Automatic Verification of Multi-Agent Systems in Parameterised Grid-Environments

Benjamin Aminof
Technische Universität Wien, Austria
benj@forsyte.at

Aniello Murano, Sasha Rubin
Università di Napoli Federico II, Italy
aniello.murano@unina.it
sasharubin@unina.it

Florian Zuleger
Technische Universität Wien, Austria
zuleger@forsyte.at

ABSTRACT

We present a framework for modeling and analysing multiple mobile agents on grid-environments such as finite mazes and labyrinths. Agents are modeled as automata, and the grid-environments are parameterised by their size and the relative positions of the obstacles. We study the verification problem, i.e., whether given agents complete a given task on a given (possibly infinite) set of grid-environments. We identify restrictions on the agents and on the environments for which the verification problem is decidable (and in PSPACE). These assumptions are: i) there are a bounded number of obstacles, and ii) the agents are not allowed to issue commands like "increase my x-coordinate by 1" but can only issue commands that change their relative positions, e.g., "increase my x-coordinate until I go past this wall".

We prove PSPACE-hardness already for the verification problem of a single agent on singleton parameterised environments with no obstacles. It is therefore remarkable that the PSPACE-upper bound also holds for the verification problem with multiple agents, parameterised environments and multiple obstacles. We prove that weakening either of restrictions i) or ii) results in undecidability. The importance of this work is that it is the first to give a sound and complete decision procedure for the verification problem on parameterised grid-like environments. Previous work either involved only a single grid, restricted the scheduling of the agents, or excluded grids altogether.

General Terms

Theory, Verification

Keywords

Computational Models; Autonomous Mobile Agents; Distributed Robot Systems; Grids; Parameterised Verification

1. INTRODUCTION

Physical multiagent systems are designed to move in space. Thus 2D or 3D space, or their abstractions into grids [41, 7, 25, 26, 3], are the canonical environments for modeling multiagent systems. In some cases, e.g., if the environment is too dangerous or expensive to be reached by humans, we may assume that agents initially only have partial information about the environment. For instance, they may not know the exact size of the space or the extent of the obstacles, but they may know the relative positions of obstacles. Thus, agents should be designed so that they operate correctly on all possible grids that are consistent with their information [44, 43, 7, 31, 19, 25, 3]. In this paper we study the verification problem for such multiagent systems that have partial information about the environment.

Model checking is one of the main paradigms for automatic verification: the system is modeled as a finite structure M, the specification is expressed as a formula ϕ in a suitable logical formalism, and a model checking algorithm is used to verify if the structure M satisfies the formula ϕ [13]. The main obstacle to using this paradigm in our setting is that, since the environment is not completely-known, we have to model check many (potentially infinitely many) structures M. There are two main approaches to overcoming this problem, abstraction [13] and parameterised model-checking [6]. We pursue the second approach.

Motivating Scenario: Navigating Manhattan. Suppose you are arrive at Grand Central Station (GCS) in midtown Manhattan and want to get to a certain landmark, say Carnegie Hall (CH), by walking along the streets and avenues. Perhaps you already know (because you have a crude map drawn on a piece of paper) the relative positions of some of the major landmarks,[1] e.g., Times Square (TS) is west of GCS, somewhere between TS and GCS is a point, North of which is the Museum of Modern Art (MOMA), and CH is North of MOMA. Can you use this information to reach CH? Any algorithm that you use to navigate will probably have primitives of the form "walk east until you reach landmark X", or "walk east for a while, and stop at some point to look at your map again". Now, suppose you have a generic algorithm for navigating, given the relative positions of landmarks. Is your algorithm always guaranteed to succeed? This is the parameterised verification problem. It is parameterised because you do not want to test your algorithm only on this particular part of Manhattan, but on all, potentially infinitely many, such environments.

We believe that there are many scenarios of multi-agent systems where navigating using exact positioning is not possible or desirable. On the one hand, precise positioning requires complicated hardware (such as GPS receivers, servo

[1]Even if one has a map which is to scale, one probably does not navigate by measuring or calculating the distance one travels.

Appears in: *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), J. Thangarajah, K. Tuyls, C. Jonker, S. Marsella (eds.), May 9–13, 2016, Singapore.*
Copyright © 2016, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

mechanisms, etc.), and on the other hand, even if the agent knows its exact location, many times it does not have this information for other objects in the environment, e.g., if a complete to-scale map of the environment is not available, and only a partial schematic exists (such as a "map" drawn on a napkin), or if there is no map at all.

Aims and Contributions. In order to capture the motivating example above, as well as many other scenarios, the aim of this work is to provide a formal framework for reasoning mathematically and computationally about multiple mobile-agents in grid-environments, having some tasks to perform. We model agents as finite-state machines that walk on grids containing obstructions. The agents can move along the axes, e.g., in two dimensions they can move horizontally and vertically but not diagonally. Agents maintain a fixed number of beacons which they can place on the grid to remember positions they have been to, and can be used for navigation or coordination with other agents. Grid-environments are parameterised by their size and the relative positions of the obstacles on the grid. Tasks are formalised in a suitable linear-time logic that extends LTL by adding the ability to talk about the relative positions of the agents and obstacles at different points in time. In particular, it can express tasks like gathering, border patrol, and line-formation. Agents move asynchronously according to an optional scheduling policy; this is motivated by the assumption that agents have no common notion of time since they have different internal clocks [40, 2, 6].

We prove that the verification problem is decidable (and in PSPACE) under two restrictions: i) there is a bounded number of obstacles, and ii) the agents are not allowed to issue commands like "increase my x-coordinate by 1" but can only issue commands that change their relative positions, e.g., "increase my x-coordinate until I go past this wall". We prove PSPACE-hardness already for the verification problem of a single agent on singleton parameterised environments with no obstacles and no beacons. We therefore find it remarkable that the PSPACE-upper bound also holds for the verification problem with multiple agents, beacons, parameterised environments and multiple obstacles.

In stark contrast, we prove that verification is undecidable, even for very simple tasks (such as reachability), if either of these two conditions is relaxed. E.g., even on empty grids, thus i) holds, if agents can issue commands with absolute distances then verification is undecidable; and, if ii) holds, but there is no bound on the number of obstacles, then verification is undecidable even on simple "Avenues-and-Street" grids.[2]

The importance of this work is that it is the first to give a sound and complete decision procedure for the verification problem on parameterised grid-like environments. Previous work either involved abstractions [37], only a single environment [36], parameterised environments but with schedulings that restricted the number of turns [2], or parameterised environments but excluded grids altogether [45]. Thus, completely new ideas are needed.

Our work is based on the insight that the ability to move precisely in a coordinate system results in undecidability, and that methods from formal methods can shed light on weakened agents that still retain the ability of moving relative to other objects. In this way we obtain a powerful new paradigm for modelling multi agent systems.

Related work. The parameterised model-checking problem has been considered in the verification community for models of (typically immobile) processes — see [6] for a survey. On the other hand, the distributed-computing literature consideration of mobile robots (i.e., automata walking on graphs) is mostly mathematical rather than algorithmic, but naturally (and interestingly) provides theorems that are parameterised (in, e.g., the number of robots, the number of internal states of the robots, etc.) [8, 38, 28, 20, 15, 14, 27].

The main tool we exploit for decidability is a spatio temporal logic called constraint LTL that was originally defined for the verification of counter machines [17, 18].

Our paper considers qualitative tasks. In contrast, [43] studies the shortest path problem in a 2D model that bares some similarity to ours: obstacles are non-intersecting rectangles with sides parallel to the axes, and obstacles that are in a straight line of sight can be seen, and also measured.

Our beacons are reminiscent of pebble automata [21, 4] which themselves are related to extensions of first-order logic with transitive closure; we remark that in our model there is no stack-like restriction on the orders that beacons can placed or retrieved, and yet we still achieve decidability.

Our paper considers verification. A related problem is synthesis with partially known environments, also known as generalised planning, which typically considers one dimensional environments [16, 32, 34, 35].

Besides navigation in known environments, the AAMAS community has studied distributed solutions to navigation tasks in unknown environments, and in contrast to our work, most studies were experimentally verified: [33] contains a technique for computationally sophisticated robots to solve the multi-robot coverage task problem using Voronoi partitioning; In [12], computationally and memory limited agents are studied using coalition logic to manage team-formation; in [29], the agents communicate by exchanging messages with a predefined set of other agents, whose task is to reach a goal state in weighted planar graphs where robots discover adjacent nodes to visited nodes; [24] contains a variation of the A^* algorithm for finding optimal paths in partially known graphs, i.e., where robots discover nodes adjacent to visited ones; and the authors of [11], motivated by reducing deployment time, consider intelligent cameras and autonomous robots for a museum guiding service.

2. NOTATION AND BACKGROUND

Notation. Let $\mathbb{N}$ denote the integers, and $\mathbb{N}_0$ be $\mathbb{N} \cup \{0\}$. For $N \in \mathbb{N}$, write $[N]$ for the set $\{1, 2, \dots, N\}$. Fix an infinite set var of variables (that vary over $\mathbb{N}$). Fix, once and for all, a dimension $D \in \mathbb{N}$ for the grids (e.g., if $D = 1$ then the agents move on a line, if $D = 2$ then they move on planar grids, etc). Write ϵ_i for the D-tuple $(0, \dots, 0, 1, 0, \dots, 0)$ that is 0 everywhere except for the ith co-ordinate which is 1. Write $\overline{0}$ for the D-tuple $(0, \dots, 0)$. Write $\overline{x}[i \leftarrow a]$ for the tuple $\overline{x}$ with the value of the ith co-ordinate replaced by the a, e.g., $(0, 0, 1)[2 \leftarrow 1]$ is $(0, 1, 1)$. Throughout, we freely interchange between functional notation $f(i) = b$ and vector notation $(\dots, b, \dots)$. For an infinite sequence σ, write σ_i for the ith element of σ.

[2]Moreover, we get undecidability even if the absolute placement of the obstacles is not fixed, e.g., the width of the avenues may vary.

Two-counter Machines. Our undecidability proofs proceed by reducing the halting problem of two counter machines to the verification problem. An *input-free 2-counter machine* (2CM) [42] 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*. The *non-halting problem for 2CMs*, which is known to be undecidable [42], is to decide given a 2CM M whether it does not halt.

Constraint linear-temporal logic ($\mathsf{C\text{-}LTL}$)**.** Linear temporal logic (LTL) is nowadays a well-established logic for reasoning about reactive systems. We recall the definition of constraint linear-temporal logic ($\mathsf{C\text{-}LTL}$) introduced in [18]. $\mathsf{C\text{-}LTL}$ is a spatio-temporal logic that has been introduced for the verification of constraint automata, a class of automata that can be viewed as abstractions of counter-automata. It is the main tool that allows us to achieve decidability of the verification problem for multi-agent systems on grids.

A *term* t is either an element of $\mathbb{N}_0$ or an expression of the form $\mathsf{O}^i(x)$ ($x \in \mathsf{var}$ and $i \in \mathbb{N}_0$). We identify $\mathsf{O}^0(x)$ with x. An *atomic constraint* R is an expression of the form $t \bowtie t'$ where t, t' are terms and $\bowtie$ is one of $=, <, \equiv_a^b$ ($0 \leq a < b \in \mathbb{N}_0$). Here $=, <$ are interpreted as usual on $\mathbb{N}_0$, and $\equiv_a^b$ is the modulo unary relation, i.e., $\equiv_a^b (x)$ iff $x = a \mod b$. E.g., $\mathsf{O}^2(x) < \mathsf{O}^1(y)$ is an atomic constraint. Informally, it means that the value of x two time steps ahead is less than the value of y one time step ahead. A *constraint* is a Boolean combination of atomic constraints. A *number constraint* is a constraint that only involves elements of $\mathbb{N}_0$ and variables, i.e., no terms of the form $\mathsf{O}^i(x)$ for $i > 0$.

The syntax of $\mathsf{C\text{-}LTL}$ is given by: $\varphi ::= R \,|\, \neg\varphi \,|\, (\varphi \vee \varphi) \,|\, \mathsf{O}\,\varphi \,|\, \varphi \,\mathsf{U}\, \varphi$ where R is an atomic constraint. A *valuation* is a function $\mathsf{var} \to \mathbb{N}_0$. Models of $\mathsf{C\text{-}LTL}$ formulas are infinite sequences of valuations, i.e., $\sigma = \sigma_0\sigma_1 \ldots$ where each σ_j is a valuation. The semantics of $\mathsf{C\text{-}LTL}$ follows:

$$\begin{array}{ll}
(\sigma, i) \models R & \text{iff } \sigma_{i+l}(x) \bowtie \sigma_{i+m}(y) \text{ and} \\
 & \quad R = \mathsf{O}^l(x) \bowtie \mathsf{O}^m(y) \\
(\sigma, i) \models \neg\varphi & \text{iff } (\sigma, i) \not\models \varphi \\
(\sigma, i) \models \varphi \vee \psi & \text{iff } (\sigma, i) \models \varphi \text{ or } (\sigma, i) \models \psi \\
(\sigma, i) \models \mathsf{O}\,\varphi & \text{iff } (\sigma, i+1) \models \varphi \\
(\sigma, i) \models \varphi \,\mathsf{U}\, \psi & \text{iff } \exists j \geq i : (\sigma, j) \models \psi \text{ and} \\
 & \quad \forall l \in [i, j) : (\sigma, l) \models \varphi
\end{array}$$

Write $\sigma \models \varphi$ if $(\sigma, 0) \models \varphi$. We use the usual derived operators $\mathsf{F}\,\varphi := \mathsf{true}\,\mathsf{U}\,\varphi$, its dual $\mathsf{G}\,\varphi := \neg\,\mathsf{F}\,\neg\varphi$, and $\varphi_1 \,\mathsf{R}\, \varphi_2$ which is the dual of U, i.e., $\neg(\neg\varphi_1 \,\mathsf{U}\, \neg\varphi_2)$, as well as the short-hands $\bigwedge$ and $\bigvee$. For example, $\mathsf{G}(x < \mathsf{O}^1(x))$ means that "globally the value of x at the current point of time is smaller than the value of x at the next point of time", which simply means that x is always strictly increasing.

We mention the following important facts about $\mathsf{C\text{-}LTL}$, see [18]. First, recall that linear-temporal logic (LTL) has the same syntax as $\mathsf{C\text{-}LTL}$ except that instead of atomic constraints it has atomic propositions such as p and q; and LTL formulas are interpreted over infinite sequences of sets of atomic propositions. Now, $\mathsf{C\text{-}LTL}$ generalizes LTL (the idea is to associate to each atom p a unique variable x, and replace p by the constraint $x = 1$ and replace $\neg p$ by the constraint $x = 0$). Second, the satisfiability problem (and thus also the validity problem) for $\mathsf{C\text{-}LTL}$ (i.e., given a $\mathsf{C\text{-}LTL}$ formula φ, decide if there exists σ such that $\sigma \models \varphi$) is decidable, and in fact has the same complexity as for LTL, i.e., it is PSPACE-complete.

3. ENVIRONMENTS AND AGENTS

Environments. We model environments as grids of arbitrary, but fixed, dimension containing obstacles whose faces are parallel to the axes. A (finite or infinite) set of environments of interest is called a *parameterised environment*. Comparisons between points of $\mathbb{N}^D$ are taken point-wise, e.g., $\overline{v} \leq \overline{w}$ means that $v_i \leq w_i$ for $i \leq d$. We describe various subsets of $\mathbb{N}^D$. A *rectangle* is a subset of $\mathbb{N}^D$ of the form $\{\overline{a} \in \mathbb{N}^D : \overline{v} \leq \overline{a} \leq \overline{w}\}$, where $\overline{v} \leq \overline{w} \in \mathbb{N}^D$. The points $\overline{v}$ and $\overline{w}$ are called the *SW- and NE-corners* of the rectangle, respectively.[3] The *interior* of a rectangle R, whose SW and NE corners are $\overline{v}$ and $\overline{w}$, is the set $R^{int} := \{\overline{a} \in \mathbb{N}^D : \overline{v} < \overline{a} < \overline{w}\}$.

A K*-obstruction* $M = \langle M_1, \ldots, M_k \rangle$ is a tuple of K many rectangles. Given $L \in \mathbb{N}$, and a K-obstruction M such that all rectangles in M are contained in $[L]^D$, the (L, M)*-environment* is the graph $G = (V, E)$ whose vertex set $V = [L]^D \backslash (\bigcup_i M^{int})$ (i.e., it is the rectangle with corners $(0, \ldots, 0)$ and $(L, \ldots, L)$, and the interiors of the rectangles in M removed), and whose edge relation $E \subset V \times V$ consists of pairs $(\overline{v}, \overline{w})$ such that for some i we have $|v_i - w_i| = 1$ and for $j \neq i$ we have $v_i = w_i$ (i.e., horizontally- or vertically-adjacent vertices are joined with an edge). Environments are also called finite labyrinths [30]. A *parameterised environment* is some finite or infinite set of environments.

Since only the interiors of the obstructing rectangles are removed, agents are allowed to move along the "walls" of an obstacle. Observe that this has the effect that if, for example, two rectangles (or a rectangle and a boundary of the grid) have a common edge/face, agents can still walk between them "along the adjoining wall". If one wants to ensure, that agents cannot walk between two rectangles, one can specify that the rectangles overlap. We add that overlapping rectangles can be used to build obstacles of different shapes than rectangles. For example, one can build a non-rectangular obstacle such as an "L" by using two rectangles and specifying that they have the same SW-corner, and that one rectangle is thinner and higher and the other rectangle is broader and more shallow. We finally add, that if one wants to remove the points common to a rectangle and a boundary of the grid[4] this could be accomplished by adding a second type of rectangle with the desired behaviour. For ease of exposition we refrain from doing so.

Agents. Agents are modeled as finite-state machines that can move along the axes of the grid. E.g., in a 2-dimensional grid, each single move is in direction north, south, east or west, but not diagonally. More formally, an agent A is tuple (Q, I, δ) where Q is a finite set of *states* (we usually assume

[3]Note that even though our grids are not necessarily 2-dimensional this is perhaps the most common case, and we find it assists in imagining things if we use 2-dimensional terms like "NE".

[4]This can be used also to simulate a grid with sides of different length by positioning suitable obstacles along the boundaries.

w.l.o.g. that $Q = [|Q|]$); $I \subseteq Q$ is a set of *initial states*; $\delta \subset Q \times \text{GC} \times Q$ is a finite *transition relation*. Elements of GC are called *guarded commands* and are of the form $(d, guard)$ where $d \in [D]$ is the axis along which the agent should move, and *guard* is a condition specifying some restriction that the path taken by the agent during this move must satisfy for the move to be possible. For example, the guard may specify that the agent moves exactly one step in the positive direction of the axis. There are many possible definitions for the types of guards that can be used, and these choices determine the "power" the agents have, and we will consider a few possible definitions of GC in later sections. In all cases, we assume that the guard can test whether the agent is at a boundary of the grid. Observe that we implicitly assume[5] that i) the path taken by an agent in a single command does not collide with any other agent (in more detail, an agent may start and end in the same position as another agent, but it may not "pass-through" an agent in a single move); and ii) the path taken does not use positions occupied by an obstruction. Note that the guard may be such that it does not fully specify the end position of a move, in which case the system nondeterministically chooses it (in a way which satisfies the guard).

4. UNDECIDABILITY RESULTS

In this section we give two simple undecidability results that direct our choice of model for agents in parameterized grid environments, presented in the next section.

A common assumption is that agents have the ability to exactly know and/or control their position [41, 27, 38, 39, 36]. This assumption abstracts real-world situations of agents that, for example, are equipped with a global positioning system (GPS). Unfortunately, as demonstrated by the following theorem, it is very easy to show that making this assumption leads to undecidability of the verification problem with respect to even the simplest tasks and parameterized grid environments. It is worth noting that this assumption can be made, without leading to undecidability, in many cases where the environments of interest are not grids, e.g., environments with bounded tree-width [45, 2].

Say that an agent has *precise positioning* if GC contains (directly or indirectly) commands of the form $(d, x_d = x'_d + j)$, with $j \in \{0, -1, 1\}$, where x_d, x'_d are the positions of the agent along the d axis at the start and end of the move, and guards that allow the agent to test its position, in particular whether it is on the bottom or left boundary of the grid.

THEOREM 1. *The following problem is undecidable: given a single agent with precise positioning, and a state h of the agent, verify that for all obstacle-free finite 2-dimensional grids no run of the agent ever enters state h.*

PROOF. The proof is by reduction from the non-halting problem of 2-counter machines. Given a 2CM M, we build an agent A that has the same states as the 2CM plus an additional `overflow` state. The agent simulates the 2CM by using its position (x, y) to encode a configuration of M where the first counter has value x and the second counter has value y. The agent begins by going to the origin of the grid (south as far as possible then west as far as possible). Incrementing (resp. decrementing) a counter is done by simply moving one step in the correct direction. A test for zero amounts to the agent checking that it has collided with the bottom or left boundary of the grid. If the agent wants to go right or up (to simulate an increment), but hits the boundary, it enters the special `overflow` state and discontinues the simulation.

It is easy to see that, on an $L \times L$ grid, the agent can simulate any prefix of the computation of the 2CM in which the values of both counters never go above n. Thus, M does not halt iff, for all $L \in \mathbb{N}$ the agent A doesn't enter the halting state in the $L \times L$ grid. $\square$

Observe that it is simple to modify Theorem 1 and obtain that, if one wishes to consider grids of unbounded size, verifying any non-trivial task for an agent with precise positioning is undecidable. Hence, if we wish to be able to automatically verify that the agent performs its task on a families of grids of unbounded size we must limit the agent's ability to precisely control/know its location.

In our quest for a model for describing agents in parameterized grid environments we are motivated by the example from the introduction of a person navigating in an unknown city using a schematic map, or instructions from a GPS unit. Note that one is usually *not* using the precise positioning information available to the GPS, but one is simply following instructions such as "turn left at the next corner". All that is needed to follow such an instruction is the ability to detect the next corner. Let us consider then a model for the agent where the guarded command can specify a guard that expresses (directly or indirectly) that the end position $\overline{x}'$ of the move is a vertex of some obstacle in the environment, and no point strictly between the start point and the end point is a vertex of an obstacle.[6] This is arguably a very basic way of detecting a corner, as one is doing it "by feel" (reminiscent of how a blind person uses a white cane). We call agents with this ability agents with *next corner detection*.

For the following theorem, given $L \in \mathbb{N}$, let the L-regular-city be the 2-dimensional grid environment of length $2L+1$, in which for every $i, j \in \{0, \ldots, L\}$ there is a 1×1 obstacle whose SW corner is located at $(2i, 2j)$.

THEOREM 2. *The following problem is undecidable: given a single agent with next corner detection, and a state h of the agent, verify that for all $L \in \mathbb{N}$, no run of the agent ever enters state h while navigating the L-regular-city.*

PROOF. The proof is very similar to that of Theorem 1. Unlike Theorem 1, we do not assume that the agent has precise positioning, and thus it can not encode the counters of the 2CM directly by its position. However, it can do so indirectly as follows: it encodes the value of the first counter by the number of obstacles that are strictly to the west of it (at the same y coordinate as it is), and the value of the second counter by the number of obstacles that are strictly to the south of it (at the same x coordinate). The main difference with the agent in the proof of Theorem 1, is that incrementing (resp. decrementing) a counter is done by moving to the SW corner of the next obstacle in the correct direction, which takes two moves. For example, if q is an increment of the first counter, the agent moves twice to the next corner to the east: the first move takes it from the SW corner of the current obstacle to its SE corner (and to state

[5]In other words, an instruction can be taken only if these extra conditions are satisfied.

[6]If one wishes, one can further assume that if no such $\overline{x}'$ exists then the agent walks as far as it can.

q'), and the next move takes it to the SW corner of the next obstacle (and the next state of the 2CM).

It follows that, in an L-regular-city, the agent can simulate any prefix of the computation of M in which the counter values don't exceed L. Thus, M does not halt iff, for all $L \in \mathbb{N}$, the agent never enters the halting state of M. $\square$

As was the case with Theorem 1, one can modify the proof above to obtain undecidability for any non-trivial task. The proof goes through unchanged even if the definition of an L-regular-city is modified to allow the obstacles and the gaps between them to have different sizes, as long as the general structure of aligned streets and avenues is maintained.

5. A MODEL OF MAS ON GRIDS

In this section we present a model of multiple agents operating asynchronously in unbounded parameterized grid environments with a bounded number of obstacles. We also define a way to specify the tasks for these agents.

The theorems in the previous section imply that if we want to be able to decide whether our agents achieve their specified tasks on grid environments of unbounded size, and still maintain the agents' ability to detect corners of obstacles, we should disallow the agents from precisely specifying the distance they travel as well as bound the number of obstacles in a parameterized environment. However, we place no bounds on the sizes and positions of the obstacles.

Informally, the guarded commands of the agents do not allow them to specify absolute positions or step sizes, and only allow one to express relative positioning with respect to other objects in the environment. In order to regain some of the power lost by this relative positioning, agents are also allowed to place some *fixed* number of smart markers, called *beacons* (similar to non-directional radio beacons used in air and marine navigation), in the environment. At any point in time, and from any point in the environment, an agent can test whether its position along any of the D axes is smaller, equal, or larger than that of any other agent, corner of an obstacle, or any beacon (of any agent). A beacon can also be remotely retrieved by the agent that owns it (i.e., without going back to it) and deposited at the current location of the agent. Agents are also able to query other agents as to their current local state.

Note that the beacons allow the agents to perform certain operations that would otherwise be impossible with relative positioning, e.g., such as marking a position an agent can later return precisely to, or to draw a virtual line in the town square such that the agents later gather in the square with half of them on each side of the line. The beacons can be implemented, for example, by making the following (limited) use of a GPS system and memory registers: placing a beacon number i of agent j in the current position is *simulated* by j recording, in its i'th register, the current position as indicated by the GPS. Comparing an agent's position to that of the beacon is done by querying the GPS system as to whether some coordinate d of the current location is smaller, larger, or equal, to the value in the corresponding register. Retrieving the beacon is done by simply overwriting the value in the register. Obviously, not all the power of this model may be needed or possible to implement by a real system. For example, our agents can see through walls (I.e., they are in a "smart city" where the corners of each obstacle broadcast radio signals), and one is welcome to not make use of any unnecessary feature (E.g., limit the agents to not see through obstacles).

For the rest of the paper fix the number B of beacons, the number N of agents, and the number K of obstacles.

Variables. We define the following variables (i.e., elements of var), that will be used later to define other important concepts such as guards, runs, etc.:

- *agent variables* $\mathsf{avar} := \{x_{n,d} : n \in [N], d \in [D]\}$;
- *primed agent variables* $\mathsf{avar}' := \{x'_{n,d} : n \in [N], d \in [D]\}$;
- *beacon variables* $\mathsf{bvar} := \{b^j_{n,d} : n \in [N], d \in [D], j \in [B]\}$;
- *primed beacon variables* $\mathsf{bvar}' := \{b'^j_{n,d} : n \in [N], d \in [D], j \in [B]\}$;
- *agent local-state variables* $\mathsf{svar} := \{s_n : n \in [N]\}$;
- *obstruction variables* $\mathsf{ovar} := \{u_{k,d}, v_{k,d} : k \in [K], d \in [D]\}$;
- the *size variable* l, and the *turn variable* $turn$.

The values of each variable have the following meanings. $\overline{x}_n := (x_{n,1}, \ldots, x_{n,d})$, is the current position of agent n, and the primed version $\overline{x}'_n$ is the next position of agent n; the position of the j'th beacon of agent n is $\overline{b}^j_n := (b^j_{n,1}, \ldots, b^j_{n,d})$, and its next position is $\overline{b}'^j_n := (b'^j_{n,1}, \ldots, b'^j_{n,d})$. The value of s_n is the current local state of the nth agent. The variables $\overline{u}_k$ and $\overline{v}_k$ are the SW- and NE-corners of the kth obstacle; l is the length of the grid; and $turn$ is which agent's turn it is (used to define schedules). For simplicity, we assume that the values of l and variables in ovar do not change over time, i.e., that the size of the grid and the positions of obstacles are fixed throughout a run of a system.

Parameterised environments with a fixed number of obstacles. An *environment constraint* is a number constraint ϕ over variables $\mathsf{ovar} \cup \{l\}$. It determines a set of environments $\mathcal{G}_\phi$ as follows. Every (L, M)-environment G with $M = \langle M_1, \ldots, M_K \rangle$ determines a valuation σ_G over $\mathsf{ovar} \cup \{l\}$ as follows: σ_G maps variable l to L, and for each $i \in [K]$, σ_G maps $\overline{u}_i$ (resp. $\overline{v}_i$) to the SW-corners (resp. NE-corner) of the rectangle M_i. Thus, a number constraint ϕ determines the set $\mathcal{G}_\phi$ of environments G such that the valuation σ_G satisfies ϕ. E.g., for $d = 2, K = 1$, and 2-tuples of variables $(u_1, u_2), (v_1, v_2)$ representing the inner rectangle, the constraint $l \geq 10 \wedge \bigwedge_{i=1,2} 0 < u_i < v_i < l$ determines the set of rectangular "race-tracks" of size at least 10.

Note that by constraining the variable l, we may specify environments of a single size (e.g., $l = 4$), a finite set of sizes (e.g., $l = 4 \vee l = 5$ or $l < 10$), or infinitely many sizes (e.g., $l > 10$ or $\equiv^2_0 (l)$).

Agents. Recall from Section 3 that an agent A is tuple (Q, I, δ) where Q is a finite set of states, $I \subseteq Q$ is a set of initial states, and $\delta \subset Q \times \textsc{gc} \times Q$ is a finite transition relation, where the guarded commands $\textsc{gc}$ are of the form $(d, guard)$ where $d \in [D]$. It remains to define the possible values of $guard$. We let $guard$ be any number constraint over $\mathsf{avar} \cup \mathsf{avar}' \cup \mathsf{bvar} \cup \mathsf{bvar}' \cup \mathsf{ovar} \cup \{l\}$. Note that a transition specifies that beacon j of agent n is retrieved and dropped in the new location of the agent by specifying in the guard that $\overline{x}'_n = \overline{b}'^j_n$. An *agent ensemble* is a tuple of agents $\mathcal{A} = \langle A_1, \cdots, A_N \rangle$ (such that all the agents' commands use the same N, K and B).

Configurations, Schedules, Runs. Let $G = (V, E)$ be an (L, M)-environment with K obstacles, and let $\mathcal{A}$ be an agent ensemble. A *configuration* of $\mathcal{A}$ on G is a tuple $c := (loc, locb, state)$, where $loc : [N] \to V$ maps an agent to its location; $locb : [N] \times [B] \to V$ maps a beacon to its location; and $state : [N] \to \cup_j Q_j$, such that $state(j) \in Q_j$ (for $j \in [N]$), maps an agent to its current local state. Say that c is an *initial* configuration if $state(j) \in I_j$ (for $j \in [N]$).

We can think of c as a valuation val_c over variables $\mathsf{avar} \cup \mathsf{bvar} \cup \mathsf{ovar} \cup \mathsf{svar}$ that (for $n \in [N], j \in [B], k \in [K]$): maps $\overline{x}_n$ to $loc(n)$, maps $\overline{b}^j_n$ to $locb(n, j)$, maps $\overline{u}_k$ (resp. $\overline{v}_k$) to the SW-corner (resp. NE-corner) of the kth obstruction M_k of the environment G; and maps s_n to $state(n)$. Similarly, if we are given a second configuration c', we can further extend val_c to get a valuation $val_{c,c'}$ that maps $\overline{x}'_n$ to $loc'(n)$ and $\overline{b}'^j_n$ to $locb'(n, j)$.

For configurations $c = (loc, locb, state)$ and $c' = (loc', locb', state')$, and $n \in [N]$, write $c \vdash_n c'$ if agent n can, by taking one transition, change the configuration from c to c'. Formally, $c \vdash_n c'$ if there exists $(q, (d, guard), q') \in \delta_n$ and $\alpha \in \mathbb{Z}$ such that:

1. $state(n) = q$, and $state' = state[n \leftarrow q']$ (i.e., the agent changes state from q to q');
2. $loc' = loc[n \leftarrow loc(n) + \alpha\epsilon_d]$ (i.e., the agent moves some distance $\alpha \in \mathbb{Z}$ along the dth axis);
3. For every $m \in [N], j \in B$, either $locb'(m, j) = locb(m, j)$, or $m = n$ and $locb'(m, j) = loc'(n)$; (i.e., the agent can transfer any of its beacons to its new location);
4. $\sigma_{loc,loc'} \models guard$ (i.e., the guard holds);
5. for every $\overline{v} \in \mathbb{N}^D_0$ strictly between $loc(n)$ and $loc'(n)$: $\overline{v} \in V$ (i.e., no obstruction) and $\overline{v} \neq loc(j)$ for all $j \in [N] \setminus \{n\}$ (i.e., no collision).

An *schedule* κ is an element of $[N]^\omega$. A set S of schedulers is called a *scheduler*. A *scheduler constraint* is a $\mathsf{C\text{-}LTL}$ formula ω over the variable $\{turn\}$. It induces the set of schedules κ such that there exists $\sigma \models \omega$ such that $\sigma_i \models (turn = \kappa_i)$ for all $i \in \mathbb{N}$.[7]

EXAMPLE 1. *Round-robin of N agents may be expressed as the N-scheduler consisting of the schedules $(\pi(1) \ldots \pi(N))^\omega$ such that π is a bijection of $[N]$. To express this scheduler in* $\mathsf{C\text{-}LTL}$ *one may use the formula $\bigvee_\pi \mathsf{G} \bigwedge_n (turn = n) \to \mathsf{O}(turn = \pi(\pi^{-1}(n) + 1))$ where the disjunction is over all bijections π of $[N]$.*

EXAMPLE 2. *Fair scheduling of N agents may be expressed as the set of schedules satisfying $\bigwedge_n \mathsf{G}\,\mathsf{F}(turn = n)$.*

A *run ρ of $\mathcal{A}$ on G according to scheduler S* is an infinite sequence $\rho = \rho_1\rho_2\rho_3 \cdots$ of configurations such that ρ_1 is initial and there exists $\kappa \in S$ such that $\rho_i \vdash_{\kappa_i} \rho_{i+1}$, for all i. The *agent locations of the run* is the sequence $loc(\rho_1)loc(\rho_2)\ldots$, the *beacon locations of the run* is the sequence $locb(\rho_1)locb(\rho_2)\ldots$, and the *states of the run* is the sequence $state(\rho_1)state(\rho_2)\ldots$. To every run $\rho = \rho_1\rho_2\ldots$ we associate $val(\rho)$, which is the sequence of valuations $val_{\rho_1}val_{\rho_2}\ldots$.

[7]It is also possible to define more powerful schedulers by $\mathsf{C\text{-}LTL}$ formulae over the variables $\{turn\} \cup \mathsf{svar} \cup \mathsf{avar} \cup \mathsf{bvar} \cup \mathsf{ovar} \cup \{l\}$. For ease of exposition we refrain from doing so.

Agent Tasks. Agents should achieve some task in their environment. A *task* is a $\mathsf{C\text{-}LTL}$ formula τ over variables $\mathsf{avar} \cup \mathsf{svar} \cup \mathsf{bvar} \cup \mathsf{ovar} \cup \{l\}$. The ensemble $\mathcal{A}$ *achieve the task τ in environment G according to scheduler S* if for all runs ρ of $\mathcal{A}$ on G according to scheduler S, the sequence of valuations $val(\rho)$, satisfies τ. Note that the task may, if one wishes, restrict the initial states and positions of the agents. For example, agents may be required to start at the corners of the grid, etc.

EXAMPLE 3. *Agents* gather *if eventually they arrive at the same, not previously determined, location [39]. This task can be expressed by the* $\mathsf{C\text{-}LTL}$ *formula $\mathsf{F} \bigwedge \{x_{n,d} = x_{m,d} : n, m \in [N], d \in [D]\}$.*

EXAMPLE 4. *The* Line Formation *task requires the set of agents to form a line, an example of a pattern-formation task [25]. This can be expressed by the* $\mathsf{C\text{-}LTL}$ *formula: $\mathsf{F}\, Line$ where $Line := \bigvee_{d \in [D]} \bigwedge \{x_{n,d} = x_{m,d} : n, m \in [N]\}$. A variation asks that the agents repeatedly form a line, and can be written $\mathsf{G}\,\mathsf{F}\, Lin$.*

The following is inspired by the task of a guard making sure that the doors of all buildings on campus are locked.

Example.

We consider the task of **border patrol** for a **single agent**. This task consists of moving through the grid such that every border, in our encoding *every edge* of *every obstacle*, is traversed at least once by the agent. Below we will give a specification for border patrol and describe a protocol for an agent that achieves border patrol. Our main positive result (Theorem 3) means that one can automatically verify this protocol against this specification for every parameterized grid environment satisfying the constraints of Theorem 3. For ease of exposition, we consider the 2-D setting where all obstacles are disjoint (disjointness can be encoded by a $\mathsf{C\text{-}LTL}$-formula which is quadratic in the number of obstacles).

Specification for border patrol. We consider an obstacle described by SW corner (x_1, y_1) and NE (x_2, y_2). For the edge between (x_1, y_1) and (x_1, y_2) (the bottom edge of the rectangle) we can encode by an *eventually formula* the agent arrives at the corner (x_1, y_1) or (x_1, y_2). Then we can describe by an *until formula* that the agent will eventually reach the other corner of the edge while not leaving the edge on the way. Border control is specified by a conjunction of formulae as described above for every edge of every obstacle.

Protocol for border control. We assume that the agent starts at the origin (where the security personnel office is located), i.e., the SW corner of the grid. When the agent is at the left side of the grid it places a beacon on its position. Then the agent moves from the left of the grid to the right of the grid circling around every obstacle encountered on its way. After circling around an obstacle the agent uses the beacon to stay on the horizontal line defined by the coordinates of the beacon. When the agent has arrived at the right hand side of the grid it returns to the beacon. Then the agent moves upwards until it on the same height with the next obstacle, i.e, the agent and the SW corner of the first obstacle in the whole environment whose SW corner is at a larger y coordinate then its current position. The agent repeats this behavior until it reached the NE corner of the grid. This completes the description of the agent. It is interesting to note that the described agent can be implemented by a finite automaton with a single beacon whose size only depends linearly on the number of obstacles.

Observe that in the description above, the agent's move up until aligned with the SW corner of the next obstacle cannot be directly implemented in a model where the agent can only use vision to navigate (since that corner may be obstructed by other obstacles that are horizontally between the agent and that corner), and the example illustrates the power of our model. As noted before, one can obviously limit the agents to not use the full power of the model, and one can come up with border patrol protocols for vision/touch limited agents as well.

6. PARAMETERISED VERIFICATION

We first formalise the parameterised verification problem (PVP) for agents on parameterized grids with a bounded number of obstacles (as defined in the previous section) and show that it is decidable. The PVP depends on formulas for tasks, environments, and schedulers. Recall that $\mathcal{G}_\phi$ is the (possibly infinite) set of environments determined by ϕ, and S_ω is the set of schedules determined by ω.

Definition 1. *The **parameterised verification problem of Sliding Robots on Grids**, written* PVP, *is the following decision problem: given an agent ensemble $\mathcal{A}$, a task τ for the agents, an environment constraint ϕ describing a parameterized grid environment, and scheduler constraint ω, decide whether for every $G \in \mathcal{G}_\phi$, the agents $\mathcal{A}$ achieve task τ on environment G according to scheduler S_ω.*

Theorem 3. *The* PVP *is in* PSPACE.

Proof. We show this by reducing the PVP to satisfiability of C-LTL which is in PSPACE [18]. I.e., we effectively transform the agent-ensemble $\mathcal{A} = \langle A_1, \ldots, A_N \rangle$, environment constraint ϕ, and scheduler constraint ω, into a C-LTL formula $\psi_{\mathcal{A},\phi,\omega}$ whose models code all, and only, the runs of $\mathcal{A}$ on environments $G \in \mathcal{G}_\phi$ according to the scheduler S_ω. Then we check that the C-LTL formula $\psi_{\mathcal{A},\phi,\omega} \to \tau$ is valid, or equivalently, that $\neg(\psi_{\mathcal{A},\phi,\omega} \to \tau)$ is not satisfiable.

Here are the details of the transformation. Say $A_n = (Q_n, I_n, \delta_n)$, and w.l.o.g. assume, for $n \in [N], j \in Q_n$, that $Q_n = [|Q_n|]$. The formula $\psi_{\mathcal{A},\phi}$ has variables $\mathsf{avar} \cup \mathsf{ovar} \cup \{l\} \cup \{s_i : i \in [N]\} \cup \{turn\}$ and uses constants from the set $\{0, 1, \cdots, \max_n |Q_n|\}$. We first define some helper formulas:

- $In(\overline{x}, \overline{u}, \overline{v})$ is $\bigwedge_d u_d \le x_d \le v_d$ (position $\overline{x}$ is within the rectangle determined by $\overline{u}$ and $\overline{v}$);
- $NotObstructed$ is $\bigwedge_n \bigwedge_k \neg In(\overline{x}_n, \overline{u}_k, \overline{v}_k)$ (no agent is inside any rectangle obstruction);
- $Btwn_d(\overline{x}, \overline{y}, \overline{z})$ is $(x_d < y_d < z_d \vee z_d < y_d < x_d) \wedge \bigwedge_{e \neq d} z_e = y_e = x_e$ (position $\overline{y}$ is between positions $\overline{x}$ and $\overline{z}$ and lie parallel to axis d);
- $\mathsf{O}^1(\overline{z})$ is $(\mathsf{O}^1(z_1), \ldots, \mathsf{O}^1(z_n))$.

Define $\psi_{\mathcal{A},\phi,\omega}$ as the conjunction of:

- $\mathsf{G}\, NotObstructed$ (agents are unobstructed);
- $\phi \wedge \mathsf{G}[l = \mathsf{O}^1(l) \wedge \bigwedge_k \overline{u}_k = \mathsf{O}^1(\overline{u}_k) \wedge \overline{v}_k = \mathsf{O}^1(\overline{v}_k)]$ (obstructions satisfy the environment constraint and do not change over time);
- $\bigwedge_n \bigvee_{j \in I_n} s_n = j$ (each agent starts in an initial state);
- $\omega \wedge \mathsf{G} \bigwedge_n (turn \neq n) \to (\overline{x}_n = \mathsf{O}^1(\overline{x}_n) \wedge s_n = \mathsf{O}^1(s_n))$ (agents take turns according to a schedule in ω);
- $\mathsf{G} \bigwedge_n \bigwedge_{i \in B} (turn \neq n) \to (\overline{b}^i_n = \mathsf{O}^1(\overline{b}^i_n))$ (only beacons of the agent whose turn it is can be moved);
- $\bigwedge_{n,j} \mathsf{G}[turn = n \wedge s_n = j \to \bigvee_{t \in \delta_n} Next_t]$ (agents follow their protocols),

where $Next_t$, for t of the form $(j, (d, guard), j')$, is

$$guard' \wedge move \wedge \mathsf{O}^1(s_n) = j' \wedge nocollide \wedge beacon$$

where $guard'$ is $guard$ with every primed agent variable $x'_{m,e}$ (resp. beacon variable $b^i_{m,e}$) is replaced by $\mathsf{O}^1(x_{m,e})$ (resp. $\mathsf{O}^1(b^i_{m,e})$); $move$ is $\bigwedge_{e \neq d} x_{n,e} = \mathsf{O}^1(x_{n,e})$; $nocollide$ is $\bigwedge_{m \neq n} \neg Btwn_d(\overline{x}_n, \overline{x}_m, \mathsf{O}^1(\overline{x}_n))$; and $beacon$ is $\bigwedge_i \overline{b}^i_n \neq \mathsf{O}^1(\overline{b}^i_n) \to \mathsf{O}^1(\overline{b}^i_n) = \mathsf{O}^1(\overline{x}_n)$ (the agent can move a beacon only to its new location).

It is not hard to check that for every sequence σ of valuations, $\sigma \models \psi_{\mathcal{A},\phi,\omega}$ if and only if there exists $G \in \mathcal{G}_\phi$ and a run ρ of $\mathcal{A}$ on G according to scheduler S_ω such that *(i)* $\sigma \restriction \mathsf{avar}$ is equal to $loc(\rho)$; *(ii)* $\sigma \restriction \{s_n : n \in [N]\}$ is equal to $state(\rho)$; *(iii)* $\sigma \restriction \mathsf{ovar} \cup \{l\}$ is a sequence of identical valuations, each satisfying ϕ.

Thus, $\psi_{\mathcal{A},\phi,\omega} \to \tau$ is not valid if and only if there exists a sequence σ satisfying $\psi_{\mathcal{A},\phi,\omega} \wedge \neg\tau$ if and only if there exists $G \in \mathcal{G}_\phi$ and a run ρ of $\mathcal{A}$ on G according to scheduler S_ω such that the sequence $val(\rho)$ of valuations of the run ρ satisfies $\neg\tau$ if and only if the agents $\mathcal{A}$ do not achieve the task τ according to scheduler S_ω on all environments from $\mathcal{G}_\phi$. This completes the proof. $\square$

Before presenting a PSPACE lower-bound for the PVP, we need some definitions and a lemma.

Let AP be a set of atomic propositions, and let $\mathsf{AP}_{ig} := \mathsf{AP} \cup \{ignore\}$, where $ignore$ is a new atomic proposition not in AP. Let $\Sigma := 2^{\mathsf{AP}}$ and $\Sigma_{ig} := 2^{\mathsf{AP}_{ig}}$. Given a word $w' \in \Sigma^\omega_{ig}$, let $sub(w)$ be the word obtained by deleting all letters in w that are not in Σ (i.e., which contain $ignore$).

Lemma 1. *Let ϕ be an* LTL *formula over atomic propositions* AP. *One can compute in polynomial time an* LTL *formula ϕ' over AP_{ig}, such that for every $w' \in \Sigma^\omega_{ig}$ we have that $w' \models \phi'$ iff $sub(w)$ is infinite and models ϕ.*

Proof. W.l.o.g., we assume that ϕ is in positive normal form (i.e., negations are pushed to the atoms). The required formula is the conjunction of two formulas: the first requires that $\neg ignore$ holds infinitely often (thus ensuring that if $w' \models \phi'$ then $sub(w)$ is infinite), and the second "simulates" ϕ on the points where $\neg ignore$ holds (and "skipping" the points in which $ignore$ holds). Formally, $\phi' := \widehat{\phi} \wedge \mathsf{G}\,\mathsf{F}(\neg ignore)$, where $\widehat{\phi}$ is constructed by induction on the structure of ϕ as follows:

- if $\phi := t$, where t is an atomic proposition or its negation, then $\widehat{\phi} := ignore\, \mathsf{U} (\neg ignore \wedge t)$;

- if $\phi := \phi_1 \bowtie \phi_2$, where $\bowtie \in \{\vee, \wedge\}$, then $\widehat{\phi} := \widehat{\phi_1} \bowtie \widehat{\phi_2}$;
- if $\phi := \mathsf{O}\, \phi_1$ then $\widehat{\phi} := \mathit{ignore}\, \mathsf{U}(\neg \mathit{ignore} \wedge \mathsf{O}\, \widehat{\phi_1})$;
- if $\phi := \phi_1 \,\mathsf{U}\, \phi_2$ then $\widehat{\phi} := (\mathit{ignore} \vee \widehat{\phi_1})\, \mathsf{U}(\neg \mathit{ignore} \wedge \widehat{\phi_2})$;
- if $\phi := \phi_1 \,\mathsf{R}\, \phi_2$ then $\widehat{\phi} := (\neg \mathit{ignore} \wedge \widehat{\phi_1})\, \mathsf{R}(\mathit{ignore} \vee \widehat{\phi_2})$.

This completes the proof. □

THEOREM 4. *PVP is PSPACE-hard already for the case of a single agent on singleton parameterized environments with no obstacles and no beacons.*

PROOF. We reduce the problem of validity of LTL formulas, which is PSPACE-hard, to the PVP. Let ϕ be an LTL formula over the atomic propositions (w.l.o.g.) $\mathsf{AP} := \{1, \ldots, D\}$. Consider the parameterized environment G containing the single D-dimensional grid with sides of length 1 (i.e., the discrete d-dimensional unit hypercube), with no obstacles. Note that every position on this grid is a vector $\overline{x} \in \{0,1\}^D$. Consider a single agent A with two states $\top, \bot$. The transition relation allows the agent, from each state $\top, \bot$, to transition to either $\top, \bot$ while moving in any direction (or staying in the same place).

A configuration c of A on G encodes a set $X \in \Sigma_{ig}$ of atomic propositions in AP_{ig} as follows: $i \in [D]$ is in X iff the i'th coordinate of the location of A is 1, and $\mathit{ignore} \in X$ iff the local state of A is $\bot$. Hence, by the definition of A, for every word $w' \in \Sigma_{ig}^{\omega}$ there is some run of A on G that encodes w', and vice-versa. Also, using the same encoding (of atomic propositions of AP_{ig} as constraints on the location/state of the agent) we can write, with a linear blowup, a task τ that is a C-LTL formula that encodes the LTL formula ϕ' we obtain from ϕ using Lemma 1.

By Lemma 1, ϕ is valid iff ϕ' is valid. By the reasoning above, the later is true iff the output of the PVP is true on the input: agent A, a formula describing the hypercube G, and the task τ (and the unrestricted scheduler true). □

Observe that, in the proof above, the location of the robot encodes the values of all atomic propositions in parallel, which is the reason D dimensions are used. However, we can make do instead with a single dimension if we encode the values in serial. I.e., if we partition the positions (over time) of the robot into blocks of length D, and let the j position inside a block encode the value of the j'th atomic proposition at the time corresponding to the block number. I.e., for $i \in \mathbb{N}_0$, and $1 \leq j \leq D$, the position of the robot at time $iD+j$ encodes the value of atomic proposition number i at time j. By suitably modifying ϕ' and $\widehat{\phi}$ to this new encoding we can deduce that the problem remains PSPACE-hard also in the interesting cases of 2 and 3-dimensional grids.

7. DISCUSSION

Parameterized verification of multi-agent systems is a hard problem, and the case of most interest, i.e., that of 2 and 3-dimensional grids, even more so. Indeed, in [2] it is shown that (for agents with precise positioning) while the problem is decidable for many parameterized environments it is undecidable even for agents working on a 1-D grid with only collision-detection abilities. In this work we gave one way to regain decidability. We have presented a model for describing multiple agents moving in parameterised grid environments of arbitrary size with a fixed number of obstacles of arbitrary sizes. We have shown that relaxing the requirement that agents can not specify exact absolute positioning or step size, or the requirement of having a bounded number of obstacles, results in undecidability of the verification problem of very simple tasks of even a single agent. Our model of agents is very powerful (without becoming undecidable), and it generalises many other models (such as vision and touch based agents).

Many works in the past reason about environments by using all kinds of abstractions into graphs [9, 10, 44, 46]. Our approach avoids such abstractions (except for discretising space) by directly reasoning about the grid environments. We believe that our choice of using C-LTL as a logic for specifying both agents, parameterised environments, and tasks, allows one to encode things in a direct and natural way, and yields an elegant automatic verification algorithm. We draw an analog to the case of timed-automata, which are extremely popular because they provide users with direct and natural modeling formalism. Indeed, timed-automata can be verified by a sound and complete "region abstraction" to finite graphs [1, 22, 23], but working directly with these graphs is not convenient. A similar abstraction possibly exists for our framework, although it is not immediately clear or obvious how to define an abstraction with multiple agents, especially since they can stop "in the middle" of edges, and one can speak of their relative positions. Many forms of abstraction are not sound/complete. Thus, they do not allow one to map the border between decidability and undecidability as we do. Also, because an abstraction "throws away information", it must be tailor-made depending on the property to be verified.

Future Work. As mentioned above, one possible task is to try and come up with a form of "region abstraction" of our framework that is both sound and complete.

It is interesting to note that one can modify Theorem 1 to the case that the agent has access to absolute positioning with some error (i.e., $\pm\epsilon$, as would be provided by a real-world GPS system). However, it is not clear if it can be modified to the case of an agent that has no global positioning but can measure its step size with some error. This suggests that one may come up with a model for such agents that has a decidable parameterised verification problem (with or without relative positioning).

Two other natural directions of future work are the following. First, one can investigate in practice the actual performance of the presented framework by implementing the proposed algorithms. Although the complexity is PSPACE in general, one can hope — as it is with many algorithms in formal verification — that in many natural cases the algorithm may exhibit acceptable performance. Another possible direction is to try to extend this framework to get decidability results (with reasonable complexity) for multi-robot system scenarios that are rich enough to capture protocols found in the distributed computing literature, e.g., [5, 25, 26, 38].

Acknowledgments

B. Aminof and F. Zuleger are supported by the Austrian National Research Network S11403-N23 (RiSE) of FWF and by WWTF through grant ICT12-059. S. Rubin is a Marie Curie fellow of the Istituto Nazionale di Alta Matematica. A. Murano is partially supported by the GNCS 2016 project: Logica, Automi e Giochi per Sistemi Auto-adattivi.

REFERENCES

[1] R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking in dense real-time. *Inf. Comput.*, 104(1):2–34, 1993.

[2] B. Aminof, A. Murano, S. Rubin, and F. Zuleger. Verification of asynchronous mobile-robots in partially-known environments. In *PRIMA*, LNCS 9387, pages 185–200, 2015.

[3] E. M. Barrameda, S. Das, and N. Santoro. Uniform dispersal of asynchronous finite-state mobile robots in presence of holes. In *ALGOSENSORS*, LNCS 8243, pages 228–243, 2013.

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

[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] R. Bloem, S. Jacobs, A. Khalimov, I. Konnov, S. Rubin, H. Veith, and J. Widder. *Decidability of Parameterized Verification*, volume 6 (1) of *Synthesis Lectures on Distributed Computing Theory.* 2015.

[7] A. Blum, P. Raghavan, and B. Schieber. Navigating in unfamiliar geometric terrain. *SIAM J. Comput.*, 26(1):110–137, 1997.

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

[9] R. Brafman, J. Latombe, Y. Moses, and Y. Shoham. Applications of a logic of knowledge to motion planning under uncertainty. *J.ACM*, 44(5):633–668, 1997.

[10] W. Burgard, M. Moors, C. Stachniss, and F.Schneider. Coordinated multi-robot exploration. *IEEE Transactions on Robotics*, 21(3):376–386, 2005.

[11] A. Canedo-Rodríguez, V. Alvarez-Santos, C. V Regueiro, X. M. Pardo, and R. Iglesias. Multi-agent system for fast deployment of a guide robot in unknown environments. *Journal of Physical Agents*, 6(1):31–41, 2012.

[12] K. Cheng and P. Dasgupta. Coalition game-based distributed coverage of unknown environments by robot swarms. In *AAMAS*, pages 1191–1194, 2008.

[13] E. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT, 2002.

[14] R. Cohen, P. Fraigniaud, D. Ilcinkas, A. Korman, and D. Peleg. Label-guided graph exploration by a finite automaton. In *Automata, Languages and Programming*, LNCS 3580, pages 335–346. 2005.

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

[16] G. De Giacomo, P. Felli, F. Patrizi, and S. Sardiña. Two-player game structures for generalized planning and agent composition. In *AAAI*, 2010.

[17] S. Demri and D. D'Souza. An automata-theoretic approach to constraint LTL. *Inform. and Comp.*, 205(3):380 – 415, 2007.

[18] S. Demri and R. Gascon. Verification of qualitative Z constraints. *Theor. Comput. Sci.*, 409(1):24–40, 2008.

[19] A. Dessmark and A. Pelc. Optimal graph exploration without good maps. In *ESA*, LNCS 2461, pages 374–386. Springer, 2002.

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

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

[22] M. Faella, S. La Torre, and A. Murano. Dense real-time games. In *LICS*, pages 167–176. IEEE Computer Society, 2002.

[23] M. Faella, S. La Torre, and A. Murano. Automata-theoretic decision of timed games. *Theor. Comput. Sci.*, 515:46–63, 2014.

[24] A. Felner, R. Stern, and S. Kraus. PHA*: performing A* in unknown physical environments. In *AAMAS*, pages 240–247. ACM, 2002.

[25] P. Flocchini, G. Prencipe, and N. Santoro. Computing by mobile robotic sensors. In *Theoretical Aspects of Distributed Computing in Sensor Networks*, EATCS, pages 655–693. 2011.

[26] P. Flocchini, G. Prencipe, and N. Santoro. *Distributed Computing by Oblivious Mobile Robots*. Synthesis Lectures on Distributed Computing Theory. 2012.

[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 *Graph-Theoretic Concepts in Computer Science*, LNCS 5344, pages 14–29. Springer, 2008.

[29] A. Gilboa, A. Meisels, and A. Felner. Distributed navigation in an unknown physical environment. In *AAMAS*, pages 553–560, 2006.

[30] F. Hoffmann. One pebble does not suffice to search plane labyrinths. In *Fundamentals of Computation Theory*, LNCS 117, pages 433–444. 1981.

[31] T. Hsiang, E. M. Arkin, M. A. Bender, S. P. Fekete, and J. S. B. Mitchell. Algorithms for rapidly dispersing robot swarms in unknown environments. In *WAFR*, Springer Tracts in Advanced Robotics, vol. 7, pages 77–94. Springer, 2002.

[32] Y. Hu and G. De Giacomo. Generalized planning: Synthesizing plans that work for multiple environments. In *IJCAI*, pages 918–923, 2011.

[33] K. Hungerford, P. Dasgupta, and K. R. Guruprasad. Distributed, complete, multi-robot coverage of initially unknown environments using repartitioning. In *AAMAS*, pages 1453–1454. IFAAMAS, 2014.

[34] A. Khalimov, S. Jacobs, and R. Bloem. PARTY parameterized synthesis of token rings. In *CAV*, LNCS 8044, pages 928–933, 2013.

[35] A. Khalimov, S. Jacobs, and R. Bloem. Towards efficient parameterized synthesis. In *VMCAI*, LNCS 7737, pages 108–127, 2013.

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

[37] P. Kouvaros and A. Lomuscio. A counter abstraction technique for the verification of robot swarms. In *AAAI*, pages 2081–2088, 2015.

[38] E. Kranakis, D. Krizanc, and S. Rajsbaum. Mobile agent rendezvous: A survey. In *SIROCCO*, LNCS 4056, pages 1–9, 2006.

[39] E. Kranakis, D. Krizanc, and S. Rajsbaum.

Computing with mobile agents in distributed networks. In *Handbook of Parallel Computing: Models, Algorithms, and Applications. Chapter 8*. 2007.

[40] N. A. Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996.

[41] D. K. M. Blum. On the power of the compass (or, why mazes are easier to search than graphs). In *FOCS*, pages 132–142, 1978.

[42] M. L. Minsky. *Computation: finite and infinite machines.* Prentice-Hall, Inc., 1967.

[43] C. H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theor. Comput. Sci.*, 84(1):127–150, 1991.

[44] N. Rao, S. Kareti, W. Shi, and S. Iyengar. *Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms.* Jul 1993.

[45] S. Rubin. Parameterised verification of autonomous mobile-agents in static but unknown environments. In *AAMAS*, pages 199–208, 2015.

[46] K. Senthilkumar and K. Bharadwaj. Multi-robot exploration and terrain coverage in an unknown environment. *Robotics and Autonomous Systems*, 60(1):123–132, 2012.