A Counter Abstraction Technique for Verifying Properties of Probabilistic Swarm Systems

Alessio Lomuscio and Edoardo Pirovano

Department of Computing
Imperial College London
London, UK

Abstract

We introduce a semantics for reasoning about probabilistic multi-agent systems in which the number of participants is not known at design-time. We define the parameterised model checking problem against PLTL specifications for this semantics, and observe that this is undecidable in general. Nonetheless, we develop a partial decision procedure for it based on counter abstraction. We prove the correctness of this procedure, and present an implementation of it. We then go on to use our implementation to verify a number of example scenarios from swarm robotics and other settings.

Keywords: Probabilistic model checking; Parameterised verification; Swarm robotics; Multi-agent systems

1. Introduction

Robotic swarms [1, 2] have been put forward as a method for addressing a number of applications including environmental monitoring [3], search-and-rescue [4], and surveillance [5]. Compared to single-robot systems, robotic swarms may provide a number of advantages, including robustness (tolerance of the system to faults) and scalability (the ability to deploy a different number of robots depending on the scale of the problem being tackled). Many target applications, such as autonomous monitoring of the condition of pipelines [6], benefit from the scalability and robustness of the swarm.

Before deploying a robotic swarm in a safety-critical context, it is desirable to obtain formal guarantees of its behaviour. These may include properties such as safety (the system will never reach some undesirable state) or correctness (some goal is always achieved). A number of solutions have been put forward to verifying swarms, including process calculi [7] and mathematical modelling using rate equations [8]. These approaches to obtaining safety guarantees typically require significant input from an expert.

Another approach to verifying desired properties in a multi-agent system is through model checking [9], which aims to use a more automated approach to reduce the necessary expert input. In the model checking approach, the system is described in terms of the states it can be in and how it transitions between them. In addition, a specification of the desired properties is provided in a logical language. An automated toolkit is used to verify that the possible runs of the system satisfy the desired specification. However, traditional model checking techniques can only verify systems of a fixed size [10, 11]. Since one of the main advantages of robotic swarms is their scalability, checking a property only on a system with a fixed number of participants is typically not sufficient.

Parameterised model checking [12] is an extension of model checking that enables the verification of systems of an arbitrary size. Despite the parameterised model checking problem being undecidable in its most general formulation [13], decidable fragments exist. In particular, one avenue to decidability is restricting either the communication pattern between agents, or the logic in which specifications are expressed [14, 15]. Parameterised model checking has also been explored in the context of robotic swarms [16], where it has enabled the verification of a number of protocols such as the alpha algorithm [17]. However, many protocols followed by robotic swarms are probabilistic in nature and parameterised model checking has mainly so far focused on non-stochastic systems (with a few exception that we discuss in our survey of related work below).

Probabilistic model checking of fixed size systems has been extensively studied [18] and a number of toolkits have been developed [19, 20] to enable the verification of practical systems in a variety of different applications [21, 22]. Extensions of probabilistic model checking have also been developed to study a number of more complex scenarios including stochastic games [23] and timed systems [24]. These techniques have been applied to analyse a number of different protocols from swarm robotics [10, 11, 25], but always with fixed number of participants. We aim to overcome this limitation. We will extend the parameterised verification techniques discussed earlier to the probabilistic case in order to provide methods and tools for the formal verification of agent-based systems that are both stochastic and possibly unbounded in size.

Contribution. In this work we introduce a formal semantics for reasoning about unbounded probabilistic swarms, i.e., multi-agent systems in which the number of agents is not known at design-time. We study the parameterised model checking problem for this semantics against PLTL specifications. While the problem is undecidable in general, we develop a novel method based on counter abstraction [26] to provide an over-approximation of the possible executions of the system. Using this over-approximation, we develop a partial decision procedure for the verification problem. We formally prove the correctness of this procedure and report its implementation in a novel toolkit built on top of PRISM [19]. We use this implementation to study a number of scenarios, including autonomous robots moving along a track [27], a swarm of robots foraging for food [28], and a channel jamming scenario [29].

1.1. Related Work

The field of parameterised model checking [12, 30] is concerned with studying the verification of systems that contain an arbitrary number of components. While in the most general formulation the parameterised problem is undecidable [13], decidable fragments are known to exist. Some of these fragments are identified in the context of token-passing systems [15]. Here, communication happens by passing a token around a network, which is arranged in configurations such as rings, cliques, and stars. In other work, network invariants [31] and network decomposition [32] have resulted in partial verification procedures. Further, restrictions on the communication pattern between agents, such as considering guarded protocols [14], have also been used to lead to decidability of the verification problem.

Many approaches to parameterised model checking rely on computing cutoffs [33]. A cutoff provides an indication of the number of agents that is sufficient to consider in order to check whether a specification holds on the arbitrary system. A cutoff may either be *static* if it does not depend on the property in question, or *dynamic* if it does [34].

Static cutoffs have been explored in a line of work addressing parameterised swarm systems [16, 17, 35]. The semantics presented there are closely related to ours: they are also asynchronous and use the same types of synchronisation between agents as we do. The parameterised model checking toolkit MCMAS-P, relying on MCMAS [36] for finite model verification, was put forward in that work and later extended to support richer scenarios such as systems that may exhibit faults [37, 38]. However, this work mostly does not address probabilities as we do here. Indeed, to the best of our knowledge, no results exist at present concerning cutoffs for stochastic systems. One parametised model checking work in which probability was considered to some extent is [39]. However, this was done in an ad-hoc manner to model the specific case of opinion formation [40], and is less general than the techniques presented here.

Another formalism that has been widely considered is that of Petri nets [41] or equivalently vector addition systems [42]. This work has included studies of reachability properties [43] and of techniques to identify dynamic cutoffs [34, 44]. Extensions have also been developed that incorporate continuous time [45, 46, 47]. We note that encoding a system into a Petri net requires requires significant input from an expert, whereas the model checking approach we present here aims to verify systems in a more automated manner.

A further formalism that has been used to study multi-agent systems is that of population protocols [48]. In this semantics, agents interact in a pair-wise manner and update their state according to the state of the agent they are interacting with. A protocol is well-defined, and said to compute a predicate P if all fair sequences of interactions from an initial configuration C eventually result in the agents agreeing on a value $P(C)$. It has been proved [49] that the predicates computable by population protocols are exactly those definable in Presburger arithmetic [50]. Further, it has also been shown [51] that the problem of checking whether a protocol is well-defined and computes a given predicate is decidable by reduction to reachability in Petri nets. As with Petri nets, significant input from an expert human is needed to encode a system into a population protocol. This limits the applicability of the technique.

There are also a variety of other methods that have been proposed for verifying multi-agents that are less closely related to the verification approaches considered here, including techniques such as process calculi [7] and rate equations [8]. As with the other techniques described, some expert input is required to verify these systems. Futher, the properties considered with these techniques are typically of the whole population and properties of inidividual agents cannot be checked as we do here.

There is also a significant body of work on the model checking of probabilistic systems [9]. This work is very wide-ranging and has resulted in the development of techniques working on a number of different models including DTMCs [52], MDPs [53], CTMCs [54] and PTAs [24]. Practical toolkits for probabilistic verification such as PRISM [19] and Storm [20] have been developed and successfully applied in applications ranging from systems biology [21] to network protocols [22]. Probabilistic model checking has also been applied to verifying multi-agent algorithms such as foraging [55, 11], team formation [56], aggregation [10] and search-and-rescue [25]. Some of this work has also included other aspects such as strategy synthesis [57]. However, we note that this work does not address the unbounded nature of swarm systems as it relies on fixing a number of participants in the swarm with the verification of larger swarms quickly becoming intractable.

The parameterised verification of probabilistic systems is a relatively unexplored area. In [58], the authours present a technique for verifying probabilistic network protocols that is based on identifying a sub-class of systems (named *degenerative*) in which larger networks eventually behave like smaller ones. This is similar to the non-probabilistic notion of cutoffs discussed earlier. The approach in this paper tackles a similar problem but instead of studying a sub-class of systems, we here aim to give a more general verification technique for a broader ranger of systems.

In [59], a parameterised semantics is introduced that includes both probabilities and continuous time. Reachability properties are studied on both systems where the number of participants is fixed and ones where participants may enter and leave at run-time. Some of the resulting decision problems are found to be decidable whilst others are shown to be undecidable. The models considered there are distinct from ours. In particular, we do not have a continuous notion of time and the communication pattern between agents in our systems is different from the broadcasting and receiving pattern used there which is more suited to network protocols than the AI applications we consider here. Further, the PLTL properties we consider here are more expressive than the qualitative reachability ones considered there.

The only variable that may grow unboundedly in these studies is the number of participants in the system, as is indeed the case in this paper. However, other forms of parameterised systems have also been considered. For example, in systems with parameterised environments, aspects of the environment can grow arbitrarily [60]. Similarly to the works above, this also raises issues with decidability of the resulting verification problem. Further, in these lines as well as in this paper, the number of participants may be arbitrarily large, but it does not change during the system's execution. Other approaches, however, have considered *open*, or *dynamic*, systems in which the participants may join and leave the system at run-time [61, 62, 63]. However, the setup and the objectives of these contributions are quite different from those in this paper.

Previous work by the authors. The work here presented builds upon previous papers by the same authors [64, 65, 66, 67].

In [64] a semantics for reasoning about probabilistic swarm systems in which actions are performed in synchronous rounds was presented. A logic on finite traces and a verification procedure for it were introduced. The work here proposed is richer in expressiveness since this semantics is asynchronous and the specifications are not limited to finite traces.

More similarly to the work presented here, in [65] the verification of asynchronous models against a logic on infinite traces is studied. However, the present work extends [65] in several ways. Firstly, the semantics here address inhomogenous systems in which agents can have different behaviours. This is necessary to verify the scenarios presented in Section 5. Secondly, while [65] provides no proofs and detailed technical constructions, we provide both here. Thirdly, in [65] only one form of PLTL properties is considered, while here we give a procedure for the full logic. Finally, the implementation here discussed is an extension of that from [65] to account for different agent behaviours.

Different from the work here presented are the results in [66, 67], which relate to strategic properties of the system and run-time faults. These explorations are orthogonal to what we consider here.

1.2. Outline of the paper

The rest of the paper is organised as follows. In Section 2 we present the background on probabilistic model checking, and introduce the notation that will be used throughout the paper. In Section 3 we define the parameterised semantics for reasoning about unbounded probabilistic swarms. We show this is associated to an undecidable verification problem. Nonetheless, 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 it against a number of example scenarios. We conclude in Section 6.

2. Background

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

2.1. Discrete Time Markov Chains

We briefly summarise *discrete time Markov chains* (DTMCs). For more background on DTMCs and the model checking of them see [52, 9, 68].

Definition 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 any path $\rho = s_0 s_1 s_2 \ldots$ and $i \in \mathbb{N}$ we use $\rho_i \triangleq s_i$ to denote the ith state in ρ and and $\rho(i) \triangleq s_i s_{i+1} s_{i+2} \ldots$ to denote the suffix of ρ starting from the ith position. 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 [68], this can be extended to a probability space on 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 a set of infinite paths occurring is equal to the probability of their finite prefixes occurring.

2.2. Markov Decision Processes

We now summarise some key aspects of *Markov decision processes* (MDPs). We refer to [9, 69, 53] for more details. We mostly follow the notation used in [53].

Definition 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 $s_0 a_0 s_1 a_1 s_2 \ldots$ 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 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 $Sch_{\mathcal{M}}$ the set of all schedulers for $\mathcal{M}$. Various classes of schedulers may be defined [53]. Note that when maximising or minimising the probability of reaching a target set of states, it is sufficient to consider schedulers that are memoryless, i.e.,. functions that only depend on the final state of the path, and deterministic, i.e., functions that 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 4 (Induced DTMC)**.** *Given an MDP $\mathcal{M} = \langle S, \iota, A, P, t, L\rangle$ and a scheduler $\sigma : FPath_{\mathcal{M}} \times A \to [0, 1]$, 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:*

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

- *$L'(\rho) \triangleq L(last(\rho))$ 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 [53].

3. Probabilistic Swarm Systems

In this Section we introduce the syntax and semantics of probabilistic swarm systems and define their model checking problem. In what follows a swarm system is characterised by finitely many types of agents, each giving rise to an arbitrary number of agents.

3.1. Models

We begin by defining probabilistic parameterised interleaved interpreted systems (PPIIS). These extend the framework of parameterised interpreted systems (PIIS) presented in [16] with probabilistic behaviour. In turn, PIIS extend interleaved interpreted systems [70].

A PPIIS is composed of a finite number of *agent templates*, which describe the different possible behaviours of individual agents and an *environment* which captures the behaviour of the other parts of the system. Note that while there is a finite number of agent templates describing the different behaviours of agents, there may be an arbitrarily large number of agents instantiated from each template.

Definition 5 (Probabilistic agent template)**.** *A probabilistic agent template is a tuple $T_i = \langle S_i, \iota_i, Act_i, P_i, t_i\rangle$ where:*

- *The set S_i is a finite set of agent local states.*

- *$\iota_i \in S_i$ is a distinguished initial state.*

- *$Act_i = A_i \cup AE_i \cup GS$ 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 9.*
- *The agent's protocol function $P_i : S_i \rightarrow \mathcal{P}(Act_i)$ defines which actions are enabled at a given state.*
- *The agent's transition function $t_i : S_i \times Act_i \times S_i \rightarrow [0, 1]$ 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 $l \in S_i$ and $a \in P_i(l)$ we have $\sum_{l' \in S_i} t_i(l, a, l') = 1$.*

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

One particular class of agent templates that is worthwhile to distinguish is those that transition deterministically when performing a global-synchronous action. We formalise this below.

Definition 6 (Deterministic-synchronous agent templates)**.** *A probabilistic agent template $T_i = \langle S_i, \iota_i, Act_i, P_i, t_i \rangle$ is said to be deterministic-synchronous if it is the case that whenever $a \in GS$ then $t_i(s, a, s') \in \{0, 1\}$ for all $s, s' \in S_i$.*

For simplicity of presentation, we will assume that all agent templates are deterministic-synchronous. However, note that an agent template that is not deterministic-synchronous can be transformed into one that is by adding an asynchronous action to resolve the probabilistic choice after the global-synchronous action.

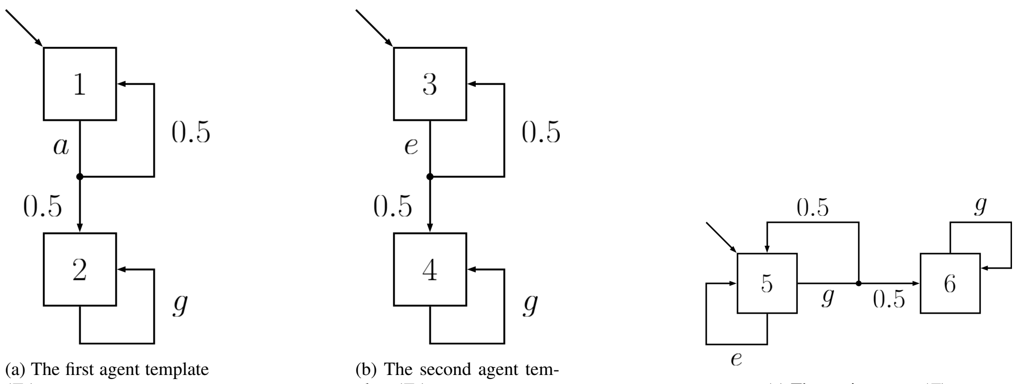
We now proceed to define the environment that the agents operate in.

Definition 7 (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 $Act_E = A_E \cup \bigcup_{i=1}^{k} AE_i \cup GS$.*
- *The environment protocol P_E is a function $P_E : S_E \rightarrow \mathcal{P}(Act_E)$ giving the permissible actions in each state.*
- *The transition function $t_E : S_E \times Act_E \times S_E \rightarrow [0, 1]$ which is such that for every $l \in S_E$ and $a \in P_E(l)$ we have $\sum_{l' \in S_E} t(l, a, l') = 1$.*

Once again, the environment is similar to an MDP but has been extended to encode different action types. We can now define PPIIS, which will be composed of a finite number of agent templates together with an environment and labelling functions which capture what atomic propositions hold at different points of the system's evolution.

Definition 8 (PPIIS)**.** *A probabilistic parameterised interleaved interpreted system (PPIIS) is a tuple $\mathcal{S} = \langle \mathcal{T}, E, \mathcal{V}, V_E \rangle$, where $\mathcal{T} = \{T_1, \ldots, T_k\}$ is a non-empty and finite set of probabilistic agent templates, E is an environment and $\mathcal{V} = \{V_1, \ldots, V_k\}$ is a set of valuation functions $V_i : S_i \times S_E \rightarrow \mathcal{P}(AP)$, one for each agent template, to a set of atomic propositions AP. Finally, $V_E : S_E \rightarrow \mathcal{P}(AP)$ is a valuation function for the environment to the atomic propositions.*



(a) The first agent template (T_1).

(b) The second agent template (T_2).

(c) The environment (E).

Figure 1: An example PPIIS $\mathcal{S} = \langle\{T_1, T_2\}, E, \{V_1, V_2\}, V_E\rangle$ with two agent templates. The a action is asynchronous, the e action is an agent-environment one, while the g action is global-synchronous.

Notice that we assume that the sets S_i of local states of each agent template T_i are disjoint, and likewise the sets of actions Act_i are disjoint, with the exception of the global actions GS which are the same for all agents and the environment.

Further, notice that while each agent template has its own valuation function V_i in order to allow us to express local properties of that agent, the valuation functions can all also use the local state of the environment in order to allow us to consider this when deciding what atomic propositions hold. Further, the V_E valuation function allows us to express global properties that are only concerned with the state of the environment.

An example PPIIS with two agent templates can be seen in Figure 1. The agent template T_1 has two states. In the initial state, it can perform the asynchronous a action which with equal probability takes it back to the same state, or takes it to its other state. Once in its second state, the agent can perform the global-synchronous action g. The agent template T_2 is similar except it uses an agent-environment action e instead of an asynchronous one a. Finally, the environment has two states. In its initial state, it can perform the e action or the g one. On performing the g action it can, with probability 0.5, transition to its second state. Once in the second state, it can only perform the g action and remain there.

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 $\bar{n} \in \mathbb{Z}^k$, the system $\mathcal{S}(\bar{n})$ of $\bar{n}$ agents is the result of the composition of $\bar{n}_i$ copies of T_i for each $1 \leq i \leq k$ and one environment. Each agent has an identity $(i, j) \in \mathbb{N} \times \mathbb{N}$ giving its type i (the agent template it was instantiated from), and a unique number j distinguishing it from the other agents of type i. We will denote by $\mathcal{A}(\bar{n})$ the set of all agents, i.e.

$$\mathcal{A}(\bar{n}) \triangleq \{(i, j) \in \mathbb{N} \times \mathbb{N} : 1 \leq i \leq k, 1 \leq j \leq \bar{n}_i\}$$

For a system of size $\bar{n} \in \mathbb{Z}^k$, a *global state* $g = \langle \bar{s}_1, \ldots, \bar{s}_k, s_e\rangle$ is a $(k+1)$-tuple. The vector $\bar{s}_i \in (S_i)^{\bar{n}_i}$ gives the local state of the $\bar{n}_i$ agents of type i. The state $s_e \in S_E$ gives the local state of the environment. Thus, the global state encodes all the information to describe the system at a particular instant of time. Let $S_{\bar{n}}$ denote the set of all such global states. For a global state $g = \langle \bar{s}_1, \ldots, \bar{s}_k, s_e\rangle \in S_{\bar{n}}$ we write $g.(i, j)$ to denote the local state of the j-th agent of type i in g, i.e., $(\bar{s}_i)_j$ and $g.E$ to denote the state of the environment in g, i.e. s_e.

Actions in the global system may be one of four types as follows (this will be formalised in Definitions 9 and 10).

1. *Global-synchronous actions* are performed by all agents and the environment together.
2. *Asynchronous environment actions* are performed by the environment on its own.
3. *Asynchronous agent actions* are performed by one agent on its own, and in the global system are labelled with which agent performed the action.
4. *Agent-environment agent actions* are performed by one agent together with the environment, and in the global system are labelled with which agent performed the action.

We denote by $Act_{\bar{n}}$ the set of all such global actions, i.e.

$$Act_{\bar{n}} \triangleq GS \cup A_E \cup \bigcup_{(i,j)\in\mathcal{A}(\bar{n})} ((A_i \cup AE_i) \times \{(i,j)\})$$

We now go on to define the global protocol, which determines which actions are available at each state in the global system.

Definition 9 (Global protocol)**.** *The global protocol $P_{\bar{n}} : S_{\bar{n}} \to \mathcal{P}(Act_{\bar{n}})$ is defined by $a \in P_{\bar{n}}(g)$ if and only if*

- (Global-synchronous). *(i) $a \in GS$; (ii) for all $(i,j) \in \mathcal{A}(\bar{n})$, $a \in P_i(g.(i,j))$; (iii) $a \in P_E(g.E)$.*
- (Asynchronous environment). *(i) $a \in A_E$; (ii) $a \in P_E(g.E)$.*
- (Asynchronous agent). *(i) $a = (a',(i,j)) \in A_i \times \mathcal{A}(\bar{n})$; (ii) $a' \in P_i(g.(i,j))$.*
- (Agent-environment). *(i) $a = (a',(i,j)) \in AE_i \times \mathcal{A}(\bar{n})$; (ii) $a' \in P_i(g.(i,j))$; (iii) $a' \in P_E(g.E)$.*

Thus, a global-synchronous action must be enabled for all agents and the environment. An asynchronous environment action must be enabled for the environment. An asynchronous agent action must be enabled for the agent performing it and, finally, an agent-environment action must be enabled for both the agent performing it and the environment.

The next definition gives the global transition function, which defines the probability of transitioning between two states of the global system.

Definition 10 (Global transition function)**.** *The global transition function $t_{\bar{n}} : S_{\bar{n}} \times Act_{\bar{n}} \times S_{\bar{n}} \to [0,1]$ is defined by*

$$t_{\bar{n}}(g,a,g') \triangleq \begin{cases} t_E(g.E,a,g'.E)\cdot\prod_{i=1}^{k}\prod_{j=1}^{\bar{n}_i} t_i(g.(i,j),a,g'.(i,j)) & \text{if } a \in GS \\ t_E(g.E,a,g'.E) & \text{if } a \in A_E \text{ and} \\ & \quad \forall (i,j)\in\mathcal{A}(\bar{n}) : g.(i,j) = g'.(i,j) \\ t_i(g.(i,j),a',g'.(i,j)) & \text{if } a = (a',(i,j)) \in A \times \mathcal{A}(\bar{n}) \text{ and } g.E = g'.E \text{ and} \\ & \quad \forall (i',j') \in \mathcal{A}(\bar{n}) \setminus \{(i,j)\} : g.(i',j') = g'.(i',j') \\ t_E(g.E,a',g'.E)\cdot t_i(g.(i,j),a',g'.(i,j)) & \text{if } a = (a',(i,j)) \in AE \times \mathcal{A}(\bar{n}) \text{ and} \\ & \quad \forall (i',j') \in \mathcal{A}(\bar{n}) \setminus \{(i,j)\} : g.(i',j') = g'.(i',j') \\ 0 & \text{otherwise} \end{cases}$$

Thus, the probability of a global transition is always given by multiplying the probability of the agent(s) involved in the action (and, if applicable, the environment) performing their local transition, whilst forcing all those not involved in the action to remain in the same state.

Having given this definition, we now prove that this defines a valid probability distribution.

Lemma 1. *For any $g \in S_{\bar{n}}$ and $a \in Act_{\bar{n}}$, it is the case that:*

$$\sum_{g' \in S_{\bar{n}}} t_{\bar{n}}(g, a, g') = 1$$

Proof. We check this for each different type of action a:

- $a \in GS$. Then:

$$\begin{aligned}
\sum_{g' \in S_{\bar{n}}} t_{\bar{n}}(g, a, g') &= \sum_{s_E \in S_E} \sum_{g' \in S_{\bar{n}} : g'.E = s_E} \left(t_E(g.E, a, s_E) \cdot \prod_{i=1}^{k} \prod_{j=1}^{\bar{n}_i} t_i(g.(i,j), a, g'.(i,j)) \right) \\
&= \sum_{s_E \in S_E} t_E(g.E, a, s_E) \cdot \sum_{g' \in S_{\bar{n}} : g'.E = s_E} \left(\prod_{i=1}^{k} \prod_{j=1}^{\bar{n}_i} t_i(g.(i,j), a, g'.(i,j)) \right) \\
&= \sum_{s_E \in S_E} t_E(g.E, a, s_E) \cdot \prod_{i=1}^{k} \prod_{j=1}^{\bar{n}_i} \left(\sum_{s' \in S} t_i(g.(i,j), a, s') \right) = 1
\end{aligned}$$

  with the first equality following by splitting the sum on its change to the state of the environment and the state of the agents, and the second equality following by splitting the sum on the states of the individual agents.

- $a \in A_E$. Then:

$$\sum_{g' \in S_{\bar{n}}} t_{\bar{n}}(g, a, g') = \sum_{s_E \in S_E} t_E(g.E, a, s_E) = 1$$

  with the first equality following from all g' with $g.(i,j) \neq g'.(i,j)$ for some $(i,j) \in \mathcal{A}(\bar{n})$ having $t_{\bar{n}}(g, a, g') = 0$.

- $a = (a', (i,j)) \in A \times \mathcal{A}(\bar{n})$. Then:

$$\sum_{g' \in S_{\bar{n}}} t_{\bar{n}}(g, a', g') = \sum_{s \in S} t_i(g.(i,j), a, s) = 1$$

  with the first equality following again from all other terms of the sum being 0.

- $a = (a', (i,j)) \in AE \times \mathcal{A}(\bar{n})$. Then:

$$\begin{aligned}
\sum_{g' \in S_{\bar{n}}} t_{\bar{n}}(g, a', g') &= \sum_{s_E \in S_E} \sum_{s \in S} t_E(g.E, a', s_E) t_i(g.(i,j), a', s) \\
&= \sum_{s_E \in S_E} t_E(g.E, a', s_E) \left(\sum_{s \in S} t_i(g.(i,j), a', s) \right) = 1
\end{aligned}$$

  with the first equality following by splitting the terms of the sum up between those that change the state of the environment and those that change the state of agent (i, j) and noting that those that change any other component of the state will be 0.

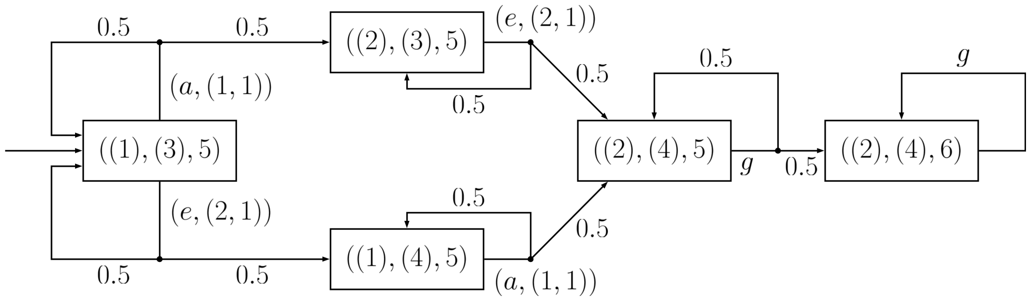


Figure 2: An example concrete system $\mathcal{S}((1, 1))$ where $\mathcal{S}$ is the PPIIS from Figure 1.

This covers all cases for the action a, and thus completes our proof. □

Having defined all the necessary components and shown the validity of our definitions, we combine these to define concrete systems.

Definition 11 (Concrete system). *Given a PPIIS $\mathcal{S}$ with k agent templates and $\bar{n} \in \mathbb{Z}^k$, the concrete system of $\bar{n}$ agents is defined by $\mathcal{S}(\bar{n}) = \langle S_{\bar{n}}, \iota_{\bar{n}}, Act_{\bar{n}}, P_{\bar{n}}, t_{\bar{n}}, V_{\bar{n}} \rangle$, where $\iota_{\bar{n}} = \langle (\iota_1, \ldots, \iota_1), \ldots, (\iota_k, \ldots, \iota_k), \iota_E \rangle$, the labelling function $V_{\bar{n}} : S_{\bar{n}} \to \mathcal{P}((AP \times \mathcal{A}(\bar{n})) \cup AP)$ is defined by*

$$V_{\bar{n}}(g) \triangleq \{(p, (i, j)) \in AP \times \mathcal{A}(\bar{n}) : p \in V_i(g.(i, j), g.E)\} \cup V_E(g.E)$$

and the other components are defined as in Definitions 9 and 10.

Notice our labelling function creates a new atomic proposition $(p, (i, j))$ for each atomic proposition p and agent (i, j) that holds precisely in the global states where p holds for the local state of agent (i, j). Atomic propositions p that hold for the environment are added to the set without such a label.

An example of such a concrete system with two agents (one of each type) can be seen in Figure 2. In the initial state, either the agent of type 1 can perform the a action or the agent of type 2 can perform the e action. If an agent changes state, then it can no longer perform any action (since the only available action is the g one, but the other agent cannot yet perform it). So, the other agent now performs its action until it changes state, at which point the g action becomes possible. This action may change the state of the environment with probability 0.5, at which point the system reaches state $((2), (4), 6)$ and no further state changes occur (though the g action can continue being performed).

Notice that the concrete system is simply an MDP, so we can express properties of it in any logics that are used in probabilistic model checking of MDPs. We now define such a logic.

Definition 12 (Probabilistic LTL). *For $a \in AP$ and $i, j \in \mathbb{N}$, the probabilistic LTL logic (PLTL) is the set of formulas ϕ defined by the following BNF:*

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

We say that a PLTL formula is $\bar{m}$-indexed, where $\bar{m} \in \mathbb{N}^k$, if $\bar{m}$ is the least tuple such that for any atomic proposition $(a, (i, j))$ that appears in the formula it is the case that $1 \leq i \leq k$ and

$1 \leq j \leq \bar{m}_i$. Intuitively, this means that satisfaction of the formula is based only on the states of the first $\bar{m}_i$ agents for each type of agent i. Notice that since formulas are finite, every formula has a well-defined index.

The PLTL formula $P^{\text{max/min}}_{\bowtie x}[\psi]$ is read as "with a scheduler that tries to maximise/minimise the probability of ψ occurring, this probability is $\bowtie x$". The path formula $X\psi$ is read as "after one time-step ψ holds", whilst $\psi_1 \; U \; \psi_2$ is read as "at some point ψ_2 will hold, and before then ψ_1 must hold."

We will also use standard LTL abbreviations such as $F\psi \equiv \top U\psi$ to mean "at some point ψ holds" and $G\psi \equiv \neg F\neg\psi$ to mean "ψ always holds".

We now formalise the notion of satisfaction for PLTL specifications on swarm systems.

Definition 13 (Satisfaction)**.** *Given a concrete swarm system $\mathcal{S}(\bar{n})$ and ϕ an $\bar{m}$-indexed PLTL formula, where $\bar{n}_i \geq \bar{m}_i$ for all i, the satisfaction of ϕ on $\mathcal{S}(\bar{n})$ is inductively defined as follows:*

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

Satisfaction for path formulae is defined as usual, i.e.,

$\omega \models \top$		*always holds*
$\omega \models (a, (i, j))$	*iff*	$(a, (i, j)) \in V_{\bar{n}}(\omega_0)$
$\omega \models a$	*iff*	$a \in V_{\bar{n}}(\omega_0)$
$\omega \models \neg\psi$	*iff*	$\omega \not\models \psi$
$\omega \models \psi_1 \wedge \psi_2$	*iff*	$\omega \models \psi_1$ *and* $g \models \psi_2$
$\omega \models X\psi$	*iff*	$\omega(1) \models \psi$
$\omega \models \psi_1 U \psi_2$	*iff*	*for some* $i \geq 0$, $\omega(i) \models \psi_2$ *and for all* $0 \leq j < i$, $\omega(j) \models \psi_1$

For instance, consider a collective transport scenario [71] where a group of robots (all of the same type) aims to collaborate to move an object to a destination. In this scenario we might want to consider specifications such as:

$$P^{min}_{>0.7}[F\texttt{objectAtDestination}]$$

which says that even with a scheduler that tries to minimise the chances of this happening, there is still a probability of over 0.7 that the object will reach its destination. The index of this formula is (0) since it refers to no agents.

For another example, consider an opinion formation scenario [40] where a group of agents (of two different types) must agree on a choice from a set of actions. In this scenario it might be desirable to consider a property such as:

$$P^{max}_{\leq 0.1}[G(\neg(\texttt{decisionReached}, (2, 1))]$$

which says that even with a scheduler that tries to maximise the chances of this happening, the probability of agent $(2, 1)$ never reaching a decision is at most 0.1. The index of this formula is $(0, 1)$ since it refers to no agents of the first type, and the first agent of the second type.

In this Section we introduced PPIIS, a semantics for reasoning about probabilistic swarm systems, as well as the logic PLTL to express specifications for them. In what follows we will address the automatic verification of PLTL specifications on PPIIS.

4. Model Checking PPIIS

In this Section we introduce the parameterised model checking problem for PPIIS, which consists of determining whether a PLTL formula holds in concrete systems of any size built from a template. We formalise this below.

Definition 14 (Parameterised Model Checking)**.** *Given a PPIIS $\mathcal{S}$ and an $\bar{m}$-indexed formula ϕ, the parameterised model checking problem involves establishing whether it is the case that $\mathcal{S}(\bar{n}) \models \phi$ for any $\bar{n}$ with $\bar{n}_i \geq \bar{m}_i$ for all i. 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 [13], it is also undecidable in general. In the following we will explore a partial decision procedure.

The partial decision procedure to be introduced is based on counter abstraction [26]. Intuitively, we will build a finite abstract model of the system capturing an over-approximation of the infinitely many paths that can be executed by the infinitely many concrete systems of increasing size. To do so, we will discard information about precisely how many agents are in each state at each step and instead only record which states had one or more agents in them.

This will mean that for every path in a concrete system (of any size) there will be a corresponding path in our abstract model. We will exploit this property to give a partial decision procedure for the parameterised model checking problem.

We now proceed to define our abstract system based on counter abstraction.

Definition 15 (Abstract system)**.** *Given a PPIIS $\mathcal{S}$ and an $\bar{m} \in \mathbb{Z}^k$, the abstract system of $\bar{m}$ agents is defined by $\hat{\mathcal{S}}(\bar{m}) = \langle \hat{S}_{\bar{m}}, \hat{\iota}_{\bar{m}}, \hat{Act}_{\bar{m}}, \hat{P}_{\bar{m}}, \hat{t}_{\bar{m}}, \hat{V}_{\bar{m}} \rangle$, where:*

- $\hat{S}_{\bar{m}} \triangleq S_{\bar{m}} \times \mathcal{P}(\bigcup_{i=1}^{k} S_i)$ *is the set of abstract global states.*
- $\hat{\iota}_{\bar{m}} \triangleq (\iota_{\bar{m}}, \{\iota_1, \ldots, \iota_k\})$ *is the initial abstract global state.*
- $\hat{Act}_{\bar{m}} \triangleq Act_{\bar{m}} \cup \bigcup_{i=1}^{k}((A_i \cup AE_i) \times S_i \times \{\uparrow, \downarrow\})$ *is the set of abstract global actions.*
- $\hat{P}_{\bar{m}} : \hat{S}_{\bar{m}} \to \mathcal{P}(\hat{Act}_{\bar{m}})$ *is defined by:*

$$\begin{aligned}\hat{P}_{\bar{m}}(g, X) \triangleq\ & (P_{\bar{m}}(g) \setminus \{a \in GS : \exists i \in \{1, \ldots, k\}, x \in X \text{ with } x \in S_i \text{ and } a \notin P_i(x)\}) \\ & \cup \bigcup_{i=1}^{k} \{(a, s, v) \in A_i \times X \times \{\uparrow, \downarrow\} : a \in P_i(s)\} \\ & \cup \bigcup_{i=1}^{k} \{(a, s, v) \in AE_i \times X \times \{\uparrow, \downarrow\} : a \in P_i(s) \cap P_E(g.E)\}\end{aligned}$$

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

$$\hat{t}_{\bar{m}}((g,X),(a,l,\uparrow),(g',X')) \triangleq \begin{cases} t_i(l,a,l') & \textit{if } X' \setminus X = \{l'\} \textit{ and } l,l' \in S_i \textit{ and } g = g' \\ \sum_{l' \in X \cap S_i} t_i(l,a,l') & \textit{if } X' = X \textit{ and } l \in S_i \textit{ and } g = g' \\ 0 & \textit{otherwise} \end{cases}$$

$$\hat{t}_{\bar{m}}((g,X),(a,l,\downarrow),(g',X')) \triangleq \begin{cases} t_i(l,a,l') & \textit{if } X' = (X \setminus \{l\}) \cup \{l'\} \textit{ and } l \in S_i \\ & \quad \textit{and } l' \in (S_i \setminus X) \cup \{l\} \textit{ and } g = g' \\ \sum_{l' \in X' \cap S_i} t_i(l,a,l') & \textit{if } X' = X \setminus \{l\} \textit{ and } l \in S_i \textit{ and } g = g' \\ 0 & \textit{otherwise} \end{cases}$$

$$\hat{t}_{\bar{m}}((g,X),(a,i),(g',X')) \triangleq \begin{cases} t_{\bar{m}}(g,a,g') & \textit{if } X' = X \\ 0 & \textit{otherwise} \end{cases}$$

For agent-environment actions, we similarly define:

$$\hat{t}_{\bar{m}}((g,X),(a,l,\uparrow),(g',X')) \triangleq t_E(g.E,a,g'.E) \times \begin{cases} t_k(l,a,l') & \textit{if } X' \setminus X = \{l'\} \textit{ and } l,l' \in S_k \\ & \quad \textit{and } \forall (i,j) \in \mathcal{A}(\bar{m}) : g.(i,j) = g'.(i,j) \\ \sum_{l' \in X \cap S_k} t_k(l,a,l') & \textit{if } X' = X \textit{ and } l \in S_k \\ & \quad \textit{and } \forall (i,j) \in \mathcal{A}(\bar{m}) : g.(i,j) = g'.(i,j) \\ 0 & \textit{otherwise} \end{cases}$$

$$\hat{t}_{\bar{m}}((g,X),(a,l,\downarrow),(g',X')) \triangleq t_E(g.E,a,g'.E) \times \begin{cases} t_k(l,a,l') & \textit{if } X' = (X \setminus \{l\}) \cup \{l'\} \textit{ and } l \in S_k \textit{ and } l' \in (S_k \setminus X) \cup \{l\} \\ & \quad \textit{and } \forall (i,j) \in \mathcal{A}(\bar{m}) : g.(i,j) = g'.(i,j) \\ \sum_{l' \in X' \cap S_k} t_k(l,a,l') & \textit{if } X' = X \setminus \{l\} \textit{ and } l \in S_k \\ & \quad \textit{and } \forall (i,j) \in \mathcal{A}(\bar{m}) : g.(i,j) = g'.(i,j) \\ 0 & \textit{otherwise} \end{cases}$$

$$\hat{t}_{\bar{m}}((g,X),(a,i),(g',X')) \triangleq \begin{cases} t_{\bar{m}}(g,a,g') & \textit{if } X' = X \\ 0 & \textit{otherwise} \end{cases}$$

For asynchronous environment actions, we define:

$$\hat{t}_{\bar{m}}((g,X),a,(g',X')) \triangleq \begin{cases} t_{\bar{m}}(g,a,g') & \textit{if } X' = X \\ 0 & \textit{otherwise} \end{cases}$$

Finally, for global actions we define:

$$\hat{t}_{\bar{m}}((g,X),a,(g',X')) \triangleq \begin{cases} t_{\bar{m}}(g,a,g') & \textit{if } X' = \{s' \in S \mid \exists i \in \{1,\ldots,k\}, s \in X \cap S_i : t_i(s,a,s') = 1\} \\ 0 & \textit{otherwise} \end{cases}$$

- $\hat{V}_{\bar{m}} : \hat{S}_{\bar{m}} \to \mathcal{P}((AP \times \mathcal{A}(\bar{m})) \cup AP)$ *is the labelling function given by* $\hat{V}_{\bar{m}}(g, X) \triangleq V_{\bar{m}}(g)$.

Intuitively, the abstract global state records the precise state of $\bar{m}$ agents only. This is because the problem involves checking an $\bar{m}$-indexed formula. All other agents in the concrete systems are represented via a set of states whose elements are the local states that have *one or more* agents in that state. We will refer to the agents corresponding to the second component of the state as the *abstract agents*.

Abstract global actions are either concrete actions of the first $\bar{m}$ agents, or are actions of the abstract agents. In the latter case, we add some further labelling to the actions to resolve how the transition occurs. In particular, we label the action with both the state it was performed from, and whether the agent performing the action was the only agent in that state ("shrinking" actions, labelled by $\downarrow$) or one of several ("growing" actions, labelled by $\uparrow$).

The protocol then enables all concrete actions for the concrete agents (with the exception of global-synchronous actions that cannot be performed by one of the abstract agents). For the abstract agents, it enables their actions along with the corresponding labelling (notice that, since we do not actually know how many agents are in each state, we always enable both the $\uparrow$ and the $\downarrow$ version of actions).

For the labelling function, we are only interested in the $\bar{m}$ concrete agents in the first component of the state (since these are the ones for which corresponding atomic propositions are used in the formula), so we simply discard the second component.

It remains to check that the transition function we have defined gives a valid probability distribution. We do so below.

Observation 1. *For any* $s \in \hat{S}_{\bar{m}}$ *and* $a \in \hat{Act}_{\bar{m}}$*, it is the case that:*

$$\sum_{s' \in \hat{S}_{\bar{m}}} \hat{t}_{\bar{m}}(s, a, s') = 1$$

Proof. Let $\hat{s} = (g, X) \in \hat{S}_{\bar{m}}$. We consider different choices of action.

- $a \in Act_{\bar{m}} \setminus GS$. Then:

  $$\sum_{s' \in \hat{S}_{\bar{m}}} \hat{t}_{\bar{m}}((g, X), a, s') = \sum_{g' \in S_{\bar{m}}} \hat{t}_{\bar{m}}((g, X), a, (g', X)) = \sum_{g' \in S_{\bar{m}}} t_{\bar{m}}(g, a, g') = 1$$

  The equalities follow by noting which terms are non-zero, then applying the definition of $\hat{t}$ and finally using Lemma 1.

- $a \in GS$. Similarly to the above case, we have:

  $$\sum_{s' \in \hat{S}_{\bar{m}}} \hat{t}_{\bar{m}}((g, X), a, s') = \sum_{g' \in S_{\bar{m}}} \hat{t}_{\bar{m}}((g, X), a, (g', X')) = \sum_{g' \in S_{\bar{m}}} t_{\bar{m}}(g, a, g') = 1$$

  where $X' = \{s' \in S | \exists i \in \{1, \ldots, k\}, s \in X \cap S_i : t_i(s, a, s') = 1\}$. Once again, this follows by picking out the non-zero terms of the sum, applying the definition of $\hat{t}$, and finally using Lemma 1.

- $a = (a', l, \uparrow)$ for $a' \in A_i$ and $l \in S_i$. Then:

  $$\sum_{s' \in \hat{S}_{\bar{m}}} \hat{t}_{\bar{m}}((g, X), a, s') = \sum_{l' \in S_i \setminus X} t_i(l, a, l') + \sum_{l' \in X \cap S_i} t_i(l, a, l') = \sum_{l' \in S_i} t_i(l, a, l') = 1$$

by picking out the non-zero terms, performing some rearranging and then noting that t_i is a valid transition function.

- $a = (a', l, \downarrow)$ for $a' \in A_i$ and $l \in S_i$. This is almost identical to the above case.
- $a = (a', l, \uparrow)$ for $a' \in AE_i$ and $l \in S_i$. Then:

$$\begin{aligned}\sum_{s' \in \hat{S}_{\bar{m}}} \hat{t}_{\bar{m}}((g, X), a, s') &= \sum_{s_E \in S_E} \left(\sum_{l' \in S_i \setminus X} t_E(g.E, a', s_E) t_i(l, a, l') + \sum_{l' \in X \cap S_i} t_E(g.E, a', s_E) t_i(l, a, l') \right) \\ &= \sum_{s_E \in S_E} \left(t_E(g.E, a', s_E) \sum_{l' \in S_i} t_i(l, a, l') \right) \\ &= \sum_{s_E \in S_E} t_E(g.E, a', s_E) = 1\end{aligned}$$

by picking out the non-zero terms, performing some rearranging and then noting first that t_i is a valid transition function and subsequently that t_E is also a valid transition function.

- $a = (a', l, \downarrow)$ for $a' \in AE_i$ and $l \in S_i$. This is almost identical to the above case.

This covers all possible cases for the action a and completes our proof. $\square$

We refer to Figure 3 to exemplify the definitions of abstract model, where the first few transitions are depicted. From the initial state, the two concrete agents $(1, 1)$ and $(2, 1)$ may perform their concrete transitions with a and e actions exactly as in the concrete system. Further, the abstract agents can also perform a or e. This gives abstract actions like $(a, 1, \uparrow)$ which indicates one of several abstract agents in state 1 performing the a action and may result in the abstract state being updated from $\{1, 3\}$ to $\{1, 2, 3\}$. The abstract action $(a, 1, \downarrow)$ corresponds to the last abstract agent in state 1 performing the a action and may result in the abstract state being updated from $\{1, 3\}$ to $\{2, 3\}$. The transitions for the abstract e action are similar.

Having defined abstract systems, we now formalise the extent to which paths in concrete models correspond to paths in the corresponding abstract model. As a stepping stone to doing so, we now define how a concrete state can be mapped to an abstract one.

Definition 16 (State abstraction). *Let $\bar{n}, \bar{m} \in \mathbb{Z}^k$ with $\bar{n}_i > \bar{m}_i$ for all i. Then, the abstraction map on states $\lambda_{\bar{n},\bar{m}} : S_{\bar{n}} \to S_{\bar{m}}$ is given by:*

$$\lambda_{\bar{n},\bar{m}}(\bar{s}_1, \dots, \bar{s}_k, s_e) \triangleq ((\bar{s}'_1, \dots, \bar{s}'_k, s_e), X) \text{ where}$$
$$\bar{s}'_i \triangleq (\bar{s}_{i,1}, \dots, \bar{s}_{i,\bar{m}_i}) \text{ and } X \triangleq \bigcup_{i=1}^{k} \{\bar{s}_{i,\bar{m}_i+1}, \dots, \bar{s}_{i,\bar{n}_i}\}$$

Intuitively, for each type of agent i this map preserves the exact state of the first $\bar{m}_i$ such agents in the first component. The states of the remaining agents are projected into the set in the second component, thus preserving which states are present in one or more agents, but discarding the information on precisely how many agents are in each state. We now define a similar abstraction notion for actions.

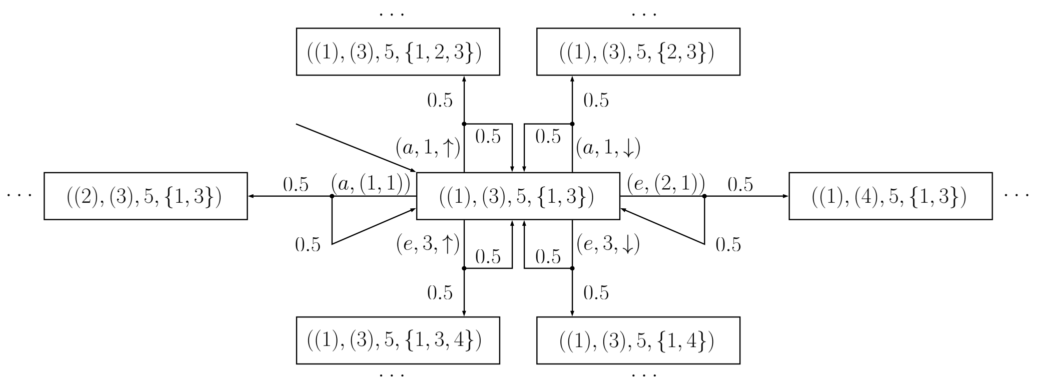


Figure 3: The first state and its outgoing transitions of the abstract system $\hat{\mathcal{S}}((1, 1))$ where $\mathcal{S}$ is the PPIIS from Figure 1. Note the full abstract system is much larger (it contains 19 states and 105 transitions) so is not shown.

Definition 17 (Action abstraction). *Let $\bar{n}, \bar{m} \in \mathbb{Z}^k$ with $\bar{n}_i > \bar{m}_i$ for all i. Then, the abstraction map on actions $\lambda_{\bar{n},\bar{m}} : S_{\bar{n}} \times Act_{\bar{n}} \to \hat{Act}_{\bar{m}}$ is given by:*

$$\lambda_{\bar{n},\bar{m}}((\bar{s}_1, \ldots, \bar{s}_k, s_e), a) \triangleq \begin{cases} a & \text{if } a \in Act_{\bar{m}} \\ (a', \bar{s}_{i,j}, \downarrow) & \text{if } a = (a', (i, j)) \text{ for some } a' \in (A_i \cup AE_i) \text{ and} \\ & \quad (i, j) \in \mathcal{A}(\bar{n}) \setminus \mathcal{A}(\bar{m}) \text{ with} \\ & \quad |\{j' \in \bar{m}_i + 1, \ldots, \bar{n}_i | \bar{s}_{i,j} = \bar{s}_{i,j'}\}| = 1 \\ (a', \bar{s}_{i,j}, \uparrow) & \text{if } a = (a', (i, j)) \text{ for some } a' \in (A_i \cup AE_i) \text{ and} \\ & \quad (i, j) \in \mathcal{A}(\bar{n}) \setminus \mathcal{A}(\bar{m}) \text{ with} \\ & \quad |\{j' \in \bar{m}_i + 1, \ldots, \bar{n}_i | \bar{s}_{i,j} = \bar{s}_{i,j'}\}| > 1 \end{cases}$$

Intuitively, for each type of agent i this maps actions concerning the first $\bar{m}_i$ such agents to themselves. For actions of the remaining agents, these are labelled with the state that they occurred from and whether they were shrinking (↓) or growing (↑) ones.

We now give a technical lemma that will be helpful to prove the validity of the verification algorithm later. Intuitively, this result assures that the probability of a transition in the abstract model is precisely the sum of the probabilities of the concrete transitions it could represent.

Lemma 2. *Let $\bar{n}, \bar{m} \in \mathbb{Z}^k$ with $\bar{n}_i > \bar{m}_i$ for all i. Let $g \in S_{\bar{n}}$ and $a \in P_{\bar{n}}(g)$. Then, for any $\hat{g}' \in \hat{S}_{\bar{m}}$:*

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

Proof. Let $g = (s_1, \ldots, s_k, s_e)$ and $\lambda_{\bar{n},\bar{m}}(g) = ((\hat{s}_1, \ldots, \hat{s}_k, \hat{s}_e), X)$. Denote $\hat{g}' = ((\hat{s}'_1, \ldots, \hat{s}'_k, \hat{s}'_e), X')$. We now consider different possible cases for a:

- $a \in A_E$. Note that if $X \neq X'$, both sides of the equation are equivalent to 0 following the definitions. Similarly, if for any $i \in \mathbb{Z}$ we have $\hat{s}_i \neq \hat{s}'_i$ then both sides are 0. It remains to consider the case where $X = X'$ and $\hat{s}_i = \hat{s}'_i$ for all $i \in \mathbb{Z}$. Then, note that:

$$\hat{t}(\lambda_{\bar{n},\bar{m}}(g), \lambda_{\bar{n},\bar{m}}(g, a), \hat{g}') = t_{\bar{m}}((\hat{s}_1, \ldots, \hat{s}_k, \hat{s}_e), a, (\hat{s}_1, \ldots, \hat{s}_k, \hat{s}'_e)) = t_E(s_e, a, \hat{s}'_e)$$

by following the definitions from the left hand side. On the right hand side we have:

$$\sum_{g' \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{g}')} t_{\bar{n}}(g, a, g') = t_{\bar{n}}((s_1, \ldots, s_k, s_e), a, (s_1, \ldots, s_k, \hat{s}'_e)) = t_E(s_e, a, \hat{s}'_e)$$

with the first equality following from the fact that since $a \in A_E$, the only non-zero cases for $t_{\bar{n}}$ are those where only the state of the environment changes, and the second equality following by definition. Thus, the result holds.

- $a \in GS$. Note that since the agent transitions are deterministic (Definition 6), if we have $X' \neq \{s' \in S | \exists i \in \{1, \ldots, k\}, s \in X \cap S_i : t_i(s, a, s') = 1\}$ then both sides would be equal to 0 (the left simply by definition, the right by noting that no $g' \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{g}')$ is reachable from g so every term being summed is 0). So, we need only consider the case when $X' = \{s' \in S | \exists i \in \{1, \ldots, k\}, s \in X \cap S_i : t_i(s, a, s') = 1\}$.

  Further note that (once again by the agent transitions being deterministic) there are unique states $\hat{p}_1, \ldots, \hat{p}_k \in S_{\bar{m}}$ such that $t_{\bar{m}}((\hat{s}_1, \ldots, \hat{s}_k, \hat{s}_e), a, (\hat{p}_1, \ldots, \hat{p}_k, \hat{s}'_e)) \neq 0$. If for any $i \in \mathbb{Z}$ it is the case that $\hat{s}'_i \neq \hat{p}_i$ then both sides are equal to 0. So, we only need to check that case where $\hat{s}'_i = \hat{p}_i$ for all $i \in \mathbb{Z}$. Then, on the left hand side we have:

$$\hat{t}(\lambda_{\bar{n},\bar{m}}(g), \lambda_{\bar{n},\bar{m}}(g, a), \hat{g}') = t_{\bar{m}}((\hat{s}_1, \ldots, \hat{s}_k, \hat{s}_e), a, (\hat{s}_1, \ldots, \hat{s}_k, \hat{s}'_e)) = t_E(s_e, a, \hat{s}'_e)$$

  since all of the deterministic agent transitions are 1, so only the transition of the environment is needed to determine the probability. On the right hand side, we note that once again by the deterministic nature of the agent transitions there is precisely one reachable $g' = (s'_1, \ldots, s'_k, \hat{s}'_e) \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{g}')$. Then:

$$\sum_{g' \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{g}')} t_{\bar{n}}(g, a, g') = t_{\bar{n}}((s_1, \ldots, s_k, \hat{s}_e), a, (s'_1, \ldots, s'_k, \hat{s}'_e)) = t_E(s_e, a, \hat{s}'_e)$$

  with the second equality following once again from the deterministic agent transitions being 1. Thus, the result holds.

- $a = (a', (i, j)) \in A_i \times Agt_{\bar{m}}$. We consider only the case when $X = X'$, $\hat{s}_e = \hat{s}'_e$ and $\hat{s}_{l,m} = \hat{s}'_{l,m}$ for all $(l, m) \neq (i, j)$. If any of those conditions are violated, it follows easily from the definitions that both sides are 0. In the remaining case note that:

$$\hat{t}(\lambda_{\bar{n},\bar{m}}(g), \lambda_{\bar{n},\bar{m}}(g, a), \hat{g}') = t_i(\hat{s}_{i,j}, a, \hat{s}'_{i,j}) = t_i(s_{i,j}, a, s'_{i,j})$$

  by definition. On the right hand side, note that the only non-zero term of the sum is when $g' = (s'_1, \ldots, s'_k, \hat{s}'_e)$ with $s'_{l,m} = s_{l,m}$ for all $(l, m) \neq (i, j)$ and $s'_{i,j} = \hat{s}'_{i,j}$. Thus:

$$\sum_{g' \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{g}')} t_{\bar{n}}(g, a, g') = t_{\bar{n}}((s_1, \ldots, s_k, s_e), a, (s'_1, \ldots, s'_k, s'_e)) = t_i(s_{i,j}, a, s'_{i,j})$$

- $a = (a', (i, j)) \in AE_i \times Agt_{\bar{m}}$. This case is identical to the one above, except that we needn't have $\hat{s}_e = \hat{s}'_e$ and thus the final result is multiplied by $t_E(s_e, a, \hat{s}'_e)$ to account for the environment's transition.

- $a = (a', (i, j)) \in A_i \times (Agt_{\bar{n}} \setminus Agt_{\bar{m}})$. We consider only the case when $\hat{s}_e = \hat{s}'_e$ and $\hat{s}_l = \hat{s}'_l$ for all l. If either of those conditions are violated, it follows easily from the definitions that both sides are 0. We further subdivide this case into two:

  – $\{j' \in \bar{m}_i + 1, \ldots, \bar{n}_i | s_{i,j} = s_{i,j'}\} = 1$. Then, $\lambda(g, a) = (a', s_{i,j}, \downarrow)$. Here, there are two non-zero cases. If $X' = X \setminus \{s_{i,j}\}$ then on the LHS we have:

  $$\hat{t}(\lambda_{\bar{n},\bar{m}}(g), (a', s_{i,j}, \downarrow), \hat{g}') = \sum_{l' \in X \cap S_i} t_i(s_{i,j}, a', l')$$

  by definition. But note this is equal to the RHS since the only $g' = (s'_1, \ldots, s'_k, s_e) \in \lambda^{-1}(\hat{g}')$ for which $t_{\bar{n}}(g, a, g')$ is non-zero are those where $s'_o = s_o$ for all $o \neq (i, j)$, and in order to be in the pre-image, it must be the case that $s_{i,j} \in X \cap S_i$.

  The other non-zero case is when $X' = (X \setminus \{s_{i,j}\}) \cup \{l'\}$ for some $l' \in S_i \setminus X$. Then, on the LHS we have:

  $$\hat{t}(\lambda_{\bar{n},\bar{m}}(g), (a', s_{i,j}, \downarrow), \hat{g}') = t_i(s_{i,j}, a', l')$$

  by definition. But this is equal to the RHS, since the only $g' = (s'_1, \ldots, s'_k, s_e) \in \lambda^{-1}(\hat{g}')$ for which $t_{\bar{n}}(g, a, g')$ is non-zero is the one with $s'_o = s_o$ for all $o \neq (i, j)$ and $s'_{i,j} = l'$.

  – $\{j' \in \bar{m}_i + 1, \ldots, \bar{n}_i | s_{i,j} = s_{i,j'}\} > 1$. Then, $\lambda(g, a) = (a', s_{i,j}, \uparrow)$, and once again there are two non-zero cases which are similar to the ones for the case above.

- $a = (a', (i, j)) \in AE_i \times (Agt_{\bar{n}} \setminus Agt_{\bar{m}})$. This case is identical to the one above, except that we needn't have $\hat{s}_e = \hat{s}'_e$ and thus the final result is multiplied by $t_E(s_e, a, \hat{s}'_e)$ to account for the environment's transition.

This covers every possible case for our action and completes the proof. □

Having defined the notion of abstraction on states and actions and investigated the probability of a transition in the abstract model, we extend these notions to execution paths.

Definition 18 (Path abstraction)**.** *Let $\bar{n}, \bar{m} \in \mathbb{Z}^k$ with $\bar{n}_i > \bar{m}_i$ for all i. Then, the abstraction map on paths $\lambda_{\bar{n},\bar{m}} : IPaths_{S(\bar{n})} \to IPath_{\hat{S}(\bar{m})}$ is given by mapping each infinite path $\rho = g_0 a_0 g_1 \ldots$ in $S(\bar{n})$ to the unique infinite path $\hat{\rho} = \hat{g}_0 \hat{a}_0 \hat{g}_1 \ldots$ in $\hat{\mathcal{S}}(\bar{m})$ such that, for all $i \in \mathbb{N}$: (i) $\hat{g}_i = \lambda_{\bar{n},\bar{m}}(g_i)$; (ii) $\hat{a}_i = \lambda_{\bar{n},\bar{m}}(g_i, a_i)$.*

We restrict this definition to finite paths in the obvious way, i.e., by mapping a finite path to abstract finite path of the same such that conditions (i) and (ii) hold for all i up to the length of the path.

We now proceed to define the notion of an equivalent scheduler for the abstract model. Given a scheduler in a concrete system of size $\bar{n}$, we would like to have a scheduler in the abstract model of size $\bar{m}$, which achieves "the same behaviour" (to be formalised later in Lemma 3).

Definition 19 (Equivalent scheduler)**.** *Let $\sigma : FPaths_{S(\bar{n})} \times Act_{\bar{n}} \to [0, 1]$ be a scheduler in $\mathcal{S}(\bar{n})$. Then, for any $\bar{m}$ with $\bar{m}_i < \bar{n}_i$ for all i, we define the equivalent scheduler $\hat{\sigma} : FPath_{\hat{S}(\bar{m})} \times \hat{Act}_{\bar{m}} \to [0, 1]$ by:*

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

Intuitively, since in the abstract model we do not know precisely which path would have been followed in the concrete model, we have to consider all paths that could have been followed (i.e., those in the preimage of $\lambda_{\bar{n},\bar{m}}$) and weigh the probabilistic choice of the next action according to the probability of each path.

Before proceeding in this direction, we show that this definition is indeed a valid scheduler (i.e., it only gives actions that are permitted by the protocol function and it defines a probability distribution on the next action). We do so below.

Observation 2. $\hat{\sigma} : FPath_{\hat{\mathcal{S}}(\bar{m})} \times \hat{Act}_{\bar{m}} \to [0,1]$ *is a valid scheduler.*

Proof. First, suppose that $\hat{\sigma}(\hat{\rho}, \hat{a}) > 0$. Then, there is at least one $\rho \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{\rho})$ and one $a \in Act_{\bar{n}}$ such that $\lambda_{\bar{n},\bar{m}}(last(\rho), a) = \hat{a}$ and $\sigma(\rho, a) > 0$. Then it must be the case that $a \in P_{\bar{n}}(last(\rho))$ since σ is a valid scheduler. It follows that $\hat{a} \in \hat{P}_{\bar{n}}(last(\hat{\rho}))$

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

$$\begin{aligned}
&\sum_{\rho\in\lambda^{-1}_{\bar{n},\bar{m}}(\hat{\rho})} \sum_{\hat{a}\in\hat{Act}_{\bar{m}}} \sum_{a\in Act_n : \lambda_{\bar{n},\bar{m}}(last(\rho),a)=\hat{a}} \mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho) \cdot \sigma(\rho, a) \\
&= \sum_{\rho\in\lambda^{-1}_{\bar{n},\bar{m}}(\hat{\rho})} \sum_{a\in Act_{\bar{n}}} \mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho) \cdot \sigma(\rho, a) = \sum_{\rho\in\lambda^{-1}_{\bar{n},\bar{m}}(\hat{\rho})} \mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho)
\end{aligned}$$

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

Having proved the validity of our definition, we present a further Lemma. This shows the desirable property that when following the equivalent scheduler, the probability of a path in the abstract model is exactly equal to the sum of the probabilities of the concrete paths that it could represent.

Lemma 3. *Let $\sigma : FPath_{S(\bar{n})} \times Act_{\bar{n}} \to [0,1]$ be a scheduler in $\mathcal{S}(\bar{n})$. Then it is the case that, for any path $\hat{\rho} \in FPath_{\hat{\mathcal{S}}(\bar{m})}$:*

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

Proof. We proceed by induction on the length of the path $\hat{\rho}$. For the base case, note the only path of length 1 has just the initial state $\hat{\iota}_{\bar{m}}$. This has probability 1. Further, $\rho = \iota$ is the only element of $\lambda^{-1}_{\bar{n},\bar{m}}(\hat{\rho})$ for which $\mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho)$ is non-zero, and this is also 1. So, both sides of the equation are 1, and the statement holds.

Now, suppose the statement holds for a path $\hat{\rho}'$ and consider its extension $\hat{\rho} = \hat{\rho}'\hat{a}\hat{s}$ for some

$\hat{a} \in \hat{Act}_{\bar{m}}$ and $\hat{s} \in \hat{S}_{\bar{m}}$. Now:

$$\begin{aligned}
\mathbf{P}_{\hat{\mathcal{S}}(\bar{m})_{\hat{\sigma}}}(\hat{\rho}) &= \mathbf{P}_{\hat{\mathcal{S}}(\bar{m})_{\hat{\sigma}}}(\hat{\rho}') \cdot \frac{\sum_{\rho \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{\rho}')} \sum_{a \in Act_{\bar{n}} : \lambda_{\bar{n},\bar{m}}(last(\rho),a)=\hat{a}} \mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho) \cdot \sigma(\rho, a)}{\sum_{\rho \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{\rho}')} \mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho)} \cdot \hat{t}_{\bar{m}}(last(\hat{\rho}'), \hat{a}, \hat{s}) \\
&= \left(\sum_{\rho \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{\rho}')} \sum_{a \in Act_{\bar{n}} : \lambda_{\bar{n},\bar{m}}(last(\rho),a)=\hat{a}} \mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho) \cdot \sigma(\rho, a) \right) \cdot \hat{t}_{\bar{m}}(last(\hat{\rho}'), \hat{a}, \hat{s}) \\
&= \sum_{\rho \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{\rho}')} \sum_{a \in Act_{\bar{n}} : \lambda_{\bar{n},\bar{m}}(last(\rho),a)=\hat{a}} \left(\mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho) \cdot \sigma(\rho, a) \cdot \sum_{s \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{s})} t_{\bar{n}}(last(\rho), a, s) \right) \\
&= \sum_{\rho \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{\rho})} \mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho)
\end{aligned}$$

with the first equality following from the definitions, the second from the inductive hypothesis, the third from Lemma 2 and the final one simply by rearranging. Thus, by induction on the length of the path, the lemma holds for all finite paths. □

We are now ready to present one of the main results on this paper. We show that if an $\bar{m}$-indexed formula of the form $P^{\max}_{\leq x}[\psi]$ holds in the abstract model of size $\bar{m}$, then it holds in any larger concrete system. This will enable us to partially solve the PMCP from Definition 14.

Theorem 1. *Suppose $\hat{\mathcal{S}}(\bar{m}) \models P^{max}_{\leq x}[\psi]$ for some $\bar{m}$-indexed formula ψ. Then, $\mathcal{S}(\bar{n}) \models P^{max}_{\leq x}[\psi]$ for all $\bar{n} \in \mathbb{Z}^k$ with $\bar{n}_i > \bar{m}_i$ for all i.*

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

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

Now, let $\hat{\sigma}$ be the equivalent scheduler in $\hat{\mathcal{S}}(\bar{m})$ as given by Definition 19. But now 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 thus the same LTL formulas hold along the path). So:

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

since by Lemma 3 the probability of a path in the abstract model is at least as much as that of the corresponding path in the concrete model.

Thus, by considering $\hat{\sigma}$ as our scheduler, we see that $\hat{\mathcal{S}}(\bar{m}) \models P^{\max}_{>x}[\psi]$. □

Note the above theorem also gives a method to check properties of the form $P^{\min}_{>x}[\psi]$ since this is equivalent to checking $P^{\max}_{\leq 1-x}[\neg\psi]$. Further, if we can verify $P^{\max}_{\leq x}[\psi]$ then certainly it cannot be the case that $P^{\max}_{>x}[\psi]$. This enables us to falsify properties of the form $P^{\max}_{>x}[\psi]$ (and thus also properties of the form $P^{\min}_{\leq x}[\psi]$, which are equivalent to $P^{\max}_{>1-x}[\neg\psi]$).

Theorem 2. *Suppose $\hat{\mathcal{S}}(\bar{m}) \models P^{max}_{<x}[\psi]$ for some $\bar{m}$-indexed formula ψ. Then, $\mathcal{S}(\bar{n}) \models P^{max}_{<x}[\psi]$ for all $\bar{n} \in \mathbb{Z}^k$ with $\bar{n}_i > \bar{m}_i$ for all i.*

Proof. This is identical to the proof for Theorem 1 with strict and non-strict inequalities swapped. □

Algorithm 1 Decision Procedure for the PMCP

```
    Input: PPIIS $\mathcal{S}$ with $k$ agent templates, $\bar{m}$-indexed formula $\phi$
    Output: Boolean, or FAIL
 1: function CHECK($\mathcal{S}$, $\phi$)
 2:     switch $\phi$ do
 3:         case $\phi = P^{\max}_{\leq x}[\psi]$ or $\phi = P^{\max}_{<x}[\psi]$
 4:             if $\hat{\mathcal{S}}(\bar{m}) \not\models \phi$ then
 5:                 return FAIL
 6:             end if
 7:             for $\bar{n} \leftarrow \{\bar{n} \in (\mathbb{Z}^+)^k \mid \bar{m}_i \leq \bar{n}_i \leq \bar{m}_i + 1$ for $i = 1, \ldots, k\}$ do
 8:                 if $\mathcal{S}(\bar{n}) \not\models \phi$ then
 9:                     return false
10:                 end if
11:             end for
12:             return true
13:         case $\phi = P^{\max}_{>x}[\psi]$
14:             if Check($\mathcal{S}$, $P^{\max}_{\leq x}[\psi]$) = true then
15:                 return false
16:             end if
17:             return FAIL
18:         case $\phi = P^{\max}_{\geq x}[\psi]$
19:             if Check($\mathcal{S}$, $P^{\max}_{<x}[\psi]$) = true then
20:                 return false
21:             end if
22:             return FAIL
23:         case $\phi = P^{\min}_{>x}[\psi]$
24:             return Check($\mathcal{S}$, $P^{\max}_{\leq 1-x}[\neg\psi]$)
25:         case $\phi = P^{\min}_{\geq x}[\psi]$
26:             return Check($\mathcal{S}$, $P^{\max}_{<1-x}[\neg\psi]$)
27:         case $\phi = P^{\min}_{<x}[\psi]$
28:             return Check($\mathcal{S}$, $P^{\max}_{\geq 1-x}[\neg\psi]$)
29:         case $\phi = P^{\min}_{\leq x}[\psi]$
30:             return Check($\mathcal{S}$, $P^{\max}_{>1-x}[\neg\psi]$)
31: end function
```

As before, this theorem can also be used to verify properties of the form $P^{\min}_{\geq x}[\psi]$, and to falsify properties of the form $P^{\max}_{\geq x}[\psi]$ or $P^{\min}_{<x}[\psi]$.

All the above combine to give our partial decision procedure for solving the PMCP, which can be seen in Algorithm 1. The main case of the algorithm is when the formula ϕ being checked is either of the form $P^{\max}_{\leq x}[\psi]$ or $P^{\max}_{<x}[\psi]$. Other cases are transformed into this case by checking the negation of the formula. In the main case, the algorithm constructs (using Definition 15) the abstract system $\hat{\mathcal{S}}(\bar{m})$, where $\bar{m}$ is the index of the formula. If the formula holds on this abstract system, then we know by Theorem 1 (Theorem 2, respectively) that it will hold in systems larger than $\bar{m}$, so it only remains to check systems of size up to $\bar{m}$. If the formula holds in all of these, then we return *true*. If not, then we return *false*. If the formula does not hold in the abstract system, then we cannot make any claim on whether it will hold in systems of arbitrary size, so we simply return `FAIL`.

We now prove the correctness of the procedure.

Corollary 1. *Let $\mathcal{S}$ be a PPIIS, and ϕ and $\bar{m}$-indexed formula. Then, if* `Check`$(\mathcal{S}, \phi) = $ *true it is the case that $\mathcal{S} \models \phi$. Further, if* `Check`$(\mathcal{S}, \phi) =$ *false then it is the case that $\mathcal{S} \not\models \phi$.*

Proof. Suppose `Check`$(\mathcal{S}, \phi)$ returns *false* at line 9. Then, there is some $\bar{n}$ such that $\mathcal{S}(\bar{n}) \not\models \phi$. So, it must be the case that $\mathcal{S} \not\models \phi$.

If `Check`$(\mathcal{S}, \phi)$ returns *true* at line 12, then by either Theorem 1 or Theorem 2 (depending on the form of ϕ) it is the case that $\mathcal{S}(\bar{n}) \models \phi$ for all $\bar{n} \in (\mathbb{Z}^+)^k$ with $\bar{n}_i > \bar{m}_i$ for all i. Further, since the algorithm did not exit at line 9, then $\mathcal{S}(\bar{n}) \models \phi$ for all $\bar{n} \in (\mathbb{Z}^+)^k$ with $\bar{m}_i \leq \bar{n}_i \leq \bar{m}_i + 1$ for all i. Together, these two facts prove that $\mathcal{S} \models \phi$.

If `Check`$(\mathcal{S}, \phi)$ returns *false* at line 15, then $\phi = P^{max}_{>x}[\psi]$ and, by similar reasoning as in previous parts of this proof, then we must have $\mathcal{S} \models P^{max}_{\leq x}[\psi]$. Thus, it must be the case that $\mathcal{S} \not\models \phi$. The same argument proves the return value at line 20.

For the return value at line 24, note that checking $P^{min}_{>x}[\psi]$ is identical to checking $P^{max}_{\leq 1-x}[\neg\psi]$ since the scheduler that minimises ψ is just the one that maximises $\neg\psi$, and if this achieves a probability $\leq 1 - x$ of satisfying $\neg\psi$, it will certainly achieve a probability $> x$ of satisfying ψ. A similar argument shows the validity of the return values at lines 26, 28 and 30. □

Thus, we have showed that the `Check` procedure's result (when it returns one) is always valid. Further, note that the procedure always terminates. This can be observed by noting that when a recursive call is made, it will always go into one of the cases higher up, and the only loop present (line 7) is over a finite set. However, note that the procedure is not complete since in some cases it will not return a result but will return `FAIL` instead. This is inevitable, since the PMCP decision problem it is solving (Definition 14) is a more general version of one that is already known to be undecidable [13] and thus is also undecidable.

5. Implementation and Evaluation

We now report on an implementation of the method described in the previous section and present the experimental results obtained by verifying swarm protocols.

5.1. Implementation Details

The Java toolkit `PSV-CA` (**P**robabilistic **S**warm **V**erifier by **C**ounter **A**bstraction) is built as an extension of PRISM 4.0 [19]. The toolkit takes as input a description of the behaviour of

```
asynchronous      = {a}
agentEnvironment  = {e}
globalSynchronous = {g}

agent module AgentA
  stateA : [1..2] init 1;

  [a] (stateA=1) -> 0.5:(stateA'=1) + 0.5:(stateA'=2);
  [g] (stateA=2) -> 1.0:(stateA'=2);
endmodule

agent module AgentB
  stateB : [3..4] init 3;

  [e] (stateB=3) -> 0.5:(stateB'=3) + 0.5:(stateB'=4);
  [g] (stateB=4) -> 1.0:(stateB'=4);
endmodule

environment module Environment
  stateE : int init 5;

  [e] (stateE=5) -> 1.0:(stateE'=5);
  [g] (stateE=5) -> 0.5:(stateE'=5) + 0.5:(stateE'=6);
  [g] (stateE=6) -> 1.0:(stateE'=6);
endmodule

label "firstAgentTransitioned" = (stateA_0_0 = 2);
```

Figure 4: The code modelling the example from Figure 1 in PSV-CA.

one or more agent templates and of the environment. The specifications to be checked are given in PLTL. The language used to describe the system is inspired by that used in PRISM (which we also re-used some of the parsing procedures from), suitably extended to handle parameterised systems. In particular, the declaration of variables and updates described below is the same as that used by PRISM. However, the definining of action types as asynchronous, agent-environment or global-synchronous is novel, as is the fact that we define distinct agent and environment modules which give rise to the different synchronisation patterns.

Depending on the arguments passed to the toolkit, PSV-CA will either construct the abstract model (according to Definition 15) or the concrete model of a given number of agents (according to Definition 11). It then passes this to PRISM to verify the properties given against it. The source code and a number of case studies reported below are released as open source [72].

An example model can be seen in Figure 4. The code begins with three sets that define the type of each action (asynchronous, agent-environment or global synchronous). Every action used later in the model must appear in one of these sets in order to specify what synchronisation is required when performing the action.

This is followed by a number of agent modules. Each agent module defines a number of variables using the syntax

$$\texttt{v}_i\ \texttt{:}\ \ \texttt{t}_i\ \texttt{init}\ k_i\texttt{;}$$

which declares a new variable $\texttt{v}_i$ of type $\texttt{t}_i$ with initial value $\texttt{k}_i$. The variable types supported are the same as those in PRISM – they can either be Booleans (declared with `bool`), unbounded integers (declared with `int`) or integers bounded between *a* and *b* (declared with `[a, b]`). Note that while unbounded integers can be used for convenience, in order for `PSV-CA` to terminate it must be the case that the part of the model that is actually explored is finite.

These variable definitions are followed by a number of update actions of the form

$$\texttt{[actionName] (guard) -> p}_1\texttt{:(update}_1\texttt{) + ... + p}_n\texttt{:(update}_n\texttt{);}$$

Update actions as above are interpreted as follows: If `guard` evaluates to `true` in the current state, then the agent may perform action `actionName`. Upon performing this action, there is a probability $\texttt{p}_i$ of $\texttt{update}_i$ being performed for each *i* in order to transition to the next state. Note that it must be the case that $\Sigma_{i\in\{1,\ldots,n\}}\texttt{p}_i = 1$.

The agent modules are followed by an environment module which is defined in exactly the same way as an agent (but is distinct in what synchronisation it gives rise to). Finally, some labels allow us to define which atomic propositions hold at specific points in the execution of the system. These labels are Boolean expressions on the variables of agents and/or the environment. To refer to variable `v` of the *i*th agent of type *j* we use `v_j_i`. To refer to a variable `k` of the environment we use `k_E`. These labels are used in the properties file, which contains a sequence of properties expressed in PLTL, such as

```
P<=0.9 [ F<4 ("firstAgentTransitioned") ]
```

which expresses that the probability of the first agent transitioning to state 2 within four time-steps is at most 0.9. Once `PSV-CA` has constructed the abstract or concrete model (depending on the arguments passed to it), PRISM is used to check each property in the file against it.

Finally, `PSV-CA` can also be passed additional arguments to export the full model constructed to a DOT file or export the optimal strategy for each property under consideration to be satisfied. Further details on these options can be found in the documentation within the software package.

We now describe three case studies that can be analysed by using `PSV-CA`. All timing results were obtained on a machine running Ubuntu 19.10 (Linux kernel 5.3.0) with an Intel i9-9940X processor and 128GB of RAM. The code was run using OpenJDK 1.8.0 (64-bit version), with 96GB of RAM allocated to the Java heap.

5.2. Autonomous robots

As a first case study to verify the applicability of `PSV-CA` we consider a probabilistic variant of the autonomous robots example from [27]. In this scenario, there are a number of robots moving along a track of infinite length. The robots begin at the start of the track and are moved synchronously along by an environment until they decide to stop. The robots aim to stop in a target region (for our experiment, we fixed this to be between 19 and 21 units of distance along the track). Some of the robots are equipped with sensors to detect their position, but others are not and must rely on communication from other agents in order to decide when to stop.

```
...
agent module WithoutSensor
        locationB : [1..7] init 1;
        stoppedB : bool init false;
        [move] (!stoppedB) -> 1.0:(locationB'=locationB+1);
        [signal] (true) -> 1.0:(stoppedB'=true);
endmodule
...
```

Figure 5: A snippet of the code modelling the autonomous robots scenario giving the code for sensorless robots.

For our experiment, we assumed that the sensor of the robots equipped with one is noisy and can give readings up to two below or up to two above the actual position. Further, we assumed that all these readings are equally likely (i.e., they all have a probability of 0.2). The robots with a sensor decide to halt if the reading they receive from their sensor is at least 20. When they halt, they also broadcast a signal to sensorless agents telling them to stop. The model includes a combination of asynchronous, agent-environment and global-synchronous actions. Part of the code for the model can be seen in Figure 5.

We considered the $(0, 1)$-indexed property

$$P^{max}_{\leq p}[F((stopped, (2, 1)) \wedge \neg(target, (2, 1)))]$$

where stopped is an atomic proposition that holds when an agent has halted, and target holds when an agent is within the target region. The property expresses that we are certain that the probability of a sensorless agent stopping outside the target region is at most p. We checked this for a different number of concrete agents with sensors, and recorded the largest p for which this holds in each case. The results are shown in Figure 6.

We see that when the number of agents with sensors is increased the probability also increases. This is as expected, since when the number of agents is increased it is more likely that at least one of the agents with a sensor will misjudge the stopping position and start broadcasting a signal too early, causing our sensorless agent to stop early. Further, note that the maximum probability computed by the abstract model is 1. This corresponds to our intuition that when we have arbitrarily many agents the probability of at least one misjudging the position will tend to 1.

The largest of these models (with 7 concrete agents) has 15,676,417 states and 99,246,301 transitions. It takes around 190 seconds for PSV-CA to construct the model and pass it to PRISM, which then computes the value of p in around 770 seconds.

5.3. *Foraging protocol*

As a further example, we modelled a foraging protocol [28, 73]. In this scenario, agents begin resting in a nest. They may choose to leave the nest and search for food, which they then retrieve and bring back to the nest. Upon returning to the nest the agents return to the resting state.

We considered agents of two types. Both types of agents choose to stop resting with probability 0.5. However, when searching for food the first type of agent will look up to two units of distance away whereas the second will only look one unit of distance away. Accordingly, we

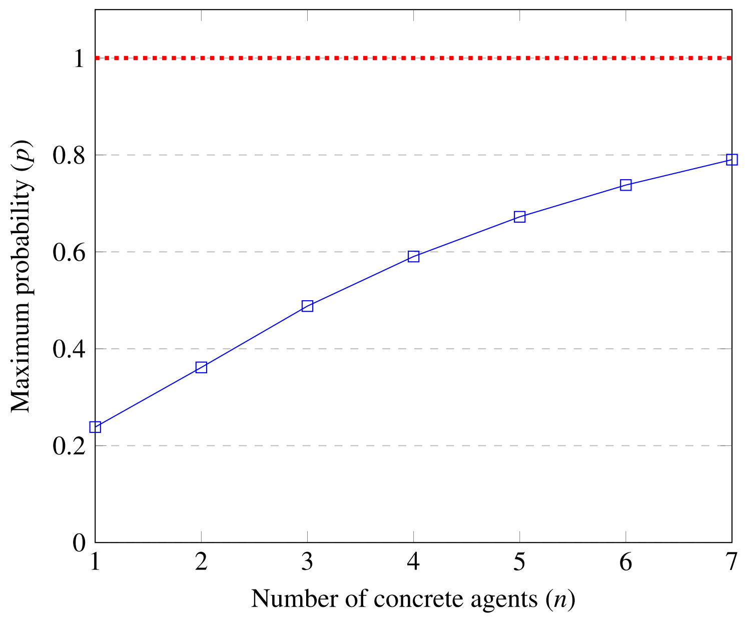


Figure 6: For different numbers of concrete type 1 agents (robots with a sensor), the maximum value of p for which the property $P^{max}_{\leq p}[F((stopped, (2, 1)) \wedge \neg(target, (2, 1)))]$ holds, i.e. how likely it is for a sensorless robot to stop outside the target region. The red line shows the maximum value computed by the abstract model.

gave the second type of agent a probability of 0.15 of finding food when searching for it, and the first a higher probability of 0.3 (with a 0.15 chance it is one unit of distance away, and a 0.15 chance it is two). Both types of agent immediately travel to the food and bring it back to the nest upon locating it.

The model involves a combination of asynchronous and agent-environment actions. Asynchronous actions are used to model the agents moving, searching for food, and collecting the food. Agent-environment actions are used to model the agents depositing food in the nest (which is captured by the environment). Part of the code for our model can be seen in Figure 7.

We used `PSV-CA` to investigate the probability that a unit 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 $(0, 0)$-indexed property

$$P^{\max}_{\leq p}[F^{<k} depositedFood],$$

where *depositedFood* is an atomic proposition that holds when a unit of food has been deposited in the nest by any agent.

the results of checking this property for different values of p and k against the abstract model are recorded in Table 1. Note that when the property holds we are guaranteed by Theorem 1 that it will hold in concrete systems of any size. However, when the property does not hold we cannot make any claims. Incompleteness of the procedure is a necessary consequence of the general undecidability.

The results match our expectations: when the agents are given a longer number of time-steps to collect food then there is a higher probability that they will succeed. The abstract model constructed to verify the properties takes around 255 seconds to construct and has 766,977 states

```
...
agent module Agent1
        state : [0..3] init 0;
        fromHome : [0..2] init 0;
        ...
        [move] state=3 & fromHome>0 -> 1.0:(fromHome'=fromHome-1);
        [deposit] state=3 & fromHome=0 -> 1.0:(state'=0);
endmodule
...
environment module Env
        deposited : [0..1] init 0;
        [deposit] deposited<1 -> 1.0:(deposited'=deposited+1);
endmodule
```

Figure 7: A snippet of the code modelling the foraging scenario giving the part of the model that encodes robots moving back to the nest with food and depositing it.

		k						
		4	6	8	10	12	14	16
	0.10	False	False	False	False	False	False	False
	0.25	True	False	False	False	False	False	False
	0.50	True	True	False	False	False	False	False
	0.75	True	True	True	False	False	False	False
p	0.90	True	True	True	True	False	False	False
	0.95	True	True	True	True	True	False	False
	0.98	True	True	True	True	True	True	False
	0.99	True	True	True	True	True	True	True

Table 1: For different numbers of time-steps k and probabilities p, whether or not the property $P^{max}_{\leq p}[F^{<k}\text{depositedFood}]$ holds, where depositedFood is an atomic proposition that holds when a unit of food has been deposited.

and 20,994,481 transitions. Once the abstract model is constructed, checking the individual properties on it takes a negligible time (around 50ms per property).

5.4. Channel jamming

As a final scenario to further assess the scalability of PSV-CA, we considered the channel jamming scenario described in [29]. In our model of this scenario, there are four different channels that some agents may wish to use to transmit messages to a receiver on. Additionally, there are some jamming agents who can jam the channels the transmitters are using.

At each round of communication the transmitters each choose a channel to transmit on and the jammers each choose a channel to jam. The probability of successful transmission on an unjammed channel is set to 0.8, while on a jammed channel it is reduced to 0.3. The receiver only accepts at most one transmission per communication round with further successful transmissions being ignored.

Our model of this system involves a combination of all three types of actions. Asynchronous actions are used by receivers and jammers to decide which channel they will transmit on or jam.

```
...
agent module Transmitter
        stateT: [1..4] init 1;
        channel : [0..4] init 0;
        ...
        [chooseTransmit] (stateT=2) -> 0.25:(stateT'=3) & (channel'=1)
                + 0.25:(stateT'=3) & (channel'=2)
                + 0.25:(stateT'=3) & (channel'=3)
                + 0.25:(stateT'=3) & (channel'=4);
        ...
endmodule
...
```

Figure 8: A snippet of the code modelling the channel jamming scenario showing how a transmitter chooses which channel to transmit on.

Agent-environment actions are used by them to communicate to the environment which channels have been jammed and to attempt to transmit the message. Finally, global synchronous actions are used to advance between communication rounds. Part of the code for the model can be seen in Figure 8.

We checked the $(0,0)$-indexed property

$$P^{max}_{\leq p}[F(roundsFinished \wedge \neg receivedAll)]$$

where *roundsFinished* is an atomic proposition that holds after some maximum number of communication rounds allowed (m) has passed and *receivedAll* holds after all k messages that are being transmitted have been received. This property expresses that the probability of the transmission not succeeding is less than p. We varied both m and k and recorded the maximum value of p for which the property holds along with some information about the size of the resulting model and time needed to construct and verify it. Our results can be found in Table 2.

The results match our expectations about the protocol. The probability of the transmission failing is highest when the number of messages being transmitted is high and the number of rounds of communication allowed is low. Further, the timing results show that PSV-CA can be used to verify reasonably large systems; the largest case (14 messages, 45 rounds of communication) has over 1 million states and 20 million transitions but is still constructed and verified in under 5 minutes.

6. Conclusion

One of the key difficulties in deploying robotic swarms is providing formal guarantees about their behaviour, particularly when either the behaviour of the robots or the environment they are acting in is stochastic in nature. In this work, we have provided a framework for formalising and verifying heterogenous swarms in which the number of components is unknown at design-time. Specifically, we have introduced PPIIS which give a framework for reasoning about probabilistic swarms. We have observed that the model checking problem of PPIIS against PLTL specifications is in general undecidable, as is typical in parameterised model checking.

		k				
		25	30	35	40	45
m	6	0.1935	0.0766	0.02690	0.0086	0.0026
		281,226	340,806	400,386	459,966	519,546
		5,490,717	6,658,077	7,825,437	8,992,797	10,160,157
		10.9 / 18.1	12.4 / 25.8	15.5 / 37.3	17.0 / 50.3	18.8 / 61.2
	8	0.5118	0.2814	0.1326	0.0553	0.0209
		355,560	434,200	512,840	591,480	670,120
		6,997,560	8,551,290	10,105,020	11,658,750	13,212,480
		13.0 / 29.8	16.2 / 41.9	18.6 / 58.3	23.4 / 76.1	25.7 / 94.6
	12	0.9558	0.8407	0.6516	0.4406	0.2620
		481,356	598,116	714,876	831,636	948,396
		9,547,602	11,874072	14,200,542	16,527,012	18,853,482
		17.5 / 53.4	24.0 / 85.0	27.6 / 116.5	30.4 / 142.2	37.8 / 190.3
	14	0.9940	0.9599	0.8650	0.7032	0.5089
		532,818	668,638	804,458	940,278	1,076,098
		10,590,801	13,303,641	16,016,481	18,729,321	21,442,161
		18.9 / 72.9	25.6 / 108.3	29.9 / 149.8	37.9 / 196.2	43.3 / 250.6

Table 2: For different number of messages (m) and rounds of communication (k) the worst-case probability that the messages won't all be communicated, along with the number of states and transitions in the abstract model used to compute this, and the time in seconds needed to construct this model and compute the value.

Nonetheless, we have developed a partial decision procedure by building an abstract model using counter abstraction. This abstract model gives an over-approximation of the executions that may be exhibited by swarms of arbitrary size. Using this intuition, we have shown that if certain properties are satisfied in the abstract model they will be satisfied in systems of any size. This allows us to solve the parameterised model checking problem in these cases.

We have implemented this technique in the probabilistic model checking toolkit `PSV-CA`, which we built on top of PRISM. We used our toolkit to verify properties in a number of scenarios both from swarm robotics and security applications.

There are a number of avenues for future work in this area. One possible research direction is developing more scalable techniques that would enable the verification of more realistic scenarios. Since the size of the abstract model here presented may grow exponentially with the number of states each agent may exhibit, this technique becomes intractable for larger systems. More efficient, tighter abstractions may scale more favourably. Another possible direction for further work is exploring a different choice of semantics. It is possible that by considering a more restricted version of either the models considered or of the specification language, a fully decidable version of the stochastic parameterised model checking problem could be obtained.

Acknowledgements

Alessio Lomuscio is supported by a Royal Academy of Engineering Chair in Emerging Technologies. Edoardo Pirovano is supported by an EPSRC-funded doctoral scholarship (grant number EP/N509486/1).

References

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

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

[3] E. Şahin, Swarm robotics: From sources of inspiration to domains of application, in: Proceedings of the International Conference on Swarm Robotics (SAB04), Vol. 3342 of Lecture Notes in Computer Science, Springer, 2005, pp. 10–20.

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

[5] W. Spears, D. Spears, R. Heil, W. Kerr, S. Hettiarachchi, An overview of physicomimetics, in: Proceedings of the 1st International Workshop on Simulation of Adaptive Behavior (SAB04), Vol. 3342 of Lecture Notes in Computer Science, Springer, 2004, pp. 84–97.

[6] C. Parrott, T. Dodd, J. Boxall, K. Horoshenkov, Simulation of the behavior of biologically-inspired swarm robots for the autonomous inspection of buried pipes, Tunnelling and Underground Space Technology 101 (2020) 103356.

[7] R. Meyer, A theory of structural stationarity in the *pi* -calculus, Acta Informatica 46 (2) (2009) 87–137.

[8] K. Lerman, A. Galstyan, Mathematical model of foraging in a group of robots, Auton. Robots 13 (2) (2002) 127–141.

[9] C. B. J. Katoen, Principles of Model Checking (Representation and Mind Series), The MIT Press, 2008.

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

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

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

[13] K. Apt, D. C. Kozen, Limits for automatic verification of finite-state concurrent systems, Information Processing Letters 22 (6) (1986) 307–309.

[14] E. Emerson, V. Kahlon, Model checking guarded protocols, in: Proceedings of the 14th International Symposium on Logic in Computer Science (LICS03), IEEE, 2003, pp. 361–370.

[15] B. Aminof, S. Jacobs, A. Khalimov, S. Rubin, Parameterized model checking of token-passing systems, in: Proceedings of the 15th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI14), Vol. 8318 of Lecture Notes in Computer Science, Springer, 2014, pp. 262–281.

[16] P. Kouvaros, A. Lomuscio, Parameterised verification for multi-agent systems, Artificial Intelligence 234 (2016) 152–189.

[17] P. Kouvaros, A. Lomuscio, Verifying emergent properties of swarms, in: Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI15), AAAI Press, 2015, pp. 1083–1089.

[18] J. Katoen, The probabilistic model checking landscape, in: Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS16), ACM, 2016, pp. 31–45.

[19] M. Kwiatkowska, G. Norman, D. Parker, PRISM 4.0: Verification of probabilistic real-time systems, in: Proceedings of the 23rd International Conference on Computer Aided Verification (CAV11), Vol. 6806 of Lecture Notes in Computer Science, Springer, 2011, pp. 585–591.

[20] C. Dehnert, S. Junges, J. Katoen, M. Volk, A storm is coming: A modern probabilistic model checker, in: Proceedings of the 29th International Conference on Computer Aided Verification (CAV17), Vol. 10427 of Lecture Notes in Computer Science, Springer, 2017, pp. 592–600.

[21] M. Kwiatkowska, G. Norman, D. Parker, Symbolic Systems Biology, Jones and Bartlett, 2010, Ch. Probabilistic Model Checking for Systems Biology, pp. 31–59.

[22] M. Duflot, M. Kwiatkowska, G. Norman, D. Parker, S. Peyronnet, C. Picaronny, J. Sproston, FMICS Handbook on Industrial Critical Systems, IEEE Computer Society Press, 2010, Ch. Practical Applications of Probabilistic Model Checking to Communication Protocols, pp. 133–150.

[23] M. Kwiatkowska, G. Norman, D. Parker, G. Santos, Automated verification of concurrent stochastic games, in: Proceedings of the 15th International Conference on Quantitative Evaluation of SysTems (QEST18), Vol. 11024 of Lecture Notes in Computer Science, Springer, 2018, pp. 223–239.

[24] G. Norman, D. Parker, J. Sproston, Model checking for probabilistic timed automata, Formal Methods in System Design 43 (2) (2013) 164–190.

[25] R. Hoffmann, M. Ireland, A. Miller, G. Norman, S. Veres, Autonomous agent behaviour modelled in PRISM - A case study, in: Proceedings of the 23rd International SPIN Symposium on Model Checking of Software (SPIN16), Vol. 9641 of Lecture Notes in Computer Science, Springer, 2016, pp. 104–110.

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

[27] R. Fagin, J. Halpern, Y. Moses, M. Vardi, Reasoning about Knowledge, MIT Press, 1995.

[28] A. Campo, M. Dorigo, Efficient multi-foraging in swarm robotics, in: Advances in Artificial Life, Vol. 4648 of Lecture Notes in Computer Science, Springer, 2007, pp. 696–705.

[29] Q. Zhu, H. Li, Z. Han, T. Basar, A stochastic game model for jamming in multi-channel cognitive radio systems, in: Proceedings of IEEE International Conference on Communications, (ICC10), IEEE, 2010, pp. 1–6.

[30] L. Zuck, A. Pnueli, Model checking and abstraction to the aid of parameterized systems (a survey), Computer Languages, Systems & Structures 30 (3) (2004) 139–169.

[31] P. Abdulla, B. Jonsson, On the existence of network invariants for verifying parameterized systems, in: Correct System Design, Vol. 1710 of Lecture Notes in Computer Science, Springer, 1999, pp. 180–197.

[32] E. Clarke, M. Talupur, T. Touili, H. Veith, Verification by network decomposition, in: Proceedings of the 15th International Conference on Concurrency Theory (CONCUR04), Vol. 3170 of Lecture Notes in Computer Science, Springer, 2004, pp. 276–291.

[33] E. Emerson, K. Namjoshi, Reasoning about rings, in: Proceedings of the 22nd Annual Sigact-Aigplan on Principles of Programming Languages (POPL95), Pearson Education, 1995, pp. 85–94.

[34] A. Kaiser, D. Kroening, T. Wahl, Dynamic cutoff detection in parameterized concurrent programs, in: Proceedings of the 22nd International Conference on Computer Aided Verification (CAV10), Vol. 6184 of Lecture Notes in Computer Science, Springer, 2010, pp. 645–659.

[35] P. Kouvaros, A. Lomuscio, 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, pp. 2013–2019.

[36] A. Lomuscio, H. Qu, F. Raimondi, MCMAS: A model checker for the verification of multi-agent systems, in: Proceedings of the 21th International Conference on Computer Aided Verification (CAV09), Vol. 5643 of Lecture Notes in Computer Science, Springer, 2009, pp. 682–688.

[37] P. Kouvaros, A. Lomuscio, Verifying fault-tolerance in parameterised multi-agent systems, in: Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI17), AAAI Press, 2017, pp. 288–294.

[38] P. Kouvaros, A. Lomuscio, E. Pirovano, 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), AAAI Press, 2018, pp. 324–330.

[39] P. Kouvaros, A. Lomuscio, Formal verification of opinion formation in swarms, in: Proceedings of the 15th International Conference on Autonomous Agents and Multi-Agent systems (AAMAS16), IFAAMAS Press, 2016, pp. 1200–1209.

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

[41] W. Reisig, Petri Nets. An Introduction, Vol. 4 of EACTS Monographs on Theoretical Computer Science, Springer, 1985.

[42] C. Rackoff, The covering and boundedness problems for vector addition systems, Theoretical Computer Science 6 (2) (1978) 223–231.

[43] J. Leroux, S. Schmitz, Demystifying reachability in vector addition systems, in: Proceedings of the 30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS15), IEEE Computer Society, 2015, pp. 56–67.

[44] L. Spalazzi, F. Spegni, Parameterized model checking of networks of timed automata with boolean guards, Theoretical Computer Science 813 (2020) 248–269.

[45] W. Penczek, A. Pólrola, Specification and model checking of temporal properties in time Petri nets and timed automata, in: Proceedings of the 25th International Conference on Applications and Theory of Petri Nets (ATPN04), Vol. 3099 of Lecture Notes in Computer Science, Springer, 2004, pp. 37–76.

[46] P. Abdulla, A. Nylén, Timed Petri Nets and BQOs, in: Proceedings of the 22nd International Conference on Applications and Theory of Petri Nets (ICATPN01), Vol. 2075 of Lecture Notes in Computer Science, Springer, 2001, pp. 53–70.

[47] L. Jacobsen, M. Jacobsen, M. Møller, J. Srba, Verification of timed-arc petri nets, in: Proceedings of the 37th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM11), Vol. 6543 of Lecture Notes in Computer Science, Springer, 2011, pp. 46–72.

[48] D. Angluin, J. Aspnes, Z. Diamadi, M. Fischer, R. Peralta, Computation in networks of passively mobile finite-state sensors, in: Proceedings of the 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC04), ACM, 2004, pp. 290–299.

[49] D. Angluin, J. Aspnes, D. Eisenstat, Stably computable predicates are semilinear, in: Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing (PODC06), ACM, 2006, pp. 292–299.

[50] S. Ginsburg, E. Spanier, Semigroups, presburger formulas, and languages, Pacific J. Math. 16 (2) (1966) 285–296.

[51] J. Esparza, P. Ganty, J. Leroux, R. Majumdar, Verification of population protocols, Acta Informatica 54 (2) (2017) 191–215.

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

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

[54] C. Baier, B. Haverkort, H. Hermanns, J. Katoen, Model-checking algorithms for continuous-time markov chains, IEEE Trans. Software Eng. 29 (6) (2003) 524–541.

[55] P. Gainer, C. Dixon, U. Hustadt, Probabilistic model checking of ant-based positionless swarming, in: Proceedings of the 17th Annual Conference Towards Autonomous Robotics (TAROS16), Vol. 9716 of Lecture Notes in Computer Science, Springer, 2016, pp. 127–138.

[56] T. Chen, M. Kwiatkowska, D. Parker, A. Simaitis, Verifying team formation protocols with probabilistic model checking, in: Proceedings of the 12th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA11), Vol. 6814 of Lecture Notes in Computer Science, Springer, 2011, pp. 190–207.

[57] R. Giaquinta, R. Hoffmann, M. Ireland, A. Miller, G. Norman, Strategy synthesis for autonomous agents using PRISM, in: Proceedings of the 10th NASA Formal Methods Symposium (NFM18), Vol. 10811 of Lecture Notes in Computer Science, Springer, 2018, pp. 220–236.

[58] D. Graham, M. Calder, A. Miller, An inductive technique for parameterised model checking of degenerative distributed randomised protocols, Electronic Notes in Theoretical Computer Science 250 (1) (2009) 87–103.

[59] N. Bertrand, P. Fournier, Parameterized verification of many identical probabilistic timed processes, in: Proceedings of the 33rd Foundations of Software Technology and Theoretical Computer Science conference (FSTTCS13), Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013, pp. 501–513.

[60] B. Aminof, A. Murano, S. Rubin, F. Zuleger, Automatic verification of multi-agent systems in parameterised grid-environments, in: Proceedings of the 15th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS16), IFAAMAS Press, 2016, pp. 1190–1199.

[61] F. Belardinelli, D. Grossi, A. Lomuscio, Finite abstractions for the verification of epistemic properties in open multi-agent systems, in: Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI15), AAAI Press, 2015, pp. 854–860.

[62] P. Kouvaros, A. Lomuscio, E. Pirovano, H. Punchihewa, Formal verification of open multi-agent systems, in: Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS19), IFAAMAS Press, 2019, pp. 179–187.

[63] R. De Masellis, V. Goranko, Logic-based specification and verification of homogeneous dynamic multi-agent systems, Auton. Agents Multi Agent Syst. 34 (2) (2020) 34.

[64] A. Lomuscio, E. Pirovano, 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), AAAI Press, 2018, pp. 403–409.

[65] A. Lomuscio, E. Pirovano, A counter abstraction technique for the verification of probabilistic swarm systems, in: Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS19), IFAAMAS Press, 2019, pp. 161–169.

[66] A. Lomuscio, E. Pirovano, Parameterised verification of strategic properties in probabilistic multi-agent systems, in: Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS20), IFAAMAS Press, 2020, pp. 762–770.

[67] A. Lomuscio, E. Pirovano, Verifying fault-tolerance in probabilistic swarm systems, in: Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI-PRICAI20), ijcai.org, 2020, pp. 325–331.

[68] J. Kemeny, J. L. Snell, A. Knapp, Denumerable Markov Chains, 2nd Edition, Graduate Texts in Mathematics, Springer, 1976.

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

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

[71] E. Ferrante, M. Brambilla, M. Birattari, M. Dorigo, Socially-Mediated Negotiation for Obstacle Avoidance in Collective Transport, Vol. 83 of Springer Tracts in Advanced Robotics (STAR), Springer, 2013, pp. 571–583.

[72] PSV-CA, Probabilistic Swarm Verifier by Counter Abstraction `https://www.dropbox.com/s/bc5xpv4ngq3thh0?dl=1` (2020).

[73] W. Liu, A. Winfield, Modelling and optimisation of adaptive foraging in swarm robotic systems, The International Journal of Robotics Research 29 (14) (2010) 1743–1760.