# Verifying Fault-tolerance in Parameterised Multi-Agent Systems

Work by Panagiotis Kouvaros[1,2] and Alessio Lomuscio[1]
Presented by Edoardo Pirovano[1]

[1]Imperial College London, UK

[2]University of Naples, Italy

FRIDA 2017 - 16 October 2017

Based on material in:
P. Kouvaros, A. Lomuscio. Verifying Fault-tolerance in Parameterised Multi-Agent Systems. Proceedings of the 26th Conference on Artificial Intelligence (IJCAI17). Melbourne, Australia. 288-294. AAAI Press.

## Outline

## Formal Verification of MAS

- Concerned with showing that a MAS is correct with respect to its specifications.
- Specifications formally expressed in temporal, epistemic, strategic languages.
- Considerable amount of work from 2000, both theoretical investigations (complexity, etc.) and practical algorithms.
- Implementations including MCMAS (Imperial), MCK (UNSW), Verics (Warsaw).
- Applications in robotics, services, security, etc.

**Limitation: number of agents fixed at design time.**

## Robot Swarms

## Unbounded MAS

- Behaviourally identical agents following *simple* protocols.
- Agents may interact in subtle ways thereby displaying emergent properties that are *difficult to predict, yet important to establish*.
- Traditionally unbounded/open MAS are analysed via optimisation techniques and simulations. Both have limitations.

**Key question: Do specifications hold *irrespective* of the number of agents in the system?**

Theoretical challenge: Verifying unbounded MAS is undecidable.

## Parameterised Model Checking for MAS

A technique to reason about MAS *irrespective of the number of components*.

**Parameterised Model Checking Problem:**

$$\forall n \geq |J| : \mathcal{S}(n) \models \bigwedge_{J} \phi(J)$$

- Originally introduced in analysis of networked systems and distributed systems.
- Several techniques recently studied in the context of MAS, including cutoffs [KL13] and counter-abstraction [KL15a].
- Applications in robot swarms [KL15b, KL16a], security [BKL16], data-aware systems [BKL17].

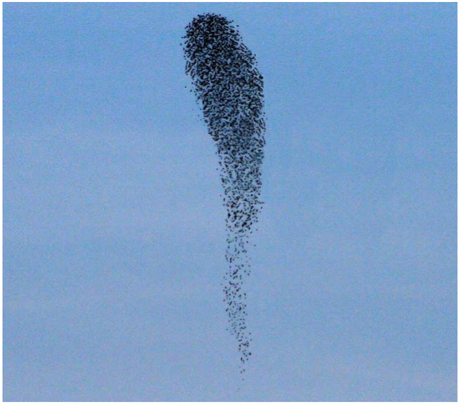
Open source model checker MCMAS-P available for download.

## Limitations of PMC

1. Sound but incomplete techniques.
2. Undecidability hinders applicability in certain settings.
3. As with plain model checking, PMC can “only” formally assess the correctness of a system against a specification.

**But how can we assess the robustness of an unbounded MAS against faults, malfunctions, or unwanted behaviours of some of its components?**

## Just “How Robust” is my MAS?



PMC may enable us to show that the flock will remain connected irrespective of how many agents populate it.

## Just "How Robust" is my MAS?

**But what happens if one or more agents deviate from the expected behaviour?**

# Outline

## Agents

**Definition (Agent Template)**

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* $(GS)$. 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.

## Environment

**Definition (Environment)**

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 $t_e$.

## Parameterised Interleaved Interpreted Systems

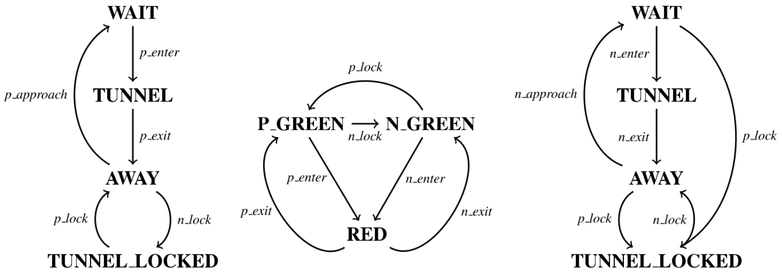
- A parametrised interleaved interpreted system (PIIS) is a finite number of agent templates, together with an environment and a labelling function that assigns which from a set of atomic propositions (each of which can be *global* or *local*, more on this later) are true at which states of the agent.

- 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 $\mathcal{S}$ with one agent template and an integer $n \geq 1$, the IIS $\mathcal{S}(n)$ of $n$ agents is the result of the composition of $n$ copies of the agent with the environment (we will describe this composition in more detail soon).

## Train-gate controller

- The scenario concerns a number of trains wishing to enter a tunnel (one at a time) and a controller that governs which trains can enter.
- Some trains are prioritised and can enter the tunnel whenever it is free. Normal trains, on the other hand, can only enter the tunnel when no prioritised trains are waiting.
- We will model the system by agent templates of two roles representing the two types of trains and an environment template representing the controller.

## Train-gate controller



(a) Template prioritised train. (b) Template controller. (c) Template normal train.

Figure: The PIIS for the train-gate controller. Note *p_enter* and *p_exit* are agent-environment actions, *n_lock* and *p_lock* are global-synchronous actions, and *p_approach* and *n_approach* are asynchronous actions.

## Global states and transitions

- Given a parametrised system and a number $n \geq 1$ for each of the agent templates, we can build a concrete system representing the composition of these along with an environment.
- This global system performs one action at each step and it is either:
  - *Asynchronous:* Precisely one agent or the environment changes state on its own. All others remain in the same state.
  - *Agent-environment:* One agent and the environment perform the action at the same time and change state accordingly. The other agents remain in the same state.
  - *Global-synchronous:* All agents and the environment perform the action at the same time and change state accordingly.
- The labelling function on global states has a predicate $(p, i)$ hold if a *local atomic proposition* $p$ is true for the local state of agent $i$ in that global state. For a *global atomic propositions* $q$ we instead require it to be true at the local state of all the agents.

## Syntax of IACTLK\X

**Definition (IACTLK\X formulae)**

Given a set $IND$ of indices, a set $L\_AP$ of local atomic propositions and a set $G\_AP$ of global atomic propositions, IACTLK\X formulae are defined by the following BNF grammar:

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

where $p \in L\_AP$, $q \in G\_AP$, and $v \in IND$.

An IACTLK\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 sentences.

## Parameterised Model Checking Problem

**Definition (PMCP)**

Given a PIIS $\mathcal{S}$ and an IACTLK\X formula $\phi$, 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 $\phi$ is said to be satisfied by $\mathcal{S}$; this is denoted by $\mathcal{S} \models \phi$.

The PMCP is in general undecidable. Nevertheless restrictions can be imposed on the systems leading to decidable problems.

## Cut-Offs

- One class of models for which the PMCP is decidable is those in which there is an *agent-environment simulation*.
- Informally, this means the way the agent and environment templates interact means that the environment behaves like a mutual exclusion controller, with actions representing taking a lock on a resource and only one agent holding a lock on each resource at any given time.
- In this case, it is possible to compute a *cut-off*, which is a bound on the size of the systems we need to check because any system with more agents than this can be simulated by a smaller system. This restores decidability.
- This was the case in, for example, our earlier train-gate controller example.

## Outline

## Fault Injection

We can construct a *faulty* PIIS $\mathcal{S}^f = \langle (T, T^f), e^f, V^f \rangle$ from a given *nominal* PIIS $\mathcal{S} = \langle T, e, V \rangle$ by adding a second agent template that can, in addition to performing all the usual actions, perform some faulty actions according to some failure modes. We consider the following *failure modes*:

- *Boolean faults* encode behaviours where a Boolean variable can erroneously get inverted, non-deterministically updated, or stuck at its current value.
- *Integer faults* represent situations where an integer variable is erroneously increased, decreased, or not updated as it should.
- *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 been.

The environment can also similarly mutated to produce a faulty environment.

## Notions of fault-tolerance

Given a specification $\phi$ in IACTLK$\backslash$X, we can investigate whether the mutated system satisfies various notions of fault tolerance including:

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

$$\phi_R \triangleq AG(injected \rightarrow AF\phi) \qquad \text{(Recoverability)}$$

$$\phi_R \triangleq AG(injected \rightarrow AFK_i\phi) \qquad \text{(Diagnosability)}$$

## Notions of fault-tolerance

The key question we would like to answer is **"How many faulty agents does it take for a specification to become unsatisfied?"**

$$\phi_{BT} \triangleq AG(faulty_{\leq 1/\lambda} \rightarrow \phi) \qquad \text{(Bounded Tolerance)}$$

$$\phi_{IT} \triangleq AG(injected_{\leq 1/\lambda} \rightarrow \phi) \qquad \text{(Intermittent Tolerance)}$$

where $1/\lambda$ is the ratio of faulty agents in the system.

Equally we can express notions of recovery in terms of a ratio $1/\lambda$.

## The Parameterised Fault-Tolerance Problem

**Definition (PFTP)**

Given a MAS $\mathcal{S}$, a natural number $\lambda$, and an IACTLK$\setminus X$ formula $\phi$, the *parameterised fault-tolerance problem* (PFTP) 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 so, we say $\mathcal{S}^f \models^\lambda \phi$.

So the PFTP returns "yes", if all (infinitely many) MAS where the proportion of faulty agents is up to $1/\lambda$ comply with $\phi$.

## Outline

## Solving the PFTP

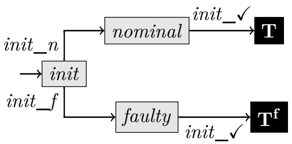
We will show to construct, given a PIIS $\mathcal{S}$ and a $\lambda \in \mathbb{N}$, a new PIIS $\mathcal{S}^\lambda$ such that the following theorem holds:

**Theorem**

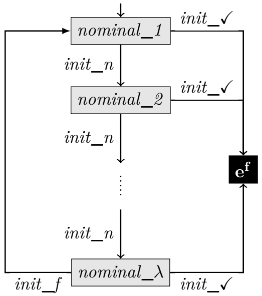
*Let $\mathcal{S}$ be a MAS, $\lambda \in \mathbb{N}$, and $\phi$ an IACTLK$\backslash X$ formula. The following holds: $\mathcal{S}^f \models^\lambda \phi$ iff $\mathcal{S}^\lambda \models \phi$.*

Then, $\mathcal{S}^\lambda \models \phi$ can be established via PMC (using, for example, MCMAS-P) to solve the PFTP.

# Encoding PFTP in PMCP



(a) Agent template.



(b) Environment template.

Figure: Our construction of a PIIS to solve the PFTP. Here, *init_n* and *init_f* are new agent-environment actions that make an agent either faulty or non-faulty. *init_*✓ is a new global synchronous action that ends the initialisation phase.

## Outline

## Implementation

- Implemented a toolkit called MCMAS-PFI on top of MCMAS-P. Its inputs are agent templates, agent mutation rules (which of the failure modes we wish to apply and to which variables), $\lambda$, and specifications.
- MCMAS-PFI constructs the mutated agent and environment, as described in previous slide and then checks the specification using MCMAS-P.
- Used this to assess the Alpha aggregation algorithm.

## The Alpha Aggregation Algorithm

- The *alpha aggregation algorithm* is an algorithm designed to make robots in a swarm group together. We assume the robots move on a two-dimensional arena and communicate with their peers via a wireless sensor of limited range. The arena is assumed to be finite and wrap around.

- We define a robot to be in another robot's 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 $\alpha$ robots, for a threshold $\alpha$.

# The Alpha Aggregation Algorithm

- 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 $180^\circ$ turn and changes its motion mode to coherence
    - if it is in coherence mode, but not connected, then it moves forward
    - if it is in coherence mode and connected, then it performs a random $90^\circ$ turn and changes its motion mode to forward.

## Fault injection example

```
FaultInjection
       ratio=3;
       stuck(connected); invert(connected);
       stuck_at(motion, coherence); update(direction,null);
end FaultInjection
```

Figure: A snippet that exemplifies our modelling language when used to encode potential faults in the Alpha algorithm.

We consider the following faulty behaviours:

- *Direction failures:* Either a robot becomes unable to change direction, or it adopts the wrong direction.
- *Detection failures:* A robot fails to detect some of the robots in its neighbourhood.
- *Motion failures:* The motion mode of a robot may not be updated as it should.

## Results

- Having encoded the Alpha aggregation protocol and possible faults in it, we checked the specification:
  $$\phi_{AA} \triangleq \forall v\colon K_v GF(connected, v)$$
  This expressed the *connectedness property* that every nominal robot knows that it will be infinitely often connected.

- This held for $\lambda = 4$ but not for $\lambda = 3$, telling is that if more than approximately a third of agents are faulty (under the three failure modes we described), the algorithm no longer meets its intended specification.

## Outline

## Conclusions

- A technique to formally reason about the consequences of faults on infinitely many MAS sharing predetermined template behaviours.
- Faults are injected automatically into correct templates.
- PFTP solved via PMCP.
- Implementation (MCMAS-PFI) available as an extention to MCMAS-P.
- Future work will look at synthesis a value for $\lambda$ given a specification and further applications to swarm analysis.

Questions?