Stockmeyer. *The complexity of decision problems in automata theory and logic*. PhD thesis, MIT, 1974.

[43] I. Suzuki. Proving properties of a ring of finite-state machines. *Inf. Process. Lett.*, 28(4):213–214, July 1988.

[44] W. Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 133–192, 1990.

[45] W. Thomas. Languages, automata, and logic. In *Handbook of Formal Languages*, pages 389–455, 1996.

[46] M. Y. Vardi. Reasoning about the past with two-way automata. In *ICALP*, pages 628–641, 1998.