Verifying Fault-Tolerance in Probabilistic Swarm Systems

Alessio Lomuscio and **Edoardo Pirovano**
Department of Computing, Imperial College London
{a.lomuscio, e.pirovano17}@imperial.ac.uk

Abstract

We present a method for reasoning about fault-tolerance in unbounded robotic swarms. We introduce a novel semantics that accounts for the probabilistic nature of both the swarm and possible malfunctions, as well as the unbounded nature of swarm systems. We define and interpret a variant of probabilistic linear-time temporal logic on the resulting executions, including those arising from faulty behaviour by some of the agents in the swarm. We specify the decision problem of parameterised fault-tolerance, which concerns determining whether a probabilistic specification holds under possibly faulty behaviour. We outline a verification procedure that we implement and use to study a foraging protocol from swarm robotics, and report the experimental results obtained.

1 Introduction

Swarm robotics is commonly seen as the key technology for many forthcoming robotic applications from logistics, search and rescue, manufacturing and maintenance and beyond [Şahin, 2005]. While the technology has shown much promise, a key difficulty in adopting swarm robotics in either safety-critical or humans in the loop applications is to give guarantees on behaviour of the swarm before deployment. A key difficulty lies in the fact that swarms are often realised by *arbitrarily large* collections of robots. While methods such as model checking [Baier and Katoen, 2008] can be used to study particular ground instantiations [Dixon *et al.*, 2012], they cannot be adopted to reason about a system of which the number of components is unknown. Indeed, this problem is known to be undecidable [Apt and Kozen, 1986] in general.

The method of parameterised model checking [Bloem *et al.*, 2015] has been adapted for swarm systems specified by a variety of AI-inspired specifications [Kouvaros and Lomuscio, 2016] and recently extended to capture open systems in which agents may join and leave at run-time [Kouvaros *et al.*, 2019] and systems incorporating probabilities [Lomuscio and Pirovano, 2019].

A key issue to establish before deployment is not just the correctness of the system, but the extent to which faults occurring at runtime can compromise the system safety. The method of safety analysis via fault injection and model checking is well established in dependable systems [Bozzano and Villafiorita, 2007; Bozzano *et al.*, 2017]. Via safety-analysis the engineer can establish the consequences of particular faults in the system and identify particular weaknesses or critical faults which may compromise the system's safety. The method also allows engineers to establish the resilience of the system in adverse circumstances, contributing to the construction of safety arguments in certification.

Safety analysis via fault injection [Ezekiel and Lomuscio, 2017] has been used in AI to establish the resilience of autonomous systems [Ezekiel *et al.*, 2011] and extensions have been explored in a swarm setting [Kouvaros and Lomuscio, 2017]. However, no method to date has addressed fault injection for probabilistic swarms. Since most swarm protocols are probabilistic in nature, this makes it impossible to perform safety analysis on unbounded swarm systems. This paper intends to rectify this shortcoming.

Related Work. Previous work has addressed the verification of fault-tolerance in multi-agent systems [Ezekiel and Lomuscio, 2017; Ezekiel *et al.*, 2011]. In turn this builds on earlier work on safety analysis [Bozzano and Villafiorita, 2007] that has been successfully adopted in avionics [Bozzano *et al.*, 2017]. While this work is relevant, the present contribution deals with unbounded swarms, where the number of agents is not known at design time, and, differently, from these, exhibit probabilistic behaviour.

A considerable body of research has been devoted to parametrised verification of non-probabilistic systems without any fault injection [Aminof *et al.*, 2018; Aminof *et al.*, 2016; Bloem *et al.*, 2015]. The work presented here is distinct in that the systems we consider are probabilistic and we perform safety analysis by injecting faults into the templates.

Fault-tolerance in systems with an unbounded number of agents has been considered in a number of different semantics, including some geared to network protocols [John *et al.*, 2013] and others catering for swarm systems [Kouvaros and Lomuscio, 2017; Kouvaros *et al.*, 2018]. Note, however, that none of these semantics can express stochastic behaviour nor probabilistic specifications as we do here.

Closest to the work here pursued is a line of work in which probabilistic systems with a possibly unbounded number of agents are considered [Lomuscio and Pirovano, 2019; Lomuscio and Pirovano, 2020]. Note, however, that this work does not consider the mechanism of fault injection typical in safety analysis as we do here. Other work in the parameterised verification of probabilistic network protocols [Graham, 2008; Fournier, 2015] also fails to address faults.

In summary, the work presented here is distinct from the existing literature in that, to the best our knowledge, it addresses the safety analysis via fault injection for unbounded swarm systems that are governed by stochastic behaviour.

2 Background

In this section we introduce some background on probabilistic model checking of swarm systems, along with the notation that we will use throughout the paper. We follow [Lomuscio and Pirovano, 2019] in notation.

Discrete Time Markov Chains. We briefly summarise *discrete time Markov chains* [Kwiatkowska *et al.*, 2007; Baier and Katoen, 2008; Kemeny *et al.*, 1976].

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

A *path* in a DTMC is a sequence of states $s_0 s_1 s_2 \ldots$ such that for every $i \in \mathbb{N}$ it is the case that $t(s_i, s_{i+1}) > 0$. We use $FPath_{\mathcal{D}}$ and $IPath_{\mathcal{D}}$ respectively, to denote the set of all finite and infinite paths starting from the initial state ι. For a finite path we define its probability by $\mathbf{P}_{\mathcal{D}}(s_0 \ldots s_n) \triangleq \prod_{i=0}^{n-1} t(s_i, s_{i+1})$. Following [Kemeny *et al.*, 1976], this can be extended to define a probability on infinite paths. Note that the probability of an infinite path occurring is uniquely defined by the probabilities of its finite prefixes occurring. For a measurable set of paths $X \subseteq FPath_{\mathcal{D}}$ or $X \subseteq IPath_{\mathcal{D}}$ we define $\mathbf{P}_{\mathcal{D}}(X) \triangleq \sum_{\rho \in X} \mathbf{P}_{\mathcal{D}}(\rho)$.

Markov Decision Processes. We mostly follow the notation used in [Forejt *et al.*, 2011] and refer to [Baier and Katoen, 2008; Puterman, 1994] for more details.

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 $s_0 a_0 s_1 a_2 \ldots$ 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 MDP, we need a way to resolve the inherent non-determinism. This is captured by the notion of an adversary.

Definition 3 (Adversary). *Given an MDP $\mathcal{M} = \langle S, \iota, A, P, t, L\rangle$ an adversary for $\mathcal{M}$ is a function $\sigma : FPath_{\mathcal{M}} \times A \to [0, 1]$ such that for any finite path $\rho \in FPath_{\mathcal{M}}$, we have $\sigma(\rho, a) > 0$ only if $a \in P(last(\rho))$ and $\sum_{a \in A} \sigma(\rho, a) = 1$.*

We denote by $Adv_{\mathcal{M}}$ the set of all adversaries for $\mathcal{M}$. Note that when a choice of adversary σ is fixed, the non-determinism in the MDP $\mathcal{M}$ is resolved, and we can describe the behaviour of the resulting system by a DTMC. We denote this DTMC by $\mathcal{M}_\sigma$ (see, e.g., [Forejt *et al.*, 2011] for details).

Probabilistic Swarm Systems. We now summarise the semantics of probabilistic parameterised interleaved interpreted systems (PPIISs). Here we will also follow the notation used in [Lomuscio and Pirovano, 2019].

A PPIIS is composed of an *agent template*, which defines the behaviour of one of the agents and an *environment* which gives the behaviour of the rest of the system.

Definition 4 (Probabilistic agent template). *A probabilistic agent template is a tuple $T = \langle S, \iota, Act, P, t\rangle$ where:*

- *The set S is a set of agent local states.*
- *$\iota \in S$ is a distinguished initial state.*
- *$Act = A \cup AE \cup GS$ is the non-empty set of actions that can be performed by the agent. These may either be asynchronous actions, agent-environment actions or global-synchronous actions. Note the sets are disjoint, so each action only has one type. Each type of action gives a different communication pattern between the agents according Definition 6.*
- *The agent's protocol function $P : S \to \mathcal{P}(Act)$ defines which actions are enabled at a given state.*
- *The agent's transition function $t : S \times Act \times S \to [0, 1]$ describes the evolution of the agent's state: given a local state s, an action a, and a local state s', t returns the probability that upon performing action a in state s the agent will transition to state s'. Notice we require that for every $l \in S$, $a \in P(l)$ we have $\sum_{l' \in S} t(l, a, l') = 1$.*

The agent template can be seen as an MDP with additional labels on the actions for synchronisation purposes with other agents and the environment. The environment $E = \langle S_E, \iota_E, Act_E, P_E, t_E\rangle$ is similarly defined (see [Lomuscio and Pirovano, 2019] for details).

Definition 5 (PPIIS). *A probabilistic parameterised interleaved interpreted system (PPIIS) is a tuple $\mathcal{S} = \langle T, E, L\rangle$, where T is a probabilistic agent template, E is an environment and $L : S \times S_E \to \mathcal{P}(AP)$ is a labelling function for a set of atomic propositions AP.*

A PPIIS is an abstract description capturing an unbounded collection of concrete MDPs that are obtained by fixing a number of agents $n \in \mathbb{Z}^+$. We denote the agents by $Agt_n \triangleq \{1, \ldots, n\}$. The instantiated, or concrete, MDPs have states, which we call *global states*, of the form $g = \langle s_1, \ldots, s_n, s_e\rangle$ where $s_1, \ldots, s_n \in S$ and $s_e \in S_E$. We denote by S_n the set of all such states, and use $g.i$ to denote the state of agent i, and $g.E$ to denote the state of the environment. The set of global actions is given by $Act_n \triangleq GS \cup A_E \cup ((A \cup AE) \times Agt_n)$.

Definition 6 (Global protocol)**.** *The global protocol $P_n : S_n \to \mathcal{P}(Act_n)$ is defined by $a \in P_n(g)$ if and only if one of the following holds:*

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

Thus, global-synchronous actions must be enabled for all agents and the environment, asynchronous ones just for the agent or environment performing them, and agent-environment ones for the environment and agent performing them. The restrictions above are intended to limit the undecidability of the resulting verification problem; they were originally introduced in [Kouvaros and Lomuscio, 2016] to study non-probabilistic systems.

We now define the global transition function, which describes the outcome of performing a global action.

Definition 7 (Global transition function)**.** *The global transition function $t_n : S_n \times Act_n \times S_n \to [0,1]$ is defined by $t_n(g, a, g')$*

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

We can now define the MDP $\mathcal{S}(n)$ encoding the global behaviour of n agents and an environment.

Definition 8 (Concrete system)**.** *Given a PPIIS $\mathcal{S}$ and an $n \in \mathbb{Z}^+$, the concrete system of n agents is defined by $\mathcal{S}(n) = \langle S_n, \iota_n, Act_n, P_n, t_n, L_n \rangle$, where $\iota_n = \langle \iota, \ldots, \iota, \iota_E \rangle$, the labelling function $L_n : S_n \to \mathcal{P}(AP \times Agt_n)$ is defined by $L_n(g) \triangleq \{(p, i) \in AP \times Agt_n : p \in L(g.i, g.E)\}$, and the other components are defined as in Definitions 6 and 7.*

In [Lomuscio and Pirovano, 2019], it is observed that the concrete system is always a valid MDP. The properties we wish to check on this family of MDPs are expressed in a variant of PLTL [Forejt *et al.*, 2011], which we define below.

Definition 9 (PLTL)**.** *For $p \in AP$ and $i \in \mathbb{Z}^+$, the probabilistic LTL logic is defined by the following BNF:*

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

We say a formula is m-indexed if it refers to agents with index at most m (i.e., any atomic proposition is of the form (p, i), where $i \le m$).

Definition 10 (Satisfaction)**.** *Given a concrete swarm system $\mathcal{S}(n)$ and ϕ an m-indexed formula, where $n \ge m$, the satisfaction of ϕ on $\mathcal{S}(n)$ is inductively defined as follows:*

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

Satisfaction for path formulae is defined as usual in LTL.

The parameterised model checking problem is concerned with checking whether a specification holds in concrete systems built on arbitrarily large number of agents.

Definition 11 (Parameterised Model Checking)**.** *Given a PPIIS $\mathcal{S}$ and an m-indexed PLTL formula ϕ, the parameterised model checking problem involves establishing whether it is the case that $\mathcal{S}(n) \models \phi$ for all $n \ge m$. We write $\mathcal{S} \models \phi$ if this is the case.*

Notice that this problem is a probabilistic extension of a problem that is known to be undecidable [Apt and Kozen, 1986]. Thus, it is also undecidable in general. However, a partial decision procedure was put forward for it in [Lomuscio and Pirovano, 2019].

3 Swarm Safety Analysis via Fault Injection

Parameterised model checking can be used to establish the correctness of, or establish bugs in a swarm protocol [Lomuscio and Pirovano, 2018]. However, certifying the absence of bugs is often not sufficient to guarantee safety of a swarm protocol nor to entirely understand the resulting run-time behaviour. For this, it is also important to establish the extent to which runtime malfunctions, or faults, affect the overall performance of the swarm. Some faults may break some overall swarm specifications; in other cases, the swarm may actually be resilient to particular malfunctions. Understanding these weaknesses or strengths can inform designers and deployers.

To conduct a formal analysis of these aspects we construct a faulty PPIIS, $\mathcal{S}^\chi = \langle T^\chi, E, L^\chi \rangle$, from a non-faulty one $\mathcal{S} = \langle T, E, L \rangle$ as follows. To model faults, we assume that the local states L of an agent are defined by a set of integer, Boolean, and enumerate (over a domain $\mathcal{E}$) variables $VAR = BVar \cup IVar \cup EVar$, i.e. $L = (b : BVar \to \{\bot, \top\}) \times (i : IVar \to \mathbb{Z}) \times (e : EVar \to \mathcal{E})$. We denote by $\mathcal{F}$ the set of all possible faults. Various studies have been conducted in safety analysis to identify the faults normally of interest [Bozzano and Villafiorita, 2007]. We here consider the most widely used faults by assuming that $\mathcal{F}$ contains:

- For every $x \in BVar$, a fault $invert(x)$ that inverts the value of x and a fault $setB(x, k)$ (where $k \in \{\bot, \top\}$) that sets the value of x to k.
- For every $y \in IVar$, a fault $up(y)$ that increments the value of y, a fault $down(y)$ that decrements the value of y, and a fault $setI(y, k)$ (where $k \in \mathbb{Z}$) that sets the value of y to k.

- For every $z \in EVar$, a fault $setE(z, v)$ (where $v \in \mathcal{E}$) that sets the value of z to v.

More formally, $\mathcal{F}$ is given by:

$$\begin{aligned}\mathcal{F} \triangleq &\{invert(x), setB(x,k) : x \in BVar, k \in \{\top, \bot\}\}\\ &\cup \{up(y), down(y), setI(y,k) : y \in IVar, k \in \mathbb{Z}\}\\ &\cup \{setE(z,v) : z \in EVar, v \in \mathcal{E}\}\end{aligned}$$

Given a fault $f \in \mathcal{F}$ and a state $s \in L$, we will denote by $(s)_f$ the result of applying f to s. Formally:

$$((b,i,e))_f = \begin{cases}(b_{x\mapsto\neg b(x)}, i, e) & \text{if } f = invert(x)\\ (b_{x\mapsto k}, i, e) & \text{if } f = setB(x,k)\\ (b, i_{y\mapsto i(y)+1}, e) & \text{if } f = up(y)\\ (b, i_{y\mapsto i(y)-1}, e) & \text{if } f = down(y)\\ (b, i_{y\mapsto k}, e) & \text{if } f = setI(y,k)\\ (b, i, e_{z\mapsto v}) & \text{if } f = setE(z,v)\end{cases}$$

where $g_{x\mapsto y}$ denotes the function obtained by replacing the value of g at x with y.

Having formalised the types of possible faults, we now define when they can occur and how likely they are. We encode this in the notion of a fault profile, which defines the probability of each fault occurring when performing a certain action from a certain state.

Definition 12 (Fault profile)**.** *A fault profile is a function $\chi : S \times Act \times (\mathcal{F} \cup \{\checkmark\}) \to [0,1]$. The expression $\chi(s,a,f) = p$ represents that when action a is performed from state s, there is a probability of p that fault f occurs. $\chi(s,a,\checkmark)$ gives the probability of a fault* **not** *occurring when performing action a from state s. We assume that for all $s \in S$ and $a \in Act$ we have $\sum_{f\in(F\cup\{\checkmark\})} \chi(s,a,f) = 1$.*

Note that the above restricts our swarms to having at most one fault at each time-step. We now define the behaviour of a faulty agent. This will augment the state of the agent with extra variables that allow us to write specifications involving whether the agent has exhibited faults.

Definition 13 (Faulty agent)**.** *Given an agent template $T = \langle S, \iota, Act, P, t\rangle$ and a fault profile $\chi : S \times Act \times (\mathcal{F} \cup \{\checkmark\}) \to [0,1]$, we define a faulty agent template $T^\chi = \langle S^\chi, \iota^\chi, Act, P^\chi, t^\chi\rangle$ as follows:*

- *$S^\chi = S \times \{\bot, \top\} \times \{\bot, \top\}$, where the first Boolean variable encodes whether the agent has ever exhibited a fault and the second whether the agent exhibited a fault in the previous transition.*
- *$\iota^\chi = (\iota, \bot, \bot)$ is a new initial state,*
- *$P^\chi : S^\chi \to \mathcal{P}(Act)$ is given by $P^\chi((s,f,i)) \triangleq P(s)$.*
- *$t^\chi : S^\chi \times Act \times S^\chi \to [0,1]$ is defined by:*

$$((s,f,i), a, (s',f',i')) \mapsto$$
$$\begin{cases} t(s,a,s')\chi(s,a,\checkmark) & \textit{if } f = f' \textit{ and } i' = \bot\\ \sum_{x\in\mathcal{F},\bar{s}\in S:(\bar{s})_x = s'} t(s,a,\bar{s})\chi(s,a,x) & \textit{if } f' = i' = \top\\ 0 & \textit{otherwise}\end{cases}$$

The definition above includes both non-faulty transitions (corresponding to $i' = \bot$) in which the value of f is not affected, and faulty transitions ($i' = \top$) in which f is set to $\top$. Note that transitions where $f = \top$ and $f' = \bot$ are not allowed; so once an agent has exhibited faulty behaviour, it is labelled as faulty for the rest of the run of the system. However, faulty agents may still carry out correct transitions.

Observe that faulty agents are well-defined agents.

Observation 1. *Let $\chi : S \times Act \times (\mathcal{F} \cup \{\checkmark\}) \to [0,1]$ be a fault profile and $T = \langle S, \iota, Act, P, t\rangle$ a non-faulty agent. Then, it is the case that for any $l = (s,f,i) \in S^\chi$ and $a \in Act$ we have $\sum_{l'\in S^\chi} t^\chi(l,a,l') = 1$.*

We can now define faulty swarm systems.

Definition 14 (Fully Faulty Swarm System)**.** *Given a fault profile $\chi : S \times Act \times (\mathcal{F} \cup \{\checkmark\}) \to [0,1]$ and a non-faulty PPIIS $\mathcal{S} = \langle T, E, L\rangle$, the (fully) faulty PPIIS $\mathcal{S}^\chi = \langle T^\chi, E, L^\chi\rangle$ is constructed by taking T^χ as in Definition 13, and $L^\chi : S^\chi \times S_E \to \mathcal{P}(AP \cup \{faulty, injected\})$ defined:*
$((s,f,i), s_E) \mapsto$

$$\begin{cases}\{faulty, injected\} \cup L(s,s_E) & \textit{if } i = \top\\ \{faulty\} \cup L(s,s_E) & \textit{if } i = \bot \textit{ and } f = \top\\ L(s,s_E) & \textit{otherwise}\end{cases}$$

Notice that the environment does not exhibit any faults. We add two additional atomic propositions to the language: *faulty* and *injected*, tracking whether an agent has ever exhibited faulty behaviour and whether it exhibited faulty behaviour at the previous time step, respectively. These atomic propositions allow us to express a number of specifications such as:

$$P^{max}_{\le 0.5}[G\neg(faulty, 0)]$$

which expresses that the probability of an agent ever exhibiting a fault does not exceed 0.5.

We can also express probabilistic variants of properties often considered in fault-tolerance literature. For instance:

$$P^{max}_{>0.9}[G((injected, 0) \to F\phi)] \qquad \text{(Recoverability)}$$

expresses that with high probability even if an agent exhibits a fault the swarm can still satisfy ϕ at some point in the future. This expresses a notion of resilience of the agents in the swarm w.r.t. the faults and the specification ϕ. Note that ϕ may depend on the state of other agents or the environment, and thus can express a property of the whole swarm system rather than just of agent 0.

Finally, the new atomic propositions allow us to limit specifications to agents which have not exhibited faulty behaviour. For instance:

$$P^{max}_{>0.9}[G(\neg(faulty, 0) \to \phi)]$$

expresses that with high probability whenever an agent is not faulty, the agent can satisfy the formula ϕ.

4 Model Checking Faulty PPIIS

In this section we define and solve the parameterised fault-tolerance problem which concerns assessing whether a probabilistic swarm is resilient w.r.t. possible faults given as above. Before doing so, we extend the notion of swarm systems of the previous section by introducing a probability that an agent may be faulty.

Definition 15 (Probabilistically Faulty Swarm System)**.** *Given a PIIS $\mathcal{S} = \langle T, E, L\rangle$, a fault profile $\chi : S \times Act \times (\mathcal{F} \cup \{\checkmark\}) \to [0,1]$ and a faultiness probability $p \in [0,1]$, we define the probabilistically faulty swarm system $\mathcal{S}_p^\chi = \langle T_p^\chi, E_p^\chi, L_p^\chi\rangle$ as follows.*

The faulty agent template $T_p^\chi = \langle S_p^\chi, \iota_p^\chi, Act_p^\chi, P_p^\chi, t_p^\chi\rangle$ is given by:

- *$S_p^\chi = S \cup S^\chi \cup \{\iota_p^\chi, \iota_f, \iota_f', \iota_n, \iota_n'\}$, defined by considering all the faulty and non-faulty states, as well as five fresh states used for initialisation below.*
- *ι_p^χ, the new initial state.*
- *$Act_p^\chi = Act \cup \{init, g, g'\}$ where $init \in A$ and $g, g' \in GS$, defined by introducing a fresh asynchronous action $init$ and fresh global-synchronous actions g and g', used for initialisation.*
- *$P_p^\chi : S_p^\chi \to \mathcal{P}(Act_p^\chi)$ is given by:*

$$s \mapsto \begin{cases} \{a\} & \textit{if } s = \iota_p^\chi \\ \{g\} & \textit{if } s \in \{\iota_f, \iota_n\} \\ \{g'\} & \textit{if } s \in \{\iota_f', \iota_n'\} \\ P(s) & \textit{if } s \in S \\ P^\chi(s) & \textit{if } s \in S^\chi \end{cases}$$

- *$t_p^\chi : S_p^\chi \times Act_p^\chi \times S_p^\chi \to [0,1]$ is given by:*

$$(s,a,s') \mapsto \begin{cases} p & \textit{if } s = \iota_p^\chi, a = init, s' = \iota_f \\ 1-p & \textit{if } s = \iota_p^\chi, a = init, s' = \iota_n \\ 1 & \textit{if } s = \iota_f, a = g, s' = \iota_f' \\ & \quad \textit{or } s = \iota_f', a = g', s' = \iota^\chi \\ & \quad \textit{or } s = \iota_n, a = g, s' = \iota_n' \\ & \quad \textit{or } s = \iota_n', a = g', s' = \iota \\ t(s,a,s') & \textit{if } s, s' \in S \\ t^\chi(s,a,s') & \textit{if } s, s' \in S^\chi \\ 0 & \textit{otherwise} \end{cases}$$

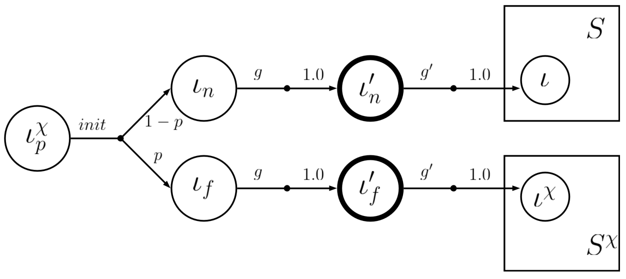
The environment $E_p^\chi = \langle S_{p,E}^\chi, \iota_{p,E}^\chi, Act_{p,E}^\chi, P_{p,E}^\chi, t_{p,E}^\chi\rangle$ is defined by:

- *$S_{p,E}^\chi = S_E \cup \{\iota_{p,E}^\chi, \iota'\}$, defined by adding two states to S_E used for initialisation.*
- *$\iota_{p,E}^\chi$, the new initial state.*
- *$Act_{p,E}^\chi = Act_E \cup \{g, g'\}$ where $g, g' \in GS$, defined by adding two fresh global-synchronous actions.*
- *$P_{p,E}^\chi : S_{p,E}^\chi \to \mathcal{P}(Act_{p,E}^\chi)$ is given by:*

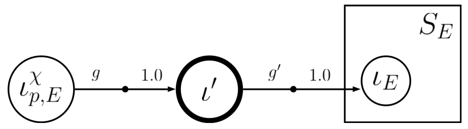
$$s \mapsto \begin{cases} \{g\} & \textit{if } s = \iota_{p,E}^\chi \\ \{g'\} & \textit{if } s = \iota' \\ P_E(s) & \textit{if } s \in S_E \end{cases}$$

- *$t_{p,E}^\chi : S_{p,E}^\chi \times Act_{p,E}^\chi \times S_{p,E}^\chi \to [0,1]$ is given by:*

$$(s,a,s') \mapsto \begin{cases} 1 & \textit{if } s = \iota_{p,E}^\chi, a = g, s' = \iota' \\ & \quad \textit{or } s = \iota', a = g', s' = \iota_E \\ t_E(s,a,s') & \textit{if } s, s' \in S_E \\ 0 & \textit{otherwise} \end{cases}$$



(a) The agent template.



(b) The environment.

Figure 1: The PPIIS for a probabilistically faulty swarm system. The bold states represent ones where the atomic proposition $starting$ holds. The g and g' actions are global-synchronous ones, whilst the $init$ action is asynchronous.

The valuation function $L_p^\chi : S_p^\chi \times S_{p,E}^\chi \to \mathcal{P}(AP \cup \{starting, faulty, injected\})$ is defined by:

$$(s, s_E) \mapsto \begin{cases} L^\chi((s,s_E)) & \textit{if } s \in S^\chi, s_E \in S_E \\ L((s,s_E)) & \textit{if } s \in S, s_E \in S_E \\ \{starting\} & \textit{if } s_E = \iota' \\ \emptyset & \textit{otherwise} \end{cases}$$

Intuitively, the original swarm is extended with an initialisation phase where each agent asynchronously uses the $init$ action to determine whether it will be faulty with probability p. Following this, the global-synchronous action g takes the system to a state where our new atomic proposition $starting$ holds. Finally, the global-synchronous action g' starts the system. This initialisation process is depicted in Figure 1. Note that the definition above results in a valid PPIIS.

Observation 2. *Consider a PIIS $\mathcal{S} = \langle T, E, L\rangle$, a fault profile $\chi : S \times Act \times (\mathcal{F} \cup \{\checkmark\}) \to [0,1]$ and a faultiness probability $p \in [0,1]$. Then, we have that $\sum_{l' \in S_p^\chi} t_p^\chi(l,a,l') = 1$ for all $l \in S_p^\chi$ and $a \in Act_p^\chi$. We also have $\sum_{l_E' \in S_{p,E}^\chi} t_{p,E}^\chi(l_E, a_E, l_E') = 1$ for all $l_E \in S_{p,E}^\chi$ and $a_E \in Act_{p,E}^\chi$.*

We are interested in assessing whether an unbounded probabilistic swarm is resilient w.r.t. a specification when subjected to probabilistically faulty agents. To do so, we formulate the following decision problem.

Definition 16 (PFTP)**.** *Given a PIIS $\mathcal{S}$, a PLTL formula $\phi = P_{\bowtie x}^{max/min}[\psi]$, a fault profile χ, and a $p \in [0,1]$, the parameterised fault-tolerance problem (PFTP) concerns answering the parameterised model checking problem (Definition 11) for $\mathcal{S}_p^\chi \models P_{\bowtie x}^{max/min}[G((starting, 0) \to X\psi)]$. If this holds, we write $\mathcal{S} \models_p^\chi \phi$.*

Definition 16 allows us to recast the PFTP via a simpler parameterised model checking query for the amended swarm system under a revised specification.

Algorithm 1 PFTP Decision Procedure

```
Input: A swarm system $\mathcal{S}$, PLTL formula $\phi = P^{max/min}_{\bowtie x}[\psi]$, fault profile $\chi$, and a $p \in [0,1]$
Output: Whether or not $\mathcal{S} \models^{\chi}_{p} \phi$
1: Construct $\mathcal{S}^{\chi}_{p}$ via Definition 15
2: Return $S^{\chi}_{p} \models P^{max/min}_{\bowtie x}[G((starting, 0) \rightarrow X\psi)]$
```

Given the above, we can introduce a simple procedure for checking the PFTP. This is presented in Algorithm 1. Note that the decision problem of line 2 can be resolved with tools for verification of probabilistic swarms [Lomuscio and Pirovano, 2019]. Note also that the procedure is incomplete, as the query on line 2 may not terminate. As we will see in the next section a number of concrete swarm protocols can effectively be checked irrespective of the theoretical undecidability of the underlying problem.

5 Implementation and Evaluation

We implemented the method described in the previous sections in a Java toolkit called `PSV-F` (**P**robabilistic **S**warm **V**erifier for **F**aulty systems), built on top of `PSV-CA` [Lomuscio and Pirovano, 2019], which in turn uses PRISM 4.0 [Kwiatkowska *et al.*, 2011] as its underlying probabilistic model checker. The source code for this and the model we consider below are released as open-source [PSV-F, 2020].

Like `PSV-CA`, our toolkit takes in a swarm file describing the behaviour of agents and the environment, and a file describing the specifications to check. In addition, `PSV-F` takes as input a file containing the list of faults that may occur in the system along with their relative conditions (actions being performed, states of the agent, and fault probabilities).

Upon invocation, `PSV-F` incorporates the faults into the model and invokes `PSV-CA` to verify the desired specification. Note that since the underlying decision procedure is partial, this call in principle may not receive an answer.

To evaluate the usefulness of our tool we consider the swarm foraging protocol [Campo and Dorigo, 2007; Liu and Winfield, 2010] that has previously been verified in a probabilistic setting without any faults in [Lomuscio and Pirovano, 2019]. We extend this analysis to introduce a fault.

The protocol has agents in four possible states: resting in a nest, searching for food, reaching some food, and returning to the nest with food. Resting robots may decide with probability 0.5 to begin searching for food. Robots searching for food may find some with probability 0.3. After two time-steps of failed searching, the robot goes back to resting. If a robot locates food, it moves towards it, picks it up, brings it back to the nest, then goes back to resting.

We inject one fault into the system encoding the fact that whenever a robot tries to move and is carrying food, it may drop the food with some probability p_f. If it does this, it will return to the nest with no food. As in the original analysis we check the specification pattern $P^{\max}_{\leq p}[F^{<k}\text{deposited}_2]$, where deposited_2 is an atomic proposition that holds if at least two units of food have been deposited in the nest and k is a parameter indicating the number of steps. The specification bounds

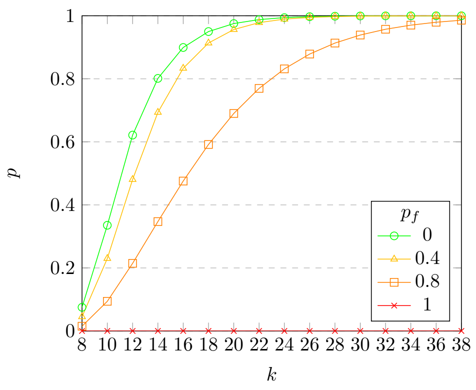


Figure 2: For different fault probabilities p_f and time-steps k, the maximum value of p for which $P^{\max}_{\leq p}[F^{<k}\text{deposited}_2]$ holds.

the maximum probability p with which the agents can deposit at least two units of food within k time-steps.

Intuitively the probability p depends on the fault probability p_f. By using `PSV-F` we can ascertain this relation precisely; importantly this analysis holds for any number of agents in the system as the analysis is based on parameterised model checking. We conducted this analysis and recorded in Figure 2 the maximum p for which the property holds for different values of k and p_f. The abstraction that `PSV-CA` generated had 2×10^6 states and 46×10^6 transitions. Injecting the faults was instantaneous, constructing the model took approximately 400 seconds and the specification was checked in approximately 5 seconds.

By analysing the results obtained, we deduce that the higher the probability of the fault, the lower the probability of success of the protocol. However, even systems with faulty agents can achieve the objective with high probability as long as sufficient time-steps are considered. Note also that, as expected, if $p_f = 1$ (i.e., agents always drop the food they are holding upon moving), then the goal is never achieved. The analysis above allows us to conclude that the foraging protocol displays a degree of resilience against the fault considered. This, however, decreases when the probability of faults increases. Note that, while these results are in line with our intuition, they could not previously be obtained formally.

6 Conclusions

We have introduced a method for injecting faults into probabilistic swarm systems and presented an implementation built from an existing toolkit for verifying such systems. To the best of our knowledge, this gives the first method and toolkit that enables the assessment of resilience to faults of swarm systems that are unbounded in size. We used these to assess the resilience of a foraging protocol; other protocols are likely to be analysable in a similar way. Another possible extension we have not pursued here is to consider more than one fault occurring at each time-step. We leave this for future work.

Acknowledgements

Alessio Lomuscio is supported by a Royal Academy of Engineering Chair in Emerging Technologies.

References

[Aminof *et al.*, 2016] B. Aminof, A. Murano, S. Rubin, and F. Zuleger. Automatic verification of multi-agent systems in parameterised grid-environments. In *Proceedings of AAMAS16*, pages 1190–1199. IFAAMAS Press, 2016.

[Aminof *et al.*, 2018] B. Aminof, S. Rubin, I. Stoilkovska, J. Widder, and F. Zuleger. Parameterized model checking of synchronous distributed algorithms by abstraction. In *Proceedings of VMCAI18*, pages 1–24. Springer, 2018.

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

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

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

[Bozzano and Villafiorita, 2007] M. Bozzano and A. Villafiorita. The FSAP/NuSMV-SA safety analysis platform. *Software Tools for Technology Transfer*, 9(1):5–24, 2007.

[Bozzano *et al.*, 2017] M. Bozzano, H. Bruintjes, A. Cimatti, J.-P. Katoen, T. Noll, and S. Tonetta. *Formal Methods for Aerospace Systems*, pages 133–159. Springer, 2017.

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

[Dixon *et al.*, 2012] C. Dixon, A. Winfield, M. Fisher, and C. Zeng. Towards temporal verification of swarm robotic systems. *Robotics and Autonomous Systems*, 60(11):1429–1441, 2012.

[Ezekiel and Lomuscio, 2017] J. Ezekiel and A. Lomuscio. Combining fault injection and model checking to verify fault tolerance, recoverability, and diagnosability in multi-agent systems. *Information and Computation*, 254(2):167–194, 2017.

[Ezekiel *et al.*, 2011] J. Ezekiel, A. Lomuscio, L. Molnar, and S. Veres. Verifying fault tolerance and self-diagnosability of an autonomous underwater vehicle. In *Proceedings of IJCAI11*, pages 1659–1664. AAAI Press, 2011.

[Forejt *et al.*, 2011] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker. Automated verification techniques for probabilistic systems. In *Proceedings of SFM11*, pages 53–113. Springer, 2011.

[Fournier, 2015] P. Fournier. *Parameterized verification of networks of many identical processes*. PhD thesis, Université de Rennes 1, 2015.

[Graham, 2008] D. Graham. *Parameterised verification of randomised distributed systems using state-based models*. PhD thesis, University of Glasgow, 2008.

[John *et al.*, 2013] A. John, I. Konnov, U. Schmid, H. Veith, and J. Widder. Parameterized model checking of fault-tolerant distributed algorithms by abstraction. In *Proceedings of FMCAD13*, pages 201–209. IEEE, 2013.

[Kemeny *et al.*, 1976] J. G. Kemeny, J. Laurie Snell, and A. W. Knapp. *Denumerable Markov Chains*. Graduate Texts in Mathematics. Springer, 1976.

[Kouvaros and Lomuscio, 2016] P. Kouvaros and A. Lomuscio. Parameterised verification for multi-agent systems. *Artificial Intelligence*, 234:152–189, 2016.

[Kouvaros and Lomuscio, 2017] P. Kouvaros and A. Lomuscio. Verifying fault-tolerance in parameterised multi-agent systems. In *Proceedings of IJCAI17*, pages 288–294. AAAI Press, 2017.

[Kouvaros *et al.*, 2018] P. Kouvaros, A. Lomuscio, and E. Pirovano. Symbolic synthesis of fault-tolerance ratios in parameterised multi-agent systems. In *Proceedings of IJCAI-ECAI18*, pages 324–330. AAAI Press, 2018.

[Kouvaros *et al.*, 2019] P. Kouvaros, A. Lomuscio, E. Pirovano, and H. Punchihewa. Formal verification of open multi-agent systems. In *Proceedings of AAMAS19*, pages 179–187. IFAAMAS Press, 2019.

[Kwiatkowska *et al.*, 2007] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In *Proceedings of SFM11*, pages 220–270. Springer, 2007.

[Kwiatkowska *et al.*, 2011] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proceedings of CAV11*, pages 585–591. Springer, 2011.

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

[Lomuscio and Pirovano, 2018] A. Lomuscio and E. Pirovano. Verifying emergence of bounded time properties in probabilistic swarm systems. In *Proceedings of IJCAI-ECAI18*, pages 403–409. AAAI Press, 2018.

[Lomuscio and Pirovano, 2019] A. Lomuscio and E. Pirovano. A counter abstraction technique for the verification of probabilistic swarm systems. In *Proceedings of AAMAS19*, pages 161–169. IFAAMAS Press, 2019.

[Lomuscio and Pirovano, 2020] A. Lomuscio and E. Pirovano. Parameterised verification of strategic properties in probabilistic multi-agent systems. In *Proceedings of AAMAS20*. IFAAMAS Press, 2020. To Appear.

[PSV-F, 2020] PSV-F. Probabilistic Swarm Verifier for Faulty systems http://vas.doc.ic.ac.uk/software/psv-f/, 2020.

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

[Şahin, 2005] E. Şahin. Swarm robotics: From sources of inspiration to domains of application. In *Proceedings of SAB04*, pages 10–20. Springer, 2005.