Verifying Fault-tolerance in Parameterised Multi-Agent Systems

Panagiotis Kouvaros
Department of Computing
Imperial College London, UK
University of Naples "Federico II", Italy
p.kouvaros@imperial.ac.uk

Alessio Lomuscio
Department of Computing
Imperial College London, UK
a.lomuscio@imperial.ac.uk

Abstract

We develop a technique to evaluate the fault-tolerance of a multi-agent system whose number of agents is unknown at design time. We present a method for injecting a variety of non-ideal behaviours, or faults, studied in the safety-analysis literature into the abstract agent templates that are used to generate an unbounded family of multi-agent systems with different sizes. We define the parameterised fault-tolerance problem as the decision problem of establishing whether any concrete system, in which the ratio of faulty versus non-faulty agents is under a given threshold, satisfies a given temporal-epistemic specification. We put forward a sound and complete technique for solving the problem for the semantical set-up considered. We present an implementation and a case study identifying the threshold under which the alpha swarm aggregation algorithm is robust to faults against its temporal-epistemic specifications.

1 Introduction

Over the past decade considerable progress has been made in developing techniques to verify that multi-agent systems (MAS) behave as intended [Bordini *et al.*, 2006; 2003]. In particular, symbolic model checking [Meyden and Su, 2004; Raimondi and Lomuscio, 2005], bounded and unbounded model checking [Kacprzak *et al.*, 2004] and abstraction [Belardinelli *et al.*, 2016] have proven useful to verify MAS against temporal-epistemic specifications. Some of these methods have resulted in push-button verification engines such as McK [Gammie and van der Meyden, 2004], Verics [Kacprzak *et al.*, 2008] and MCMAS [Lomuscio *et al.*, 2017].

One limitation of all these techniques is that they address the verification of MAS consisting of a number of agents fixed at design time. However, a number of protocols employed in the MAS domain, e.g., auctions, search and rescue protocols for robots, etc., are employed on the grounds they are assumed to be correct with respect to some specification *irrespective of how many agents populate the MAS*. Parameterised model checking (PMC) has recently been put forward as a technique to address this [Kouvaros and Lomuscio, 2016b]. PMC enables the engineer to study protocols, rather than specific concrete systems, where any number of agents follow template behaviours. In cases of practical interest, ranging from swarm robotics [Kouvaros and Lomuscio, 2015b; 2016a] to security protocols [Boureanu *et al.*, 2016], PMC can be used to determine whether MAS with an unbounded number of components comply with a given temporal-epistemic specification.

A key issue in evaluating protocols, from swarms robotics to auctions and beyond, is the extent to which they are *resilient to adverse functioning behaviour of some of the agents in the system*. For example, to evaluate the appropriateness of a swarm robotics formation flying protocol it is generally not sufficient to establish whether the local behaviours establish a group formation, but also the degree of robustness of the protocol to local faults. In particular, we may be interested in questions such as: If, for whatever reason, an agent displays faulty behaviour, will this propagate through the group thereby splitting the formation, or will the swarm remain in formation tolerating the faulty unit? If the protocol is tolerant to some faults, is there an upper bound ratio of faulty versus non-faulty agents above which correctness is no longer guaranteed? If so, what is it? This paper aims to develop a technique to answer similar questions in the context of temporal-epistemic specifications. Being able to provide an answer to these questions enables the engineer to engineer MAS that are resilient to adverse functioning circumstances.

The rest of the paper is organised as follows. In Section 2 we present the syntax of indexed CTLK and the semantics of parameterised interleaved interpreted systems which we will use to model MAS. In Section 3 we develop a method for injecting faulty behaviour into the templates of correct agents thereby generating systems that meet a predetermined ratio of faulty versus non-faulty agents. In Section 4 we define the parameterised fault-tolerance problem as the decision problem of establishing whether all (infinitely many) systems under a certain ratio of faults satisfy a given specification of interest. We put forward a complete algorithm that solves the problem under the semantics studied. In Section 5 we present an implementation and report the experimental results obtained when studying the Alpha swarm aggregation algorithm.

Related Work. The PMC problem has long been studied in the context of reactive systems and temporal specifica-

tions [Bloem *et al.*, 2015]. More recently PMC has been put forward to study fault tolerance in the context of distributed computing [John *et al.*, 2013; Fisher *et al.*, 2009]. However these approaches do not tackle epistemic specifications nor ratios of faulty versus non-faulty components, and the technical machinery is different from the one developed here.

As mentioned, some approaches to the PMC problem for MAS against epistemic specifications have been put forward in the recent past [Kouvaros and Lomuscio, 2016b]. Even if fault-tolerance is considered as an advantage of swarms over other robotic architectures, and PMC has been applied to swarms [Kouvaros and Lomuscio, 2015a; 2015b], we are not aware of other solutions using PMC to achieve the aims above.

There is a considerable body of literature in safety-analysis putting forward the use of fault-injection in combination with model checking for the analysis of fault-tolerance in distributed systems. Our approach follows the one pioneered in [Joshi and Heimdahl, 2005; Bozzano and Villafiorita, 2007]. This approach was extended to the analysis of MAS in [Ezekiel and Lomuscio, 2017] and used in [Ezekiel *et al.*, 2011] for the analysis of an underwater autonomous vehicle. However, none of these approaches tackle the analysis of fault-tolerance for systems with an unbounded number of components, which is our key objective here.

2 Background

Parameterised interleaved interpreted systems (PIISs) extend interleaved interpreted systems (IIS) [Lomuscio *et al.*, 2010] to reason about the temporal-epistemic properties of asynchronous MAS with an unbounded number of agents [Kouvaros and Lomuscio, 2013]. Below we outline the PIISs semantics as presented in [Kouvaros and Lomuscio, 2013]. A PIIS consists of the descriptions of an *agent template* from which an unbounded number of homogeneous agents may be constructed and an *environment* in which the agents operate [1].

The agent template $T = \langle L, \iota, Act, P, t\rangle$ defines a non-empty set of local states L, a unique initial state $\iota \in L$, and a non-empty set of actions $Act = A \cup AE \cup GS$. Each action is either an *asynchronous action* (A) or an *agent-environment action* (AE) or a *global-synchronous action* (GS). Each type of action enables a different communication pattern between the concrete agents (see Def. 2). The actions are performed in compliance with a protocol $P : L \to \mathcal{P}(Act)$ that selects which actions may be performed at a given state. The evolution of the local states is characterised by a transition function $t : L \times Act \to L$ returning the next local state given the current local state and the action performed at the state. The environment $e = \langle L_e, \iota_e, Act_e, P_e, t_e\rangle$ is associated with a non-empty set of local states L_e, a unique initial state $\iota_e \in L_e$, a non-empty set of actions $Act_e = A_e \cup AE \cup GS$, a protocol P_e, and a transition function t_e.

Definition 1 (PIIS)**.** *A parameterised interleaved interpreted system is a tuple $\mathcal{S} = \langle T, e, \mathcal{V}\rangle$, where $\mathcal{V} : L \to 2^{L\_AP} \cup 2^{G\_AP}$ is a labelling function on the agent template's states for a set L_AP of local atomic propositions and a set G_AP of global atomic propositions.*

[1] The framework can accommodate a finite number of agent templates; for simplicity we here only present the single template case.

PIISs describe an unbounded family of concrete IIS, each one obtained by setting the parameter prescribing to the number of agents in the system. Given a PIIS $\mathcal{S}$ and an integer $n \geq 1$, the IIS $\mathcal{S}(n)$ of n agents is the result of the composition of n copies of T with the environment. We write $\mathcal{A}$ for the set $\mathcal{A} = \{1, \ldots, n\}$ of concrete agents instantiated from T. A *global state* $g = \langle l_1, \ldots, l_n, l_e\rangle$ is a tuple of local states for all the agents and the environment in $\mathcal{S}(n)$; it describes the system at a particular instant of time. For a global state g we write $g.i$ to denote the local state of agent i in g. The system's global states evolve over time in compliance with the *global transition relation.*

Definition 2 (Global transition relation)**.** *The global transition relation $R \subseteq G \times Act \cup Act_e \times G$ on a set G of global states is defined as $(g, a, g') \in R$ iff one of the following holds:*

- (Asynchronous). *(i) $a \in A \cup A_e$; (ii) there is $i \in \mathcal{A} \cup \{e\}$ s.t. $a \in P(g.i)$ and $t(g.i, a) = g'.i$; (iii) for all $j \neq i$, $g.j = g'.j$.*
- (Agent-environment). *(i) $a \in AE$; (ii) there is $i \in \mathcal{A}$ s.t. $a \in P(g.i)$ and $t(g.i, a) = g'.i$; (iii) $a \in P_e(g.e)$ and $t_e(g.e, a) = g'.e$; (iv) for all $j \neq i$, $j \neq e$, $g.j = g'.j$.*
- (Global-synchronous). *(i) $a \in GS$; (ii) for all $i \in \mathcal{A}$, $a \in P(g.i)$ and $t(g.i, a) = g'.i$; (iii) $a \in P_e(g.e)$ and $t_e(g.e, a) = g'.e$.*

Above R defines only one action to be performed at each time step. If this is an asynchronous action, then exactly one agent participates in the global transition; if it is an agent-environment action, then exactly one agent and the environment participate in the global transition; if it is a global-synchronous action, then all the agents and the environment participate in the global transition.

We now define the concrete systems generated from $\mathcal{S}$.

Definition 3 (Concrete semantics)**.** *Given a PIIS $\mathcal{S}$ and $n \geq 1$, the IIS $\mathcal{S}(n)$ is a tuple $\mathcal{S}(n) = \langle G, g_0, R, V\rangle$, where $G \subseteq L^n \times L_e$ is the set of reachable global states via R from the initial global state $g_0 = \langle \iota, \ldots, \iota, \iota_e\rangle$, and $V : G \to (2^{L\_AP \times \mathcal{A}}) \cup 2^{G\_AP}$ is the labelling function on the global states defined as follows: $(p, i) \in V(g)$ iff $p \in \mathcal{V}(g.i)$, for $p \in L\_AP$, $i \in \mathcal{A}$; and $q \in V(g)$ iff for all $1 \leq j \leq n$, $q \in \mathcal{V}(g.j)$, for $q \in G\_AP$.*

A PIIS generates different IIS depending on the parameter for the system. Each system is composed of a different number of agents. The local propositional variables in an IIS are indexed by each of the concrete agents; (p, i) holds in a global state if the agent i is at a local state labelled with p by the template labelling function; this will enable us to construct specifications independently of the size of the concrete system on which they are evaluated. A global atomic proposition q holds in a global state if all the agents are at a local state labelled with q by the template labelling function; this will allow us to formulate specifications expressing ratios of faulty to non-faulty agents.

A *path* π is an infinite sequence $\pi = g^0 a^0 g^1 a^1 g^2 \ldots$ such that $(g^i, a^i, g^{i+1}) \in R$ for every $i \geq 0$. We write $\pi(i)$ for the

i-th state in π. The set of all paths originating from a state g is denoted by $\Pi(g)$.

We express specifications in the indexed logic IACTLK$\backslash X$. The logic extends ACTLK$\backslash X$ (the universal fragment of the temporal-epistemic logic CTLK without the next time operator) by introducing indexed atomic propositions and indexed epistemic modalities that are quantified over the concrete agents [Kouvaros and Lomuscio, 2016b]. Given a set IND of indices, a set L_AP of local atomic propositions and a set G_AP of global atomic propositions, IACTLK$\backslash X$ formulae are defined by the following BNF grammar:

$$\phi ::= (p, v) \mid \neg(p, v) \mid q \mid \neg q \mid \phi \wedge \phi \mid \phi \vee \phi \mid A(\phi U \phi) \mid A(\phi R \phi) \mid K_i \phi \mid \forall v\colon \phi$$

where $p \in L\_AP$, $q \in G\_AP$, and $v \in IND$. The epistemic modality $K_i\phi$ is read as "agent i knows that ϕ". The temporal modality $A(\phi U \psi)$ stands for "for all paths, at some point ψ holds and before then ϕ is true along the path"; and $A(\phi R \psi)$ denotes "for all paths, ψ holds along the path up to and including the point when ϕ becomes true in the path". An IACTLK$\backslash X$ formula is said to be a *sentence* if every variable appearing the formula is in the scope of a universal quantifier. Hereafter we consider only indexed IACTLK$\backslash X$ sentences.

We now define the satisfaction relation.

Definition 4 (Satisfaction of IACTLK$\backslash X$)**.** *The satisfaction relation $\models$ for an IIS $\mathcal{S}(n)$, and an IACTLK$\backslash X$ sentence ϕ is inductively defined as follows (clauses for propositional connectives are immediate and thus omitted):*

- $(\mathcal{S}(n), g) \models A(\phi_1 U \phi_2)$ *iff* *for every $\pi \in \Pi(g)$, for some $i \geq 0$ $(\mathcal{S}(n), \pi(i)) \models \phi_2$ and for all $0 \leq j < i$, $(\mathcal{S}(n), \pi(j)) \models \phi_1$;*
- $(\mathcal{S}(n), g) \models A(\phi_1 R \phi_2)$ *iff* *for every $\pi \in \Pi(g)$, for all $i \geq 0$, if $(\mathcal{S}(n), \pi(j)) \not\models \phi_1$, for all $0 \leq j < i$, then $(\mathcal{S}(n), \pi(i)) \models \phi_2$;*
- $(\mathcal{S}(n), g) \models K_i \phi$ *iff* *for all $g' \in G$, $g.i = g'.i$ implies $(\mathcal{S}(n), g') \models \phi$;*
- $(\mathcal{S}(n), g) \models \forall v\colon \phi$ *iff* *$(\mathcal{S}(n), g) \models \phi[v \mapsto ag]$ for all $ag \in \mathcal{A}$.*

An IACTLK$\backslash X$ formula ϕ is said to be true in $\mathcal{S}(n)$, denoted $\mathcal{S}(n) \models \phi$, if $(\mathcal{S}(n), g_0) \models \phi$. The *parameterised model checking problem* is to check whether ϕ is true in every concrete system generated from $\mathcal{S}$.

Definition 5 (PMCP)**.** *Given a PIIS $\mathcal{S}$ and an IACTLK$\backslash X$ formula ϕ, the parameterised model checking problem (PMCP) is the decision problem of determining whether the following holds:*

$$\mathcal{S}(n) \models \phi \text{ for every } n > 1.$$

If this holds, then ϕ is said to be satisfied by $\mathcal{S}$; this is denoted by $\mathcal{S} \models \phi$.

The PCMP is in general undecidable [Apt and Kozen, 1986]. Nevertheless restrictions can be imposed on the systems leading to decidable problems; these have been exploited in a variety of applications ranging from networking to MAS against temporal or epistemic specifications [Emerson and Namjoshi, 1995; Kouvaros and Lomuscio, 2016b].

3 Fault Injection via Model Updates

In this section we construct a *faulty* PIIS $\mathcal{S}^f = \langle (T, T^f), e^f, V^f \rangle$ from a given PIIS $\mathcal{S} = \langle T, e, V \rangle$; to distinguish between the two, sometimes we refer to "faulty" $\mathcal{S}^f$ and "nominal", or "non-faulty", PIIS $\mathcal{S}$. In doing so we are inspired by the work on fault-injection in safety-analysis [Bozzano and Villafiorita, 2007], which, differently from our approach, operates on concrete models rather than abstract ones. In essence, $\mathcal{S}^f$ is a PIIS of two agent templates: T and T^f; its concrete instantiations compose an arbitrary number of agents from T and T^f. As we show below, $\mathcal{S}^f$ can be used to reason about $\mathcal{S}$'s *tolerance to faults in the system*. Specifically, in what follows we will devise a method to establish how many faulty agents, expressed as a ratio of faulty versus correctly functioning agents, a MAS can tolerate with respect to a given specification for the system.

Technically T^f is an extension of T; intuitively T^f can represent all T's behaviours but also additional ones encoding (by means of additional states, actions and transitions) various notions of faults which can non-deterministically be exhibited by any concrete agent built from T^f. In what follows we instantiate the mainstream taxonomy of faults previously used to reason about fault-tolerant systems [Bozzano and Villafiorita, 2007; Joshi and Heimdahl, 2005] on PIISs.

We use the Cartesian product of a finite set Var of Boolean, integer, and enumerate variables to encode the agent template's local states, and define a fault as an update on either of these variables that is not modelled by the transition function of the original nominal template. We write l_v to denote the value of the variable v at local state l and $D(v)$ to express the value domain of v. Intuitively each variable represents a different component of the agent and it is associated with different *failure modes* depending on its type:

- *Boolean faults* encode behaviours where a Boolean variable can erroneously get inverted, non-deterministically updated, or stuck at its current value. T^f models the realisation of Boolean faults by performing the actions $invert(v)$, $random\_0(v)$ and $random\_1(v)$, $stuck(v)$, respectively, for each Boolean variable v; we write $BAct$ for the set of all actions pertaining to Boolean faults.
- *Integer faults* represent situations where an integer variable is erroneously increased, decreased, or not updated as it should. T^f models the realisation of integer faults by performing the actions $ramp\_d(v)$, $ramp\_u(v)$, $stuck(v)$, respectively, for each integer variable v; we write $IAct$ for the set of all actions pertaining to integer faults.
- *Enumerate faults* encode transitions where the value of an enumerate variable is replaced by a different value incorrectly, or not updated when it should have. T^f models the realisation of enumerate faults by performing the actions $replace(v, x, y)$ (replace value x of v with y), $update(v, x)$ (update v to x), $stuck\_at(v, x)$ (v is stuck at x whenever at x) for each enumerate variable v; we

write $EAct$ for the set of all actions pertaining to enumerate faults.

We now present the faulty agent template T^f by defining all of its components. T^f is given as a tuple $T^f = \langle L^f, \iota^f, Act^f, P^f, t^f \rangle$, where $L^f = \prod VAR \times faulty \times injected$ is the set of local states and $\iota^f = (\iota, \bot, \bot)$ is the initial state. The Boolean variable *injected* represents whether a fault has ever been injected by a concrete agent following the template. The Boolean variable *faulty* encodes whether a concrete agent will ever inject a fault while the system is evolving. (Note that no faults are ever injected to the variables *faulty* and *injected*.) In particular at the initial state the agent chooses non-deterministically whether it will ever inject a fault. If so, it performs the asynchronous action $faulty\_\checkmark$ thereby setting *faulty* to *true*; otherwise it performs the asynchronous action $faulty\_\times$ thereby setting *faulty* to $\bot$. The set Act^f of actions is equal to $Act^f = Act \cup \{faulty\_\checkmark, faulty\_\times\} \cup FACT$, where $FACT = (Act \times (BAct \cup IAct \cup EAct))$ is the set of actions responsible for injecting faults. The protocol $P^f : L^f \to \mathcal{P}(Act^f)$ is defined by

- $P^f(\iota^f) = \{faulty\_\checkmark, faulty\_\times\}$;
- $P^f(l) = P(l)$ whenever $l \neq \iota^f$ and $l_{faulty} = \bot$;
- $P^f(l) = P(l) \cup (P(l) \times BAct \cup IAct \cup EAct)$ whenever $l \neq \iota^f$ and $l_{faulty} = \top$.

By means of the above at each time-step a fault is non-deterministically injected.

We now explain the consequences, in terms of local transitions, following the presence of a fault. Formally, whenever a fault represented by the action $(a, a^f) \in FACT$ is injected, the variables of the agent are updated as per $t(l, a)$ but for the variable associated with a^f. Specifically the transition function $t^f : L^f \times Act^f \to L^f$ is defined by $t^f(l, act) = l'$ iff either one of the following holds:

- *Initialisation:* $l = \iota^f$ and either $act = faulty\_\checkmark$, $l' = (\iota, \top, \bot)$, or $act = faulty\_\times$, $l' = (\iota, \bot, \bot)$.
- *Nominal transition:* $l \neq \iota^f$, $l'_{faulty} = l_{faulty}$, $l'_{injected} = \bot$, $act \in Act$, and $l'_x = (t(l, act))_x$ for $x \neq faulty \neq injected$.
- *Faulty transitions:* $l \neq \iota^f$, $l_{faulty} = l'_{faulty} = \top$, $l'_{injected} = \top$, $l'_x = (t(l, a))_x$ for $x \neq v \neq faulty \neq injected$, and either one of the following holds:
  - *Boolean inverted:* $act = (a, invert(v))$ and $l'_v = \neg l_v$.
  - *Boolean random:* $act = (a, random\_x(v))$, $l'_v = x$ and $x \in \{0, 1\}$.
  - *Boolean stuck:* $act = (a, stuck(v))$ and $l'_v = l_v$.
  - *Integer ramped down:* $act = (a, ramp\_d(v))$ and $l'_v = \max(0, l_v - 1)$.
  - *Integer ramped up:* $act = (a, ramp\_u(v))$ and $l'_v = \min(|D(v)|, l_v + 1)$.
  - *Integer stuck:* $act = (a, stuck(v))$ and $l'_v = l_v$.
  - *Enumerate replace:* $act = (a, replace(v, x, y))$ and either $l_v = x$, $l'_v = y$ or $l_v \neq x$, $l'_v = (t(l, a))_v$.
  - *Enumerate update:* $act = (a, update(v, x))$ and $l'_v = x$.
  - *Enumerate stuck:* $act = (a, stuck\_at(v, x))$ and either $l_v = l'_v = x$ or $l_v \neq x$, $l'_v = (t(l, a))_v$.

The faulty environment e^f extends e by including all faulty agent-environment and global-synchronous actions. Formally $e^f = \langle L^f_e, \iota^f_e, Act^f_e, P^f_e, t^f_e \rangle$, where: $L^f_e = L_e$; $\iota^f_e = \iota_e$; $Act^f_e = Act_e \cup AE^f \cup GS^f$; P^f_e is as P^e but defined on the faulty actions by $(a, a^f) \in P^f_e$ iff $a \in P_e$; and t^f_e is as t_e but defined on the faulty actions by $t^f_e(l, (a, a^f)) = l'$ iff $t_e(l, a) = l'$.

Lastly, to define the faulty PIIS $\mathcal{S}^f$ we consider the labelling function $V^f = V \cup V'$ to be the union of the labelling function V of $\mathcal{S}$ and the labelling function $V' \colon L^f \to 2^{L\_AP} \cup 2^{G\_AP} \cup 2^{\{injected, faulty\}}$ that assigns propositional variables to T^f's states as follows: $V'(l) = V((l_{v_1}, \ldots, l_{v_{|VAR|}})) \cup \{injected \mid l_{injected=\top}\} \cup \{faulty \mid l_{faulty=\top}\}$, where *injected* and *faulty* are global atomic propositions. In other words V' extends V by labelling the states in L^f with *injected*, *faulty* whenever the corresponding variables are *true*. So if *faulty* is true at a state, then all faulty agents will potentially display a fault during the evolution of the system; if *injected* is true at a state, then all faulty agents have injected a fault at the current time-step.

Having defined the notion of faulty PIISs, we move to express specifications to reason about the fault-tolerance of an unbounded MAS w.r.t. a ratio $1/\lambda$ of faulty to nominal agents and a specification ϕ; note that, as we will formally define in the next section, we are only concerned with the concrete systems satisfying said ratio. For instance, the *total tolerance specification* expresses that the system is not affected (as far as ϕ is concerned) by the actions of the faulty agents:

$$\phi_{TT} \triangleq AG\phi \qquad \text{(Total Tolerance)}$$

In other words, the faulty system satisfies ϕ_{TT} if ϕ always remains true in it, irrespective of how many agents display faulty behaviour.

The *bounded tolerance specification* can be used to give guarantees on the satisfaction of ϕ whenever the ratio of faulty to nominal agents is below $1/\lambda$:

$$\phi_{BT} \triangleq AG(\neg faulty \to \phi) \qquad \text{(Bounded Tolerance)}$$

The *intermittent tolerance specification* determines whether ϕ is tolerant to a ratio of $1/\lambda$ faulty to nominal actions occurring simultaneously at the current time-step:

$$\phi_{IT} \triangleq AG(injected \to \phi) \qquad \text{(Intermittent Tolerance)}$$

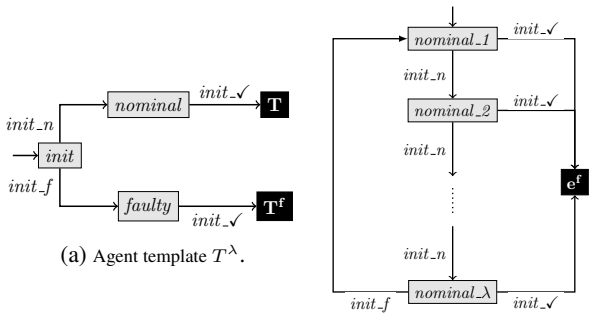
The *recoverability specification* encodes the eventual recoverability of the system in terms of the satisfaction of ϕ:

$$\phi_{R} \triangleq AG(injected \to AF\phi) \qquad \text{(Recoverability)}$$

So on the faulty system the specification ϕ may fail, but eventually it becomes satisfied, thereby indicating an element of recovery of the system with respect to ϕ.

4 Parameterised Fault-tolerance Problem

The previous section discussed the automatic construction of a faulty PIIS $\mathcal{S}^f = \langle (T, T^f), e^f, V^f \rangle$ given a *nominal* PIIS



(a) Agent template T^λ.

(b) Environment template e^λ.

Figure 1: The PIIS $\mathcal{S}^\lambda$.

$\mathcal{S} = \langle T, e, V \rangle$. In this section we develop a method to reason automatically about the faulty system $\mathcal{S}^f$. In particular we aim to be able to answer the following question. Does a system of arbitrarily many $n \geq 1$ agents, up to $\lfloor n/\lambda \rfloor$ of which may develop faults, satisfy a given specification? We define this formally as the decision problem below.

Definition 6 (PFTP)**.** *Given a PIIS $\mathcal{S}$, a natural number λ, and an IACTLK$\backslash X$ formula ϕ, the* parameterised fault-tolerance problem *is the decision problem of determining whether the following holds:*

$$\mathcal{S}^f \left((n, n^f)\right) \models \phi \text{ for every } n \in \mathbb{N}, n^f \in \mathbb{N} \text{ with } n^f = \lfloor n/\lambda \rfloor.$$

If this holds, then ϕ is said to be satisfied by $\mathcal{S}^f$; this is denoted by $\mathcal{S}^f \models^\lambda \phi$.

In other words the PFTP w.r.t. a specification ϕ and a ratio $1/\lambda$ returns "yes" if all the concrete systems of n agents of which $\lfloor n/\lambda \rfloor$ are faulty satisfy the specification ϕ.

We now compare the PFTP and the PMC when they are both defined on faulty systems built from $\mathcal{S}^f$. Note that while the PFTP has technical similarities with the PMCP, the object of study is inherently different: while the PMCP studies all possible faulty systems, the PFTP is limited to faulty systems satisfying a predetermined ratio of faults. This has consequences on the methodology to be employed. While PMCP approaches use counter abstraction methodologies [Kouvaros and Lomuscio, 2015a] (which reduce the PMCP to the problem of checking a single abstract system encoding all concrete systems) or cutoff techniques [Kouvaros and Lomuscio, 2016b] (which reduce the PMCP to the problem of checking all concrete systems up to a *cutoff* system), these procedures cannot be directly employed to solve the PFTP. Specifically, no conclusions can be drawn on the PFTP if the PMCP returns a negative answer. This is because the concrete systems falsifying the specification may contain more faulty agents than the ratio the PFTP prescribes.

To solve this problem we devise abstract templates $\mathcal{S}^\lambda$ that are built from the faulty $\mathcal{S}^f$ but only generate concrete systems satisfying the λ constraint on the ratio of faulty agents present. The PIIS $\mathcal{S}^\lambda$ includes a single agent template that includes an *initialisation phase*. The initialisation phase will ensure that any concrete systems generated from $\mathcal{S}^\lambda$ include exactly one faulty agent every λ nominal agents.

We now describe the PIIS $\mathcal{S}^\lambda = \langle T^\lambda, e^\lambda, V^\lambda \rangle$, represented in Figure 1. In any concrete system built from $\mathcal{S}^\lambda$ all agents are initially in state $init$ and the environment is initially in state $nominal\_1$. At $init$ the agents can either perform the agent-environment action $init\_n$ or the agent-environment action $init\_f$. Each action can be performed only when the environment can also perform the action. But the environment is defined to perform λ $init\_n$ actions consecutively followed by an $init\_f$ action; the former action updates the state of the agent to $nominal$ and the latter to $faulty$. Therefore there is one agent in state $faulty$ for every $\lambda - 1$ agents in state $nominal$. In these states the agents can only perform the global synchronous action $init\_\checkmark$ which marks the end of the initialisation phase. Following this the system reaches a global state in which there is one agent in the initial state of T^f for every $\lambda - 1$ agents in the state of T, and the environment is in the initial state of e^f. Following this the agents behave identically to T^f and T respectively.

Finally the evaluation function V^λ labels a state from T or T^f with an atomic proposition iff the state is labelled with the proposition by V^f.

The following theorem shows that $\mathcal{S}^\lambda$ can be used to solve the PFTP on $\mathcal{S}^f$.

Theorem 1. *Let $\mathcal{S}$ be a PIIS, $\lambda \in \mathbb{N}$, and ϕ an IACTLK$\backslash X$ formula. The following holds: $\mathcal{S}^f \models^\lambda \phi$ iff $\mathcal{S}^\lambda \models \phi$.*

We can exploit Theorem 1 by constructing $\mathcal{S}^\lambda$ automatically, thereby obtaining a procedure to evaluate the fault-tolerance of a system against a specification for a given ratio. In particular, given a nominal PIIS $\mathcal{S} = \langle T, e, V \rangle$, $\lambda \in \mathbb{N}$, and an IACTLK$\backslash X$ formula ϕ the PIIS $\mathcal{S}^f = \langle (T, T^f), e^f, V^f \rangle$ is constructed; T^f extends T to include Boolean, integer, and enumerate faults. Then the PIIS $\mathcal{S}^\lambda$ is built following the description above; the PMCP is then solved on it by employing any parameterised model checker for MAS, e.g., `MCMAS-P` [Kouvaros and Lomuscio, 2016b]. The latter would return *true* iff the PFTP is *true* for $\mathcal{S}^f$, λ, and ϕ.

5 Evaluation

We developed `MCMAS-PFI`, a toolkit realising the fault injection method described earlier, on top of `MCMAS-P`, an open-source model checker for the verification of PIISs [Kouvaros and Lomuscio, 2013]. `MCMAS-P`'s input language closely follows the modular structure of the agent templates. This was extended to include the declaration, as per Section 2, of the faults to be injected on the variables encoding the agents.

Upon its invocation `MCMAS-PFI` builds the faulty agent template by updating, using the procedure described in Section 3 applied to the faults specified, the `MCMAS-P`'s internal structures representing the agent template's components. These are further modified, by using the algorithm in Section 4, to obtain the PIIS $\mathcal{S}^\lambda$ whose concrete instantiations satisfy the given ratio $1/\lambda$. The base model-checker `MCMAS-P` is then called to verify $\mathcal{S}^\lambda$ against the given specifications. Following this, the user can conclude, by using Theorem 1, whether the specifications hold on a MAS of any size whose ratio of faulty to nominal agents falls below $1/\lambda$.

Fault-tolerance in the Alpha algorithm. We now assess the fault-tolerance of the Alpha swarm aggregation algo-

rithm [Winfield *et al.*, 2008], a protocol used to aggregate robotic swarms in the areas of operation. We adopt the typical setting employed to analyse the algorithm [Dixon *et al.*, 2012]. In particular we assume that each robot moves on a two-dimensional arena and communicates with its peers via a wireless sensor of limited range. The arena is assumed to be finite and allowed to wrap around, i.e., for an $\alpha \times \alpha$ arena, the cell $(1, 1)$ is to the right of the cell $(1, \alpha)$.

We now describe the algorithm. To begin, define a robot to be in another robots neighbourhood if the position of the former is in the range of the latter's sensor. Each robot keeps track of the number of its neighbours. This determines the robots' connectedness statuses. Specifically, a robot is said to be *connected* if its neighbourhood is composed of at least α robots, for a threshold α. The behaviour of each of the robots is characterised by their connectivity status and by whether they are in *forward motion mode* or in *coherence motion mode*: if a robot is in forward mode and connected, then it moves forward; if it is in forward mode, but not connected, then it performs a 180° turn and changes its motion mode to coherence; if it is in coherence mode, but not connected, then it moves forward; finally, if it is in coherence mode and connected, then it performs a random 90° turn and changes its motion mode to forward.

In the following we fix a 5×5 arena, assume a communication range of 1, and let $\alpha = 2$. Initially the robots are connected, in forward mode, and collectively have every possible direction of movement. We refer to [Kouvaros and Lomuscio, 2015b] for the formal account of the PIISs modelling this instantiation of the algorithm. We are here interested in analysing the swarm's *connectedness property* [Dixon *et al.*, 2012] "every nominal robot knows that it will be infinitely often connected" as encoded by the indexed ACTLK$\backslash X$ formula [2] $\phi_{AA} \triangleq \forall v\colon K_v GF(connected, v)$. In particular, we are interested in assessing the tolerance of ϕ_{AA} to local malfunctions of the robots. These can generally be realised by either hardware failures or the unpredictability of the environment. We consider the following faults:

- *Direction failures*. Either a robot becomes unable to change direction, or it adopts the wrong direction. To account for these failures, $stuck\_at$ and $update$ faults are injected to the enumerate variable encoding direction.
- *Detection failures*. A robot fails to detect some of the robots in its neighbourhood. This may result in the erroneous update of the connectivity status of the robot, as modelled by the $random$ fault injected in the Boolean variable representing connectedness.
- *Motion failures*. The motion mode of a robot may not be updated as it should. $stuck\_at$ and $update$ faults are injected to the enumerate variable encoding the motion mode to represent these failures.

Having described the above failures as input to `MCMAS-PFI` we instantiated the fault-tolerance specifications of Section 2 on ϕ_{AA}. `MCMAS-PFI` concluded that these are satisfied given $\lambda = 4$; this implies that the specifications are also satisfied whenever $\lambda > 4$. However, all specifications were falsified on input $\lambda = 3$; this implies that the specifications are also falsified whenever $\lambda > 2$ or $\lambda = 1$. Intuitively whenever the percentage of faulty robots is 33% and above, the exhibited faults impact the overall behaviour of the swarm not only by splitting the connected cluster of robots, but also by potentially never allowing them to regroup. This contributes to the recent body of work on the formal analysis of the Alpha algorithm [Dixon *et al.*, 2012; Kouvaros and Lomuscio, 2015b] by establishing the tolerance of the algorithm to direction, detection and motion failures on a swarm of any size whose percentage of faulty agents is below 33%. Having formal bounds on the system's tolerance to faults at runtime, provides clear guidance to engineer the system appropriately to ensure its overall specifications are satisfied.

[2] Note that the formula is evaluated by quantifying only over the nominal agents.

6 Conclusions

Most of the research conducted in validation of MAS concerns the development of systems for showing that a MAS is correct with respect to a given specification. While progress in this area is of fundamental importance, comparatively less attention is devoted to the problem of evaluating how robust a MAS is when it is functioning at runtime. Of course, this is a question routinely addressed by engineers before deployment; but few techniques are available to address this aspect formally and automatically. One exception is safety analysis. In particular, as discussed in the paper, safety analysis via fault-injection provides a way of studying the consequences of faults with respect to a specification, and therefore assess the resilience of a system with respect to faults.

Fault-injection has so far been limited to systems and MAS whose agents are known at design time. Yet, when MAS are deployed as in, e.g., auctions, swarm systems, etc., the number of agents that will populate the system is not known. As a consequence, no existing technique can evaluate the consequences of non-ideal behaviour of a portion of the components for a system of varying size. Yet, some unbounded MAS are employed precisely because they offer a degree of robustness with respect to run-time faults. The method that we presented in the paper enables the formal and automatic analysis of the robustness of an unbounded MAS with respect to a given temporal-epistemic specification. In particular, we can assess whether all (infinitely many) systems in which the ratio of faulty versus non-faulty agents is under a given parameter satisfy the specification. The tool we presented makes it possible to study this problem automatically.

Note that in the paper we did not discuss the problem of identifying the ratio under which a system satisfies a given specification. Still, the tool that we presented already makes this possible by iteratively calling it until the ratio is found. A deeper analysis of this will be conducted in future work together with further applications to swarm analysis.

Acknowledgements

The research described in this paper was supported by the EPSRC under grant EP/I00529X/1 and an EPSRC Doctoral Prize Fellowship.

References

[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.

[Belardinelli *et al.*, 2016] F Belardinelli, A. Lomuscio, and J. Michaliszyn. Agent-based refinement for predicate abstraction of multi-agent systems. In *Proceedings of ECAI16*, pages 286–294. IOS Press, 2016.

[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.

[Bordini *et al.*, 2003] R. H. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge. Model checking AgentSpeak. In *Proceedings of AAMAS03*, pages 409–416. ACM Press, 2003.

[Bordini *et al.*, 2006] R. H. Bordini, M. Fisher, W. Visser, and M. Wooldridge. Verifying multi-agent programs by model checking. *Autonomous Agents and Multi-Agent Systems*, 12(2):239–256, 2006.

[Boureanu *et al.*, 2016] I. Boureanu, P. Kouvaros, and A. Lomuscio. Verifying security properties in unbounded multi-agent systems. In *Proceedings of AAMAS16*, pages 1209–1218. IFAAMAS, 2016.

[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.

[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.

[Emerson and Namjoshi, 1995] E.A. Emerson and K.S. Namjoshi. Reasoning about rings. In *Proceedings of POPL95*, pages 85–94. Pearson Education, 1995.

[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.

[Fisher *et al.*, 2009] M. Fisher, B. Konev, and A. Lisitsa. Temporal verification of fault-tolerant protocols. In *Methods, Models and Tools for Fault Tolerance*, pages 44–56. Springer, 2009.

[Gammie and van der Meyden, 2004] P. Gammie and R. van der Meyden. MCK: Model checking the logic of knowledge. In *Proceedings of CAV04*, volume 3114 of *LNCS*, pages 479–483. Springer, 2004.

[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 *Formal Methods in Computer-Aided Design (FMCAD), 2013*, pages 201–209. IEEE, 2013.

[Joshi and Heimdahl, 2005] A. Joshi and M. Heimdahl. Model-based safety analysis of simulink models using scade design verifier. In *Proceedings of SAFECOMP05*, pages 122–135. Springer, 2005.

[Kacprzak *et al.*, 2004] M. Kacprzak, A. Lomuscio, and W. Penczek. From bounded to unbounded model checking for temporal epistemic logic. *Fundamenta Informaticae*, 63(2-3):221–240, 2004.

[Kacprzak *et al.*, 2008] M. Kacprzak, W. Nabialek, A. Niewiadomski, W. Penczek, A. Pólrola, M. Szreter, B. Woźna, and A. Zbrzezny. Verics 2007 - a model checker for knowledge and real-time. *Fundamenta Informaticae*, 85(1):313–328, 2008.

[Kouvaros and Lomuscio, 2013] P. Kouvaros and A. Lomuscio. A cutoff technique for the verification of parameterised interpreted systems with parameterised environments. In *Proceedings of IJCAI13*, pages 2013–2019. AAAI Press, 2013.

[Kouvaros and Lomuscio, 2015a] P. Kouvaros and A. Lomuscio. A counter abstraction technique for the verification of robot swarms. In *Proceedings of AAAI15*, pages 2081–2088. AAAI Press, 2015.

[Kouvaros and Lomuscio, 2015b] P. Kouvaros and A. Lomuscio. Verifying emergent properties of swarms. In *Proceedings of IJCAI15*, pages 1083–1089. AAAI Press, 2015.

[Kouvaros and Lomuscio, 2016a] P. Kouvaros and A. Lomuscio. Formal verification of opinion formation in swarms. In *Proceedings of AAMAS16*, pages 1200–1209. IFAAMAS, 2016.

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

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

[Lomuscio *et al.*, 2017] A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A model checker for the verification of multi-agent systems. *Software Tools for Technology Transfer*, 19(1):9–30, 2017.

[Meyden and Su, 2004] R. van der Meyden and K. Su. Symbolic model checking the knowledge of the dining cryptographers. In *Proceedings of CSFW04*, pages 280–291. IEEE Computer Society, 2004.

[Raimondi and Lomuscio, 2005] F. Raimondi and A. Lomuscio. Automatic verification of multi-agent systems by model checking via OBDDs. *Journal of Applied Logic*, 5(2):235–251, 2005.

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