A Counter Abstraction Technique for the Verification of Probabilistic Swarm Systems

Alessio Lomuscio
Imperial College London
London, United Kingdom
a.lomuscio@imperial.ac.uk

Edoardo Pirovano
Imperial College London
London, United Kingdom
e.pirovano17@imperial.ac.uk

ABSTRACT

We introduce a method for formally verifying properties of arbitrarily large swarms whose agents are modelled probabilistically. We define a parameterised probabilistic semantics for modelling swarms and observe that their verification problem against PLTL specifications is undecidable. We develop a partial procedure for verifying arbitrarily large swarms based on counter abstraction and show its correctness. We present an implementation and report the experimental results obtained when verifying a swarm foraging protocol.

ACM Reference Format:
Alessio Lomuscio and Edoardo Pirovano. 2019. A Counter Abstraction Technique for the Verification of Probabilistic Swarm Systems. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019,* IFAAMAS, 9 pages.

1 INTRODUCTION

Robotics swarms have been proposed as an alternative to single-robot systems in many applications including surveillance [38] and search and rescue [32]. Swarms have been shown to provide a number of advantages such as scalability and fault-tolerance. Typically, physical agents in a swarm follow simple protocols in line with their limited computation and sensing ability. While their individual behaviours may be simple, their interaction can lead to sophisticated overall behaviour that is difficult to predict [5, 36, 37].

At run-time a swarm system will typically be composed of a variable number of agents, depending on a number of factors such as the size of the problem being solved, the level of performance required, and whether a degree of fault-tolerance is necessary. Thus, it is desirable to verify at design-time that the protocols followed by the agents are correct regardless of the number of agents that will be deployed at run-time. This cannot be achieved by traditional model-checking techniques as they are limited to finite-state systems and thus can only verify swarms composed of a fixed number of agents.

Parameterised model checking can be used to verify systems of arbitrary size [4]. In its most general formulation, the parameterised model checking problem is known to be undecidable [2]. Nonetheless, by limiting the communication patterns between agents or the specification language, decidable fragments of this problem have been explored, particularly in the context of network systems [1, 7].

Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 13–17, 2019, Montreal, Canada.

In one direction of work a semantics amenable to modelling multi-agent systems has been proposed and parameterised verification methods based on identifying cutoffs (a sufficient number of agents to display every behavior) have been developed [20, 24].

Adaptations and extensions of these methods have enabled the verification of unbounded swarm protocols such as the alpha algorithm [22]. A key limitation of this line of work is that it does not fully support the modelling of probabilistic aspects of swarms. This hinders the applicability to protocols in which the behaviour of the agents is stochastic and thus cannot be modelled in a meaningful way using only non-determinism. Probabilistic aspects were incorporated to some extent in [23], but this was done in an ad-hoc way to consider an opinion formation protocol [8]. Recently [31] introduced a more general semantics for reasoning about probabilistic swarms. However, as discussed in the related work, this had a number of limitations, thereby limiting its applicability.

The aim of this contribution is to introduce a novel formal verification method, including an expressive probabilistic agent-based semantics and a paramaterised verification technique, for reasoning about unbounded swarm systems. As is typical in parameterised verification, this will result in an undecidable verification problem. Nonetheless, we develop a partial decision procedure for it based on counter abstraction [33] that enables us to derive conclusions on all the infinitely many swarm systems. Building from these theoretical results we introduce an open-source implementation which we use to evaluate a probabilistic swarm foraging algorithm [6, 29].

After discussing related work, the rest of the paper is organised as follows. In Section 2 we give some background on probabilistic model checking, and introduce notation that will be used throughout the paper. In Section 3, we present our novel semantics for reasoning about probabilistic swarm systems, and present the decision problem we will consider. In Section 4, we develop a partial decision procedure based on counter abstraction and show its correctness. In Section 5 we present an implementation of this procedure, and evaluate its performance on a foraging protocol. Finally, we conclude in Section 6.

Related Work. There has been a significant amount of work in verifying properties of probabilistic swarm systems [11, 15, 19, 40]; some work has included strategy synthesis aspects [13]. However, these lines have focussed on swarm systems with a bounded number of agents. Recent developments [12] have aimed to increase the maximum number of agents that can be checked; however the number of agents that will be present in the system is unknown at design-time and may be larger than what could possibly be checked. In contrast, we here present a method to investigate whether properties hold on systems of any size.

Work in parameterised model-checking for unbounded multi-agent systems and swarms [20, 22, 24] aims to give the same guarantees that we pursue here; however this line is not usually concerned with stochastic aspects and therefore it is not comparable to the present contribution. One exception is [23], where a semantics specifically tailored to opinion formation is proposed. The semantics we consider here is more general and allows us to consider not one but various protocols as well as quantitative probabilistic properties in addition to the qualitative ones considered there.

A related area of work in probabilistic verification of network protocols [10, 14] has enabled verification of probabilistic protocols with an arbitrarily large number of agents. However, the communication pattern used between agents in these semantics is not amenable to modelling swarms.

Closest to the present contribution is [31] where a stochastic treatment of swarm systems amenable to establishing whether emergent properties arise is put forward. In contrast, in this paper we focus on the parametrised verification problem and not emergence identification. Moreover, the specification language we analyse is more expressive, allowing us to express properties on infinite traces and not just properties on finite traces as considered in [31]. Lastly, our semantics allows for interleaved executions in which the agents take turns to perform actions. This is considerably more expressive than the fully synchronous semantics considered in [31], thereby enabling the analysis of further swarm protocols.

2 BACKGROUND

In this section we introduce some background on probabilistic model checking, along with the notation that we will use throughout the paper.

Discrete Time Markov Chains. We briefly summarise *discrete time Markov chains* (DTMCs). For more background on model checking DTMCs, see [3, 17, 26].

Definition 2.1 (DTMC). A *discrete-time Markov chain* (DTMC) is a tuple $\mathcal{D} = \langle S, \iota, t, L\rangle$ where S is a set of states, $\iota \in S$ is a distinguished initial state, $t : S \times S \to [0, 1]$ is a transition probability function (with $\sum_{s' \in S} t(s, s') = 1$ for any $s \in S$) and $L : S \to \mathcal{P}(AP)$ is a labelling function on a set AP of atomic propositions.

A *path* in a DTMC is a sequence of states $s_0 s_1 s_2 \ldots$ such that for every $i \in \mathbb{N}$ it is the case that $t(s_i, s_{i+1}) > 0$. We use $FPath_{\mathcal{D}}$ and $IPath_{\mathcal{D}}$ respectively, to denote the set of all finite and infinite paths starting from the initial state ι. For a finite path we define its probability by $\mathbf{P}_{\mathcal{D}}(s_0 \ldots s_n) \triangleq \prod_{i=0}^{n-1} t(s_i, s_{i+1})$. Following [17], this can be extended to the set of all infinite paths. For ease of presentation we omit this; however note that this probability space is uniquely defined by the probabilities on finite paths. This corresponds to the intuition that the probability of an infinite path occurring is uniquely defined by the probability of its finite prefixes occurring. We also define the probability of a set of paths occurring by summing over them, i.e. for a set $X \subseteq (FPath_{\mathcal{D}} \cup IPath_{\mathcal{D}})$ we define $\mathbf{P}_{\mathcal{D}}(X) \triangleq \sum_{\rho \in X} \mathbf{P}_{\mathcal{D}}(\rho)$. Further, we sometimes omit the subscript from $\mathbf{P}_{\mathcal{D}}$ and simply write $\mathbf{P}$ when $\mathcal{D}$ is clear from context.

Markov Decision Processes. We now summarise some key aspects of *Markov decision processes* (MDPs). We refer to [3, 35] for more details. We mostly follow the notation used in [9].

Definition 2.2 (MDP). A *Markov decision process* (MDP) is a tuple $\mathcal{M} = \langle S, \iota, A, P, t, L\rangle$ where S is a finite set of states, $\iota \in S$ is a distinguished initial state, A is a finite set of actions, $P : S \to \mathcal{P}(A)$ is a protocol function (such that $P(s) \neq \emptyset$ for all $s \in S$), $t : S \times A \times S \to [0, 1]$ is a transition function (with $\sum_{s' \in S} t(s, a, s') = 1$ for any $s \in S$ and $a \in P(s)$) and $L : S \to \mathcal{P}(AP)$ is a labelling function on a set AP of atomic propositions.

Intuitively, a transition from a state s of an MDP occurs by first non-deterministically selecting some action $a \in P(s)$ and then transitioning to state s' with probability $t(s, a, s')$. MDPs thus give a way of describing systems that include both probabilistic and non-deterministic choice, unlike DTMCs which do not capture the latter.

A *path* in an MDP is a sequence of states and actions $s_0 a_0 s_1 a_2 \ldots$ such that for all $i \in \mathbb{N}$ it is the case that $a_i \in P(s_i)$ and $t(s_i, a_i, s_{i+1}) > 0$. We use $FPath_{\mathcal{M}}$ ($IPath_{\mathcal{M}}$, respectively) to denote the set of all finite (infinite, respectively) paths starting from the initial state ι. For a finite path $\rho = s_0 a_0 \ldots s_n$, $last(\rho) \triangleq s_n$ denotes its last state.

In order to reason about the probability of a path occurring in an MDPs, we need a way to resolve the inherent non-determinism. This is captured by a scheduler (also referred to as an *adversary*, *strategy* or *policy* in some literature).

Definition 2.3 (Scheduler). Given an MDP $\mathcal{M} = \langle S, \iota, A, P, t, L\rangle$ a *scheduler* for $\mathcal{M}$ is a function $\sigma : FPath_{\mathcal{M}} \times A \to [0, 1]$ such that for any finite path $\rho \in FPath_{\mathcal{M}}$, we have $\sigma(\rho, a) > 0$ only if $a \in P(last(\rho))$ and $\sum_{a \in A} \sigma(\rho, a) = 1$.

We denote by $Adv_{\mathcal{M}}$ the set of all schedulers for $\mathcal{M}$. Various classes of schedulers may be defined [9]. Note that when maximising or minimising the probability of reaching a target set of states, it is sufficient to consider schedulers that are memoryless (only make choices based on the final state of the path) and deterministic (only assign values from $\{0, 1\}$ to all actions).

We now proceed to define the DTMC induced by a scheduler on an MDP. Intuitively, this describes the purely probabilistic system that results from fixing a given choice of scheduler in an MDP.

Definition 2.4 (Induced DTMC). Given an MDP $\mathcal{M} = \langle S, \iota, A, P, t, L\rangle$ and a strategy $\sigma : FPath_{\mathcal{M}} \times A \to [0, 1]$, the *induced DTMC* $\mathcal{M}_\sigma = \langle FPath_{\mathcal{M}}, \iota, t', L'\rangle$ is defined by:

- $t' : FPath_{\mathcal{M}} \times FPath_{\mathcal{M}} \to [0, 1]$ is given by:

$$t'(\rho, \rho') \triangleq \begin{cases} \sigma(\rho, a) \times t(last(\rho), a, s) & \text{if } \rho' = \rho a s, a \in P(last(\rho)) \\ 0 & \text{otherwise} \end{cases}$$

- $L'(\rho) \triangleq L(last(\rho))$ for all $\rho \in FPath_{\mathcal{M}}$

Notice that while in general the induced DTMC might have an infinite number of states, when considering memoryless schedulers it is possible to derive an equivalent DTMC that has only finitely many states [9].

3 PROBABILISTIC SWARM SYSTEMS

In this section we introduce the syntax and semantics of probabilistic swarm systems and define their model checking problem.

Models. We begin by defining probabilistic parameterised interleaved interpreted systems (PPIISs). A PPIIS is composed of an *agent template*, which describes the behaviour of individual agents and

an *environment* which captures the behaviour of the other parts of the system. It is straightforward to extend the framework to finitely many agent templates. For ease of presentation we do not carry out this exercise here.

Definition 3.1 (Probabilistic agent template). A *probabilistic agent template* is a tuple $T = \langle S, \iota, Act, P, t\rangle$ where:

- The set S is a finite set of agent local states.
- $\iota \in S$ is a distinguished initial state.
- $Act = A \cup AE \cup GS$ is the non-empty set of actions that can be performed by the agents. These may either be *asynchronous actions*, *agent-environment actions* or *global-synchronous actions*. Each type of action implies a different communication pattern between the agents, as outlined in Definition 3.4.
- The agent's protocol function $P : S \to \mathcal{P}(Act)$ defines which actions are enabled at a given state.
- The agent's transition function $t : S \times Act \times S \to [0, 1]$ describes the evolution of the agent's state: given a local state s, an action a, and a local state s', t returns the probability that upon performing action a in state s the agent will transition to state s'. Notice we require that for every $l \in S$ and $a \in P(l)$ we have $\sum_{l' \in S} t(l, a, l') = 1$.

The agent template is closely related to MDPs (see Section 2), suitably extended to encode action types to account for synchronisation purposes.

For the remainder of this paper we will assume that agents transition deterministically when performing global actions, i.e., if $a \in GS$ then $t(s, a, s') \in \{0, 1\}$. This restriction will simplify the presentation, but does not limit the expressivity of the formalism. More precisely, systems with probabilistic global transitions can be transformed into equivalent systems with deterministic global transformations by introducing a further asynchronous probabilistic action selecting the resulting global transition.

We now proceed to define the environment that the agents operate in.

Definition 3.2 (Environment). An *environment* E is a tuple $E = \langle S_E, \iota_E, Act_E, P_E, t_E\rangle$ where S_E is a finite set of local states, $\iota_E \in S_E$ is a distinguished initial state, Act_E is a non-empty set of actions $Act_E = A_E \cup AE \cup GS$, the environment protocol P_E is a function $P_E : S_E \to \mathcal{P}(Act_E)$, and a transition function $t_E : S_E \times Act_E \times S_E \to [0, 1]$ such that for every $l \in S_E$ and $a \in P_E(l)$ we have $\sum_{l' \in S_E} t(l, a, l') = 1$.

As above an environment is an MDP suitably extended with action types. We can now define PPIIS.

Definition 3.3 (PPIIS). A *probabilistic parameterised interleaved interpreted system* (PPIIS) is a tuple $\mathcal{S} = \langle T, E, L\rangle$, where T is a probabilistic agent template, E is an environment and $L : S \times S_E \to \mathcal{P}(AP)$ is a labelling function for a set of atomic propositions AP.

PPIISs, as defined above, extend the framework of parameterised interpreted systems (PIISs) presented in [24] to reason about multi-agent systems composed of an unbounded number of agents. In turn, parameterised interpreted systems extend interleaved interpreted systems (IISs) [30]. The framework we introduced above is a modification of PIISs to account for probabilistic behaviour of the agents and the environment.

Each PPIIS describes an unbounded collection of concrete systems obtained by choosing a different number of agents in the system. Given a PPIIS $\mathcal{S}$ and $n \in \mathbb{Z}^+$, the system $\mathcal{S}(n)$ of n agents is the result of the composition of n copies of T with the environment. We denote the set of concrete agents instantiated from T by $Agt_n \triangleq \{1, \ldots, n\}$. When referring to a subset of these agents, we will use the notation $Agt_{x,y} \triangleq \{x, \ldots, y\}$ when this is well-defined.

A *global state* $g = \langle s_1, \ldots, s_n, s_e\rangle$ is an $(n + 1)$-tuple of local states for all the agents and the environment in $\mathcal{S}(n)$; it describes the system at a particular instant of time. Let S_n denote the set of all such global states. For a global state $g \in S_n$ we write $g.i$ to denote the local state of agent i in g and $g.E$ to denote the state of the environment in g.

The set of global actions is given by $Act_n \triangleq GS \cup A_E \cup ((A \cup AE) \times Agt_n)$. Thus, each action is either a global synchronous one, an asynchronous environment one, an asynchronous agent one, or an agent-environment one; the latter two are labelled by the agent involved in the action. The actions that are enabled in each global state are defined by the global protocol.

Definition 3.4 (Global protocol). The *global protocol* $P_n : S_n \to \mathcal{P}(Act_n)$ is defined by $a \in P_n(g)$ if and only if

- *(Global-synchronous).* (i) $a \in GS$; (ii) for all $i \in Agt_n$, $a \in P(g.i)$; (iii) $a \in P_E(g.E)$.
- *(Asynchronous environment).* (i) $a \in A_E$; (ii) $a \in P_E(g.E)$.
- *(Asynchronous agent).* (i) $a = (a', i) \in A \times Agt_n$; (ii) $a' \in P(g.i)$.
- *(Agent-environment).* (i) $a = (a', i) \in AE \times Agt_n$; (ii) $a' \in P(g.i)$; (iii) $a' \in P_E(g.E)$.

Thus, global actions are enabled only if they are enabled for all agents and the environment. Asynchronous environment actions must be enabled for the environment, whilst asynchronous agent actions must be enabled for the participating agent. Finally, agent-environment actions must be enabled for both the participating agent and the environment.

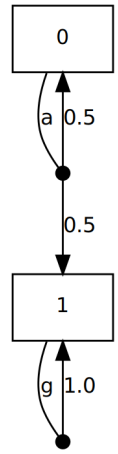
Transitions in the system occur according to the global transition function, which we will now define.

Definition 3.5 (Global transition function). The *global transition function* $t_n : S_n \times Act_n \times S_n \to [0, 1]$ is defined by $t_n(g, a, g')$

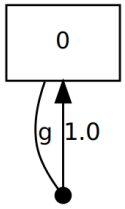
$$\triangleq \begin{cases} t_E(g.E, a, g'.E) \cdot \prod_{i=1}^{n} t(g.i, a, g'.i) & \text{if } a \in GS \\ t_E(g.E, a, g'.E) & \text{if } a \in A_E \\ & \quad \text{and } \forall i \in Agt_n : g.i = g'.i \\ t(g.i, a', g'.i) & \text{if } a = (a', i) \in A \times Agt_n \text{ and} \\ & \quad \forall j \in Agt_n \setminus \{i\} : g.j = g'.j \\ t_E(g.E, a', g'.E) \cdot t(g.i, a', g'.i) & \text{if } a = (a', i) \in AE \times Agt_n \text{ and} \\ & \quad \forall j \in Agt_n \setminus \{i\} : g.j = g'.j \\ 0 & \text{otherwise} \end{cases}$$

Intuitively, the global transition function is defined by multiplying the transition functions of the agents participating in the transition while ensuring that any agents not participating in the transition remain in the same state. The observation below follows immediately from the definitions.

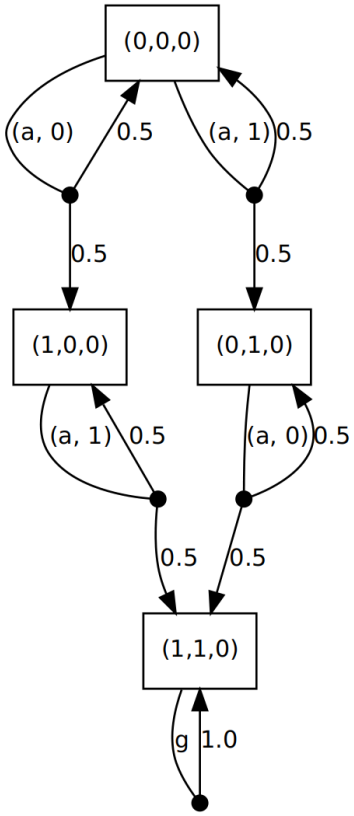
Observation 1. *For any $g \in S_n$ and $a \in P_n(g)$, it is the case that* $\sum_{g' \in S_n} t_n(g, a, g') = 1$.



(a) An agent template.



(b) An environment.



(c) The concrete system of 2 agents.

Figure 1: An example PPIIS together with the concrete instantiation for two agents. The a action is asynchronous, while the g one is global synchronous.

This observation will enable us to show that a concrete system of n agents, defined below, is an MDP. Notice that in a global system the scheduler chooses both which agents act and what action they perform. It does not, however, choose the transition, which occurs according to the probability distribution defined in Definition 3.5.

Definition 3.6 (Concrete system). Given a PPIIS $\mathcal{S}$ and an $n \in \mathbb{Z}^+$, the *concrete system* of n agents is defined by $\mathcal{S}(n) = \langle S_n, \iota_n, Act_n, P_n, t_n, L_n \rangle$, where $\iota_n = \langle \iota, \ldots, \iota, \iota_E \rangle$, the labelling function $L_n : S_n \to \mathcal{P}(AP \times Agt_n)$ is defined by $L_n(g) \triangleq \{(p, i) \in AP \times Agt_n : p \in L(g.i, g.E)\}$, and the other components are defined as in Definitions 3.4 and 3.5.

Notice that for the labelling function we create one copy of an atomic proposition for each agent, and label a global state with this if the agent's local state is labelled with it.

A small example of a PPIIS and its corresponding concrete system built on two agents is shown in Figure 1. Here, the agents begin in state 0, where they can only perform the asynchronous action a, which with equal probability either does nothing or takes the agent to state 1. Once all agents are in state 1, they can perform the global synchronous action g which does not cause a change in state. The environment has only the state 0 which always enables g. We will exemplify the use of PPIIS in swarm robotics in Section 5.

Having defined the models, we now define the specification language that will be used to reason about properties of swarms.

Specifications. We express properties of swarm systems by means of a variant of the probabilistic LTL logic [9], tailored to expressing properties of multi-agent systems by labelling atomic propositions with the agent that they should hold for.

Definition 3.7 (PLTL). For $a \in AP$ and $i \in \mathbb{N}$, the *probabilistic LTL* logic is the set of formulas ϕ defined by the following BNF:

$$\phi ::= P^{\max}_{\bowtie x}[\psi] \mid P^{\min}_{\bowtie x}[\psi] \text{ for } x \in [0,1] \text{ and } \bowtie \in \{\leq, <, \geq, >\}$$
$$\psi ::= \top \mid (a, i) \mid \neg\psi \mid \psi \wedge \psi \mid X\psi \mid \psi \; U \; \psi$$

We say a formula is m-indexed if it refers to agents with index at most m (i.e., all atomic propositions in the formula are of the form (a, i) for some $i \leq m$).

Intuitively, the property $P^{\max}_{\bowtie x}[\psi]$ holds if, with a scheduler that is trying to maximise the probability of a path satisfying ψ occurring, this probability is $\bowtie x$. Similarly, $P^{\min}_{\bowtie x}[\psi]$ holds if the probability is $\bowtie x$ with a scheduler that is attempting to minimise it. A path satisfies $X\psi$ if it satisfies ψ in its second state; it satisfies $\psi_1 U \psi_2$ if it satisfies ψ_2 at some point, and satisfies ψ_1 until then. We assume the abbreviations F, G, and finite time variants of all operators, as usual.

We now define the satisfaction relation for PLTL.

Definition 3.8 (Satisfaction). Given a concrete swarm system $\mathcal{S}(n)$ and ϕ an m-indexed formula, where $n \geq m$, the satisfaction of ϕ on $\mathcal{S}(n)$ is inductively defined as follows:

$$\mathcal{S}(n) \models P^{\max}_{\bowtie x}[\psi] \text{ iff } \sup_{\sigma \in Adv_{\mathcal{M}}} \mathbf{P}(\{\rho \in IPath_{\mathcal{S}(n)_\sigma} : \rho \models \psi\}) \bowtie x$$

$$\mathcal{S}(n) \models P^{\min}_{\bowtie x}[\psi] \text{ iff } \inf_{\sigma \in Adv_{\mathcal{M}}} \mathbf{P}(\{\rho \in IPath_{\mathcal{S}(n)_\sigma} : \rho \models \psi\}) \bowtie x$$

Satisfaction for path formulae is defined as usual in LTL.

For example, the PLTL formula $P^{\max}_{\leq 0.3}[F^{<5}(\text{win}, 1)]$ is a 1-indexed formula that holds if no matter what choices the scheduler makes, the probability of agent 1 reaching a state where win holds within 5 time-steps cannot exceed 0.3. The PLTL formula $P^{\min}_{<0.9}[G(\text{alive}, 2)]$ is a 2-indexed formula that holds if, when the scheduler tries to minimise the probability that agent 2 is always in a state where alive holds, then this probability will be below 0.9.

For the remainder of the paper with no loss of generality we will only consider properties of the form $P^{\max}_{\leq x}[\psi]$. Note in fact that the other cases can either be handled similarly or transformed into equivalent cases. For instance, checking that $P^{\min}_{>x}[\psi]$ holds is equivalent to checking that $P^{\max}_{\leq x}[\neg\psi]$ does not hold.

In the following we will focus on the parameterised model checking problem, which amounts to checking whether a certain formula holds in systems of any size. We formalise this below.

Definition 3.9 (Parameterised Model Checking). Given a PPIIS $\mathcal{S}$ and an m-indexed PLTL formula ϕ, the parameterised model checking problem involves establishing whether it is the case that $\mathcal{S}(n) \models \phi$ for all $n \geq m$. We write $\mathcal{S} \models \phi$ if this is the case.

Notice that since this is a probabilistic extension of a problem that is known to be undecidable in general [2], it is certainly also

undecidable in general. In the following we will explore decidable fragments.

4 MODEL CHECKING PPIIS

In this section we introduce a procedure for verifying swarm systems modelled by PPIISs. To do so, we introduce an abstract model, which captures all the possible paths in concrete systems of any size. The abstract model that we consider is inspired by the literature in counter abstraction [33] as well as its adaptation in swarm systems [21]. We are not aware, however, of these methods having been explored in a probabilistic setting.

First, consider that to evaluate an m-indexed formula, we need to represent the state of at least the first m agents in the system. Accordingly, the state of the abstract model will have two components: the first will represent the state of the first m agents and the environment; the second will consist of a set containing the states of all the other agents (abstracting away the information about how many agents are in each state).

Transitions of the first m agents will occur exactly as before; for transitions involving the remaining agents we will label the action with the local state the transition was occurring from and whether it was a "shrinking" ($\downarrow$) transition or a "growing" ($\uparrow$) transition. A shrinking transition will represent that there was only one agent in the local state the action was performed from. Conversely, a growing transition will represent that there were at least two agents in the local state the action was performed from.

Definition 4.1 (Abstract system). Given a PPIIS $\mathcal{S}$ and an $m \in \mathbb{Z}^+$, the *abstract system* of m agents is defined by $\hat{\mathcal{S}}(m) = \langle \hat{S}_m, \hat{\iota}_m, \hat{Act}_m, \hat{P}_m, \hat{t}_m, \hat{L}_m \rangle$, where:

- $\hat{S}_m \triangleq S_m \times \mathcal{P}(S)$ is the set of abstract global states.
- $\hat{\iota}_m \triangleq (\iota_m, \{\iota\})$ is the initial abstract global state.
- $\hat{Act}_m \triangleq Act_m \cup ((A \cup AE) \times S \times \{\uparrow, \downarrow\})$ is the set of abstract global actions.
- $\hat{P}_m : \hat{S}_m \to \mathcal{P}(\hat{Act}_m)$ is defined by:

$$\begin{aligned}\hat{P}_m(g,X) \triangleq &(P_m(g) \setminus \{a \in GS : \exists x \in X \text{ with } a \notin P(x)\}) \\ &\cup \{(a,s,v) \in A \times X \times \{\uparrow,\downarrow\} : a \in P(s)\} \\ &\cup \{(a,s,v) \in AE \times X \times \{\uparrow,\downarrow\} : a \in P(s) \cap P_E(g.E)\}\end{aligned}$$

- $\hat{t}_m : \hat{S}_m \times \hat{Act}_m \times \hat{S}_m \to [0,1]$ is the transition function, defined for asynchronous agent actions by:

$$\hat{t}_m((g,X),(a,l,\uparrow),(g',X')) \triangleq \begin{cases} t(l,a,l') \text{ if } X' \setminus X = \{l'\} \text{ and } g = g' \\ \sum_{l' \in X} t(l,a,l') \text{ if } X' = X \text{ and } g = g' \\ 0 \text{ otherwise} \end{cases}$$

$$\hat{t}_m((g,X),(a,l,\downarrow),(g',X')) \triangleq \begin{cases} t(l,a,l') \text{ if } X' = (X \setminus \{l\}) \cup \{l'\} \text{ and } g = g' \\ \sum_{l' \in X'} t(l,a,l') \text{ if } X' = X \setminus \{l\} \text{ and } g = g' \\ 0 \text{ otherwise} \end{cases}$$

$$\hat{t}_m((g,X),(a,i),(g',X')) \triangleq \begin{cases} t_m(g,a,g') \text{ if } X' = X \\ 0 \text{ otherwise} \end{cases}$$

For agent-environment actions, we similarly define:

$$\hat{t}_m((g,X),(a,l,\uparrow),(g',X')) \triangleq t(g.E,a',g'.E) \times \begin{cases} t(l,a,l') \text{ if } X' \setminus X = \{l'\} \text{ and } \forall i \in Agt_m : g.i = g'.i \\ \sum_{l' \in X} t(l,a,l') \text{ if } X' = X \text{ and } \forall i \in Agt_m : g.i = g'.i \\ 0 \text{ otherwise} \end{cases}$$

$$\hat{t}_m((g,X),(a,l,\downarrow),(g',X')) \triangleq t(g.E,a',g'.E) \times \begin{cases} t(l,a,l') \text{ if } X' = (X \setminus \{l\}) \cup \{l'\} \\ \qquad \text{and } \forall i \in Agt_m : g.i = g'.i \\ \sum_{l' \in X'} t(l,a,l') \text{ if } X' = X \setminus \{l\} \\ \qquad \text{and } \forall i \in Agt_m : g.i = g'.i \\ 0 \text{ otherwise} \end{cases}$$

$$\hat{t}_m((g,X),(a,i),(g',X')) \triangleq \begin{cases} t_m(g,a,g') \text{ if } X' = X \\ 0 \text{ otherwise} \end{cases}$$

For asynchronous environment actions, we define:

$$\hat{t}_m((g,X),a,(g',X')) \triangleq \begin{cases} t_m(g,a,g') \text{ if } X' = X \\ 0 \text{ otherwise} \end{cases}$$

Finally, for global actions we define:

$$\hat{t}_m((g,X),a,(g',X')) \triangleq \begin{cases} t_m(g,a,g') \text{ if } X' = \{s' \in S | \exists s \in X : t(s,a,s') = 1\} \\ 0 \text{ otherwise} \end{cases}$$

- $\hat{L}_m : \hat{S}_m \to \mathcal{P}(AP \times Agt_m)$ is the labelling function given by $\hat{L}_m(g,X) \triangleq L_m(g)$.

The protocol enables those actions that would have been enabled for the first m agents, unless they are global synchronous actions which one of the remaining agents cannot perform. It also enables asynchronous and agent-environment actions (both growing and shrinking), for the remaining agents.

For growing asynchronous transitions, we either use the probability of transitioning to a new state l' if the set of states grows, or the sum of the probabilities of transitioning to one of the existing states if the set of states does not change. The case for shrinking transitions is similar. Finally, transitions of the first m agents are handled as before by simply re-using t_m and enforcing that none of the other agents change state.

Agent-environment actions are handled almost identically to asynchronous ones, except that they also include the probability of the environment performing the transition it does.

Lastly, global-synchronous actions can be handled in a straightforward way by using our assumption that the agents transition deterministically when performing a global action.

Finally, notice that the labelling function simply discards the information not pertaining to the local state of the first m agents, since this is not needed to evaluate the formula.

An example of an abstract system can be seen in Figure 2. This is obtained by applying Definition 4.1 with $m = 1$ to the PPIIS previously introduced in Figure 1. For instance, in the initial state the first agent is in state 0, as is the environment and all other agents. This gives us an initial state of $((0,0),\{0\})$. In this state, the first agent could perform the action a. This is denoted by the action

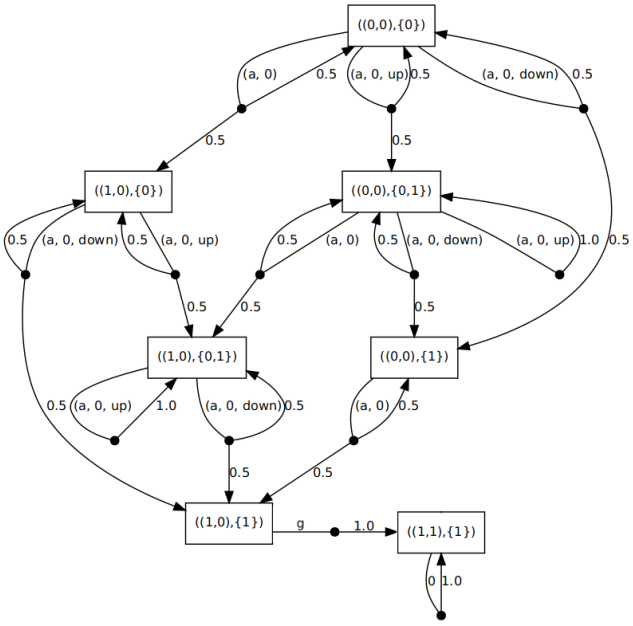


Figure 2: The abstract system with one agent for the PPIIS introduced in Figure 1.

$(a, 0)$ and results in either no change or a transition to the state $((1, 0), \{0\})$ if the first agent changes its state to 1. Similarly, the last of the other agents could perform an a action, which is denoted by the $(a, 0, \text{down})$ transition and either results in no change or a transition to the state $((0, 0), \{1\})$. Finally, one of several other agents in state 0 could perform the a action. This is denoted by the $(a, 0, \text{up})$ transition and either results in no change or a transition to the state $((0, 0), \{0, 1\})$. The rest of the transitions are similarly defined.

We now proceed to observe, as we did for the concrete system, that the transition probabilities in an abstract system sum to 1.

Observation 2. *For any $\hat{g} \in \hat{S}_m$ and $\hat{a} \in \hat{Act}_m$, it is the case that $\sum_{\hat{g}' \in \hat{S}_m} \hat{t}_m(\hat{g}, \hat{a}, \hat{g}') = 1$.*

It follows from this observation that $\hat{\mathcal{S}}(m)$ is a well-defined MDP. We now give a few technical definitions that will be useful to prove our main result in Theorem 4.9.

Definition 4.2 (State abstraction). Let $n, m \in \mathbb{Z}^+$ with $n > m$. Then, the abstraction map on states $\lambda_{n,m} : S_n \to \hat{S}_m$ is given by:

$$\lambda_{n,m}(l_1, \ldots, l_n, l_E) \triangleq ((l_1, \ldots, l_m, l_E), \{l_{m+1}, \ldots, l_n\})$$

Intuitively, this maps concrete states of size $n > m$ into abstract states of size m by abstracting away how many agents are in each state for agents after the first m. We define a similar notion for actions.

Definition 4.3 (Action abstraction). Let $n, m \in \mathbb{Z}^+$ with $n > m$. Then, the abstraction map on actions $\lambda_{n,m} : S_n \times Act_n \to \hat{Act}_m$ is given by:

$$\lambda_{n,m}(g, a) \triangleq \begin{cases} a & \text{if } a \in GS \cup A_E \cup ((A \cup AE) \times Agt_m) \\ (a', g.i, \downarrow) & \text{if } a = (a', i) \in (A \cup AE) \times Agt_{m+1,n} \\ & \quad \text{and } |\{j \in Agt_{m+1,n} | g.i = g.j\}| = 1 \\ (a', g.i, \uparrow) & \text{if } a = (a', i) \in (A \cup AE) \times Agt_{m+1,n} \\ & \quad \text{and } |\{j \in Agt_{m+1,n} | g.i = g.j\}| > 1 \end{cases}$$

Intuitively, this maps global actions and actions of the first m agents to themselves. For actions of the remaining agents, it maps them to an abstracted action by labelling them with the state they occurred from and whether they were growing or shrinking actions. We now give a technical lemma that will be useful later.

Lemma 4.4. *Let $n, m \in \mathbb{Z}^+$ with $n > m$. Let $g \in S_n$ and $a \in P_n(g)$. Then, for any $\hat{g}' \in \hat{S}_m$:*

$$\hat{t}(\lambda_{n,m}(g), \lambda_{n,m}(g, a), \hat{g}') = \sum_{g' \in \lambda^{-1}_{n,m}(\hat{g}')} t_m(g, a, g')$$

Proof sketch. The proof can be obtained by checking all possible cases for action types. □

Intuitively, this lemma states that at each time-step, the probability of a transition in the abstract model is the sum of the probabilities of the concrete transitions that it encodes.

We now extend our notion of abstraction to paths.

Definition 4.5 (Path abstraction). Let $n, m \in \mathbb{Z}^+$ with $n > m$. Then, the abstraction map on paths $\lambda_{n,m} : IPath_{S(n)} \to IPath_{\hat{\mathcal{S}}(m)}$ is given by sending each infinite path $\rho = g_0 a_0 g_1 \ldots$ in $S(n)$ to the unique infinite path $\hat{\rho} = \hat{g}_0 \hat{a}_0 \hat{g}_1 \ldots$ in $\hat{\mathcal{S}}(m)$ such that, for all $i \in \mathbb{N}$: (i) $\hat{g}_i = \lambda_{n,m}(g_i)$; (ii) $\hat{a}_i = \lambda_{n,m}(g_i, a_i)$.

Intuitively, this simply uses the maps for states and actions that we have already defined to make a map on paths. We restrict this definition to finite paths in the obvious way. We now proceed to define a notion of an equivalent scheduler in the abstract model.

Definition 4.6 (Equivalent scheduler). Let $\sigma : FPath_{S(n)} \times Act_n \to [0, 1]$ be a scheduler in $\mathcal{S}(n)$. Then, for any $m < n$ we define the *equivalent scheduler* $\hat{\sigma} : FPath_{\hat{\mathcal{S}}(m)} \times \hat{Act}_m \to [0, 1]$ by:

$$\hat{\sigma}(\hat{\rho}, \hat{a}) \triangleq \frac{\sum_{\rho \in \lambda^{-1}_{n,m}(\hat{\rho})} \sum_{a \in Act_n : \lambda_{n,m}(last(\rho), a) = \hat{a}} \mathbf{P}_{\mathcal{S}(n)_\sigma}(\rho) \cdot \sigma(\rho, a)}{\sum_{\rho \in \lambda^{-1}_{n,m}(\hat{\rho})} \mathbf{P}_{\mathcal{S}(n)_\sigma}(\rho)}$$

Observe that after an abstract path $\hat{\rho}$ the probability of choosing an abstract action $\hat{a}$ is defined by summing over all possible concrete paths ρ that map to $\hat{\rho}$ and all concrete actions a that map to $\hat{a}$ the probability $\sigma(\rho, a)$, weighted by the probability $\mathbf{P}_{\mathcal{S}(n)_\sigma}(\rho)$ of the path occurring. We now show that this does indeed give a valid scheduler.

Lemma 4.7. *$\hat{\sigma} : FPath_{\hat{\mathcal{S}}(m)} \times \hat{Act}_m \to [0, 1]$ is a valid scheduler.*

Proof. The fact that $\hat{\sigma}(\hat{\rho}, \hat{a}) > 0$ only if $\hat{a} \in P(last(\hat{\rho}))$ follows from σ being a valid scheduler and the definition of the abstract

protocol. It remains to check that for any $\hat{\rho} \in FPath_{\hat{\mathcal{S}}(m)}$ we have that $\sum_{\hat{a} \in \hat{Act}_m} \hat{\sigma}(\hat{\rho}, \hat{a}) = 1$. For this notice that:

$$\sum_{\rho \in \lambda_{n,m}^{-1}(\hat{\rho})} \sum_{\hat{a} \in \hat{Act}_m} \sum_{a \in Act_n : \lambda_{n,m}(last(\rho), a) = \hat{a}} \mathbf{P}_{\mathcal{S}(n)_\sigma}(\rho) \cdot \sigma(\rho, a)$$
$$= \sum_{\rho \in \lambda_{n,m}^{-1}(\hat{\rho})} \sum_{a \in Act_n} \mathbf{P}_{\mathcal{S}(n)_\sigma}(\rho) \cdot \sigma(\rho, a) = \sum_{\rho \in \lambda_{n,m}^{-1}(\hat{\rho})} \mathbf{P}_{\mathcal{S}(n)_\sigma}(\rho)$$

with the first equality following simply from $\lambda_{n,m}$ being a function and the second from σ being a valid scheduler. So $\sum_{\hat{a} \in \hat{Act}_m} \hat{\sigma}(\hat{\rho}, \hat{a}) = 1$, as desired. □

Having proved the validity of our definition, we present a further lemma.

Lemma 4.8. *Let $\sigma : FPath_{S(n)} \times Act_n \to [0, 1]$ be a scheduler in $\mathcal{S}(n)$. Then it is the case that, for any path $\hat{\rho} \in FPath_{\hat{\mathcal{S}}(m)}$:*

$$\mathbf{P}_{\hat{\mathcal{S}}(m)_{\hat{\sigma}}}(\hat{\rho}) = \sum_{\rho \in \lambda_{n,m}^{-1}(\hat{\rho})} \mathbf{P}_{\mathcal{S}(n)_\sigma}(\rho)$$

Proof sketch. By induction on the length of the path. The base case is the unique length one path which always occurs with probability 1. The inductive case follows by unpacking the definitions, then using the inductive hypothesis and Lemma 4.4 to get the desired result.

□

This lemma is an extension of Lemma 4.4 stating that the probability of obtaining a certain path in the abstract model when following the equivalent scheduler is precisely the sum of probabilities of following an equivalent path in the concrete model. Notice that, since as discussed in Section 2 the probability of infinite paths occurring in a DTMC is uniquely defined by the probability of finite paths occurring, this result on finite paths automatically carries over to infinite paths.

We now give the main theorem of this paper, which will give us a partial decision procedure for the verification problem.

Theorem 4.9. *Suppose $\hat{\mathcal{S}}(m) \models P^{max}_{\leq x}[\psi]$ for some m-indexed formula ψ. Then, $\mathcal{S}(n) \models P^{max}_{\leq x}[\psi]$ for all $n \in \mathbb{Z}^+$ with $n > m$.*

Proof. We prove the contrapositive of this statement. Suppose we have $\mathcal{S}(n) \models P^{\max}_{>x}[\psi]$ for some $n > m$. Then, by definition, for some scheduler σ in $\mathcal{S}(n)$ we have:

$$\mathbf{P}_{\mathcal{S}(n)_\sigma}(\{\rho \in IPath_{\mathcal{S}(n)_\sigma} : \rho \models \psi\}) > x$$

Now, let $\hat{\sigma}$ be the equivalent scheduler in $\hat{\mathcal{S}}(m)$. Then, by Lemma 4.8:

$$\mathbf{P}_{\hat{\mathcal{S}}(m)_{\hat{\sigma}}}(\{\hat{\rho} \in IPath_{\hat{\mathcal{S}}(m)_{\hat{\sigma}}} : \hat{\rho} \models \psi\})$$
$$= \sum_{\hat{\rho} \in IPath_{\hat{\mathcal{S}}(m)_{\hat{\sigma}}} : \hat{\rho} \models \psi} \mathbf{P}_{\hat{\mathcal{S}}(m)_{\hat{\sigma}}}(\hat{\rho})$$
$$= \sum_{\hat{\rho} \in IPath_{\hat{\mathcal{S}}(m)_{\hat{\sigma}}} : \hat{\rho} \models \psi} \left(\sum_{\rho \in \lambda_{n,m}^{-1}(\hat{\rho})} \mathbf{P}_{\mathcal{S}(n)_\sigma}(\rho) \right) > x$$

with the final inequality following from the observation that every path in the concrete model has a corresponding path in the abstract model where, by definition, the same atomic propositions hold at each step and hence the same LTL formulas are satisfied. So, the result follows by taking $\hat{\sigma}$ as our scheduler. □

Notice this theorem gives us a partial decision procedure for checking properties of the form $P^{\max}_{\leq x}[\psi]$. In particular, to check an m-indexed formula we can construct the abstract model $\hat{\mathcal{S}}(m)$ and check whether it satisfies said formula. If this is the case, then we know the property will be true in all concrete systems of size $n > m$, so we only have to check the system of size m to ensure the formula holds in all systems.

Observe that, since the problem is undecidable, the procedure highlighted above is not complete: if the property does not hold in the abstract model, then no claims can be made as to whether it holds in all systems. This is a consequence of the fact that the abstract model is an over-approximation of all concrete systems capturing additional paths not possible in any concrete system. In particular, there are paths in the abstract model that could only be achieved in a system with an unbounded number of agents. This means that the present method cannot be used to verify protocols such as opinion formation [39], that are correct only in systems composed of finitely many agents.

5 IMPLEMENTATION AND EVALUATION

We implemented the method described in the previous section in a Java toolkit called PSV-CA (**P**robabilistic **S**warm **V**erifier by **C**ounter **A**bstraction), built on top of PRISM 4.0 [27], which provides the underlying probabilistic model checking procedures and part of the parsing. The source code for this and a number of models (including the one we evaluate below) are released as open-source, along with documentation describing the usage of the tool [34].

The modelling language we use is based on PRISM's modelling language for MDPs. Its key difference lies in the fact that precisely two MDPs must be defined in each file (one for the agents, and one for the environment) and each action must be declared of a specific type (asynchronous, agent-environment or global-synchronous) in order to define the communication pattern between these. Further details on the modelling language are omitted and can be found in the documentation within the software package [34].

Upon launch, the tool constructs (following Definition 3.6 and Definition 4.1) either an abstract or a concrete model for a desired number of agents depending on the options specified. Following this PSV-CA checks the PLTL specifications on the resulting model by calling PRISM.

In order to evaluate the approach, we modelled a swarm foraging protocol [6, 29]. We modelled the agents by considering four different possible states: resting in the nest, searching for food, reaching some food that they have found, and returning to the nest with food. When resting, a robot may decide with probability 0.5 to start looking for food. At each step, a robot searching for food may find it with probability 0.3. Then, it will move towards the food, which may be up to d time-steps away[1]. Lastly, it will return to the nest with the food and go back to its resting state. If the robot does not find food within t time-steps, it will return to the nest and go back to the resting state. The full model involves a combination

[1]This distance is chosen uniformly at random.

of asynchronous and agent-environment actions and can be found in [34].

We used this formalisation to investigate the probability that two units of food will be found and deposited within a certain number of time-steps. In particular, we checked for different values of p and k the 0-indexed property

$$P^{\max}_{\leq p}[F^{<k}(\text{deposited}_2, 0)],$$

where deposited_2 is an atomic proposition that holds when two units of food have been deposited in the nest (by any agent). This property expresses that, when the scheduler is behaving optimally to maximise the probability that two units of food are found and deposited within k time-steps, the probability of this event does not exceed p. For this experiment, we fixed the two parameters of the algorithm t and d to 3. Our results are recorded in Table 2.

Following the check on the abstract model, whenever the property holds on it, given Theorem 4.9, we are guaranteed that the specification holds on any concrete models of any size. For example, we can conclude that no matter how many robots are present, the probability of finding two units of food within 6 time-steps cannot exceed 0.5. However, in cases where the property does not hold, we cannot derive any conclusion on the validity of the specification on the concrete models. Note that incompleteness of the procedure is a necessary consequence of the general undecidability.

In order to assess the scalability of the method and implementation, we measured the time taken to construct the abstract model when running OpenJDK 1.8 (64-bit version, 8GB heap size) on a machine with Ubuntu 18.04 (Linux kernel 4.15.0-38) and an Intel i7-7700HQ processor. The results are recorded in Table 1, along with the number of states and transitions in the abstract model constructed. In a few cases the program timed out after 150 seconds; however the performance of PSV-CA demonstrates that the tool can successfully be used to verify small protocols in a reasonable time. Once constructed, the actual checking of a specification on the model is comparatively short, taking around 50ms. We do not carry out timing of the model checking algorithm as this is the one from PRISM and has already been extensively studied [18, 28].

We are unable to provide a full comparison of these results to other implementations as, to the best of our knowledge, there is no other tool that can be used to verify properties of unbounded probabilistic swarm systems as we do here. By comparing the present results to [24], which concern the non-probabilistic setting, we find that the addition of probabilities makes, as expected, the corresponding verification problem less tractable.

	d				
	1	2	3	4	5
1	0.1 sec 1,240 10,478	0.6 sec 7,599 89,700	1.5 sec 23,106 319,557	3.5 sec 52,588 817,747	8.3 sec 101,737 1,746,363
2	0.4 sec 4,793 57,184	2.4 sec 30,860 550,023	8.9 sec 92,183 1,991,641	24 sec 204,400 5,075,582	57 sec 386,856 10,749,914
3	1.2 sec 14,486 253,192	11 sec 94,209 2,477,958	48 sec 277,593 8,880,150	140 sec 604,125 22,251,457	*timeout*
4	3.2 sec 36,267 766,823	37 sec 234,620 7,598,501	*timeout*	*timeout*	*timeout*
5	8.4 sec 79,060 1,960,487	130 sec 506,092 19,340,746	*timeout*	*timeout*	*timeout*

Table 1: Time required to construct the abstract model (secs), its corresponding number of states and transitions. t is the maximum time a robot will unsuccessfully search before resting and d is the maximum distance from the robot food may appear.

	k				
p	6	8	10	12	14
0.25	False	False	False	False	False
0.50	True	False	False	False	False
0.75	True	True	False	False	False
0.90	True	True	True	False	False
0.95	True	True	True	True	False
0.99	True	True	True	True	True

Table 2: Result of checking $P^{\max}_{\leq p}[F^{<k}$ deposited$_2]$ with $t = 3$ as the maximum time a robot will unsuccessfully search before resting and $d = 3$ as the maximum distance from the robot food may appear.

6 CONCLUSIONS

As we argued in the introduction, at present no method exists for formally verifying arbitrarily large robotic swarms whose behaviour is described probabilistically. The lack of formal guarantees on the evolution of swarm systems hinders the applicability of the technology in safety-critical systems, in remote environments and beyond.

Methods based on parameterised model checking have successfully been introduced for the formal analysis of non-probabilistic swarms [20, 22, 24]. These have enabled the verification of deterministic variants of swarm protocols such as aggregation [21]. Some of these have been extended to probabilistic protocols, such as opinion formation [23]; but in these cases the probabilistic aspects have not been given prominence neither in the model nor in the specifications. More recently, [31] introduced an approach for reasoning about probabilistic properties of swarms. However, as discussed in the related work section, that approach has considerable limitations including limitations in the modelling and specification language. The present contribution takes a distinct approach in the modelling and uses a more expressive specification language.

In future work, we plan to apply our method to further protocols in swarm robotics. We will also investigate extending our procedure to richer specification languages including notions of knowledge and strategy [16]. Finally, we would like to combine our technique with work on fault-tolerance of swarms [25] in order to verify systems that are probabilistic and also need to be resistant to faults.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for valuable feedback and suggestions.

REFERENCES

[1] B. Aminof, S. Rubin, I. Stoilkovska, J. Widder, and F. Zuleger. 2018. Parameterized Model Checking of Synchronous Distributed Algorithms by Abstraction. In *Proceedings of the 19th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI18)*. Springer International Publishing, 1–24.

[2] K.R. Apt and D. C. Kozen. 1986. Limits for automatic verification of finite-state concurrent systems. *Inform. Process. Lett.* 22, 6 (1986), 307–309.

[3] C. Baier and J. P. Katoen. 2008. *Principles of Model Checking*. The MIT Press.

[4] R. Bloem, S. Jacobs, A. Khalimov, I. Konnov, S. Rubin, H. Veith, and J. Widder. 2015. *Decidability of Parameterized Verification*. Morgan and Claypool Publishers.

[5] E. Bonabeau, M. Dorigo, and G. Theraulaz. 1999. *Swarm intelligence*. Oxford University Press.

[6] A. Campo and M. Dorigo. 2007. Efficient multi-foraging in swarm robotics. In *Advances in Artificial Life*. Springer, 696–705.

[7] E.M. Clarke, O. Grumberg, and M.C. Browne. 1989. Reasoning about networks with many identical finite state processes. *Information and Computation* 81, 1 (1989), 13–31.

[8] M. de Oca, , E. Ferrante, A. Scheidler, Carlo C. Pinciroli, M. Birattari, and M. Dorigo. 2011. Majority-rule opinion dynamics with differential latency: a mechanism for self-organized collective decision-making. *Swarm Intelligence* 5, 3-4 (2011), 305–327.

[9] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker. 2011. Automated verification techniques for probabilistic systems. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems (Lecture Notes in Computer Science)*, Vol. 6659. Springer, 53–113.

[10] P. Fournier. 2015. *Parameterized verification of networks of many identical processes*. Ph.D. Dissertation. Université de Rennes 1.

[11] P. Gainer, C. Dixon, and U. Hustadt. 2016. Probabilistic Model Checking of Ant-Based Positionless Swarming. In *Proceedings of the 17th Annual Conference Towards Autonomous Robotics (TAROS16)*. Springer, 127–138.

[12] P. Gainer, E. M. Hahn, and S. Schewe. 2018. Incremental Verification of Parametric and Reconfigurable Markov Chains. In *Proceedings of the 15th International Conference on Quantitative Evaluation of Systems (QEST18) (Lecture Notes in Computer Science)*, Vol. 11024. Springer, 140–156.

[13] R. Giaquinta, R. Hoffmann, M. Ireland, A. Miller, and G. Norman. 2018. Strategy Synthesis for Autonomous Agents Using PRISM. In *Proceedings of the 10th NASA Formal Methods Symposium (NFM18) (Lecture Notes in Computer Science)*, Vol. 10811. Springer, 220–236.

[14] D. Graham, M. Calder, and A. Miller. 2009. An Inductive Technique for Parameterised Model Checking of Degenerative Distributed Randomised Protocols. *Electronic Notes in Theoretical Computer Science* 250, 1 (2009), 87–103.

[15] R. Hoffmann, M. Ireland, A. Miller, G. Norman, and S. M. Veres. 2016. Autonomous Agent Behaviour Modelled in PRISM - A Case Study. In *Proceedings of the 23rd International SPIN Symposium on Model Checking of Software (SPIN16) (Lecture Notes in Computer Science)*, Vol. 9641. Springer, 104–110.

[16] X. Huang and C. Luo. 2013. A logic of probabilistic knowledge and strategy. In *Proceedings of AAMAS2013*. IFAAMAS, 845–852.

[17] J. G. Kemeny, J. Laurie Snell, and A. W. Knapp. 1976. *Denumerable Markov Chains* (2 ed.). Springer-Verlag.

[18] J. Klein, C. Baier, P. Chrszon, M. Daum, C. Dubslaff, S. Klüppelholz, S. Märcker, and D. Müller. 2018. Advances in probabilistic model checking with PRISM: variable reordering, quantiles and weak deterministic Büchi automata. *International Journal on Software Tools for Technology Transfer* 20, 2 (2018), 179–194.

[19] S. Konur, C. Dixon, and M. Fisher. 2012. Analysing robot swarm behaviour via probabilistic model checking. *Robotics and Autonomous Systems* 60, 2 (2012), 199–213.

[20] P. Kouvaros and A. Lomuscio. 2013. A Cutoff Technique for the Verification of Parameterised Interpreted Systems with Parameterised Environments. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI13)*. AAAI Press, 2013–2019.

[21] P. Kouvaros and A. Lomuscio. 2015. A Counter Abstraction Technique for the Verification of Robot Swarms. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI15)*. AAAI Press, 2081–2088.

[22] P. Kouvaros and A. Lomuscio. 2015. Verifying Emergent Properties of Swarms. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI15)*. AAAI Press, 1083–1089.

[23] P. Kouvaros and A. Lomuscio. 2016. Formal Verification of Opinion Formation in Swarms. In *Proceedings of the 15th International Conference on Autonomous Agents and Multi-Agent systems (AAMAS16)*. IFAAMAS, 1200–1209.

[24] P. Kouvaros and A. Lomuscio. 2016. Parameterised Verification for Multi-Agent Systems. *Artificial Intelligence* 234 (2016), 152–189.

[25] P. Kouvaros, A. Lomuscio, and E. Pirovano. 2018. Symbolic Synthesis of Fault-Tolerance Ratios in Parameterised Multi-Agent Systems. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence and 23rd European Conference on Artificial Intelligence (IJCAI-ECAI18)*. IJCAI, 324–330.

[26] M. Kwiatkowska, G. Norman, and D. Parker. 2007. Stochastic Model Checking. In *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07) (Lecture Notes in Computer Science)*, Vol. 4486. Springer, 220–270.

[27] M. Kwiatkowska, G. Norman, and D. Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-time Systems. In *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV11)*. Springer, 585–591.

[28] M. Z. Kwiatkowska, G. Norman, and D. Parker. 2012. The PRISM Benchmark Suite. In *Proceedings of the 9th International Conference on Quantitative Evaluation of Systems (QEST12)*. IEEE Computer Society, 203–204.

[29] W. Liu and A. Winfield. 2010. Modelling and optimisation of adaptive foraging in swarm robotic systems. *The International Journal of Robotics Research* (2010).

[30] A. Lomuscio, W. Penczek, and H. Qu. 2010. Partial order reduction for model checking interleaved multi-agent systems. *Fundamenta Informaticae* 101, 1–2 (2010), 71–90.

[31] A. Lomuscio and E. Pirovano. 2018. Verifying Emergence of Bounded Time Properties in Probabilistic Swarm Systems. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence and 23rd European Conference on Artificial Intelligence (IJCAI-ECAI18)*. IJCAI, 403–409.

[32] R. R. Murphy. 2000. Marsupial and shape-shifting robots for urban search and rescue. *Intelligent Systems and their Applications* 15, 2 (2000), 14–19.

[33] A. Pnueli, J. Xu, and L. Zuck. 2002. Liveness with (0, 1,infinity)-counter abstraction. In *Proceedings of the 14th International Conference on Computer Aided Verification (CAV02) (Lecture Notes in Computer Science)*, Vol. 2404. Springer, 93–111.

[34] PSV-CA. 2019. Probabilistic Swarm Verifier by Counter Abstraction http://vas.doc.ic.ac.uk/software/psv-ca/. (2019).

[35] M.L. Puterman. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc.

[36] E. Şahin. 2005. Swarm robotics: From sources of inspiration to domains of application. In *Proceedings of the 2004 international conference on Swarm Robotics (SAB04)*. Lecture Notes in Computer Science, Vol. 3342. Springer, 10–20.

[37] E. Şahin and A. Winfield. 2008. Special issue on swarm robotics. *Swarm Intelligence* 2, 2 (2008), 69–72.

[38] W. M. Spears, D. F. Spears, R. Heil, W. Kerr, and S. Hettiarachchi. 2004. An Overview of Physicomimetics. In *Proceedings of the 1st International Workshop on Simulation of Adaptive Behavior (SAB04) (Lecture Notes in Computer Science)*, Vol. 3342. Springer, 84–97.

[39] G. Valentini, H. Hamann, and M. Dorigo. 2015. Efficient decision-making in a self-organizing robot swarm: On the speed versus accuracy trade-off. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems*. IFAAMAS Press, 1305–1314.

[40] A. Winfield, W. Liu, J. Nembrini, and A. Martinoli. 2008. Modelling a wireless connected swarm of mobile robots. *Swarm Intelligence* 2, 2-4 (2008), 241–266.