Formal Verification of Open Multi-Agent Systems

Panagiotis Kouvaros
Imperial College London
London, United Kingdom
p.kouvaros@imperial.ac.uk

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

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

Hashan Punchihewa
Imperial College London
London, United Kingdom
hashan.punchihewa17@imperial.ac.uk

ABSTRACT

We study open multi-agent systems in which countably many agents may leave and join the system at run-time. We introduce a semantics, based on interpreted systems, to capture the openness of the system and show how an indexed variant of temporal-epistemic logic can be used to express specifications on them. We define the verification problem and show it is undecidable. We isolate one decidable class of open multi-agent systems and give a partial decision procedure for another one. We introduce MCMAS-OP, an open-source toolkit implementing the verification procedures. We present the results obtained using our tool on two examples.

ACM Reference Format:
Panagiotis Kouvaros, Alessio Lomuscio, Edoardo Pirovano, and Hashan Punchihewa. 2019. Formal Verification of Open Multi-Agent Systems. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019,* IFAAMAS, 9 pages.

1 INTRODUCTION

Given the increasing calls for AI to be trustworthy and explainable there has been growing attention to issues of safety and reliability in the context of AI systems. Indeed, over the past 10 years several methods have been put forward to give guarantees that multi-agent systems (MAS) meet their intended specifications.

These have taken the form of verification techniques based both on model checking [16, 18, 25] and theorem proving [1, 31]. A focus of attention has been the development of verification methods that support agent-based specifications, including the temporal evolution of the knowledge of the agents, their desires, intentions, and strategic abilities.

This work has been proven useful in a variety of application areas, including autonomous systems [14], but also services [26], communication protocols [36], and beyond, thereby enabling designs to be validated or bugs to be discovered and rectified.

A limitation of this line of work has traditionally been that the number of agents needs to be known at design-time for the models to be built and verified. However, in several application areas this is an unrealistic assumption. For example, in robotic swarms we are interested in global properties of the MAS irrespective of the actual number of robots occurring in a given execution, which may not be known until run-time. Approaches based on parameterised model checking have provided methods to give guarantees on a MAS with an arbitrary number of components [21]. While these approaches have been used to study safety in swarm protocols and their resilience to faults [22, 23], they rely on the assumption that every system run has a fixed, even if arbitrarily large, number of agents. Clearly, *open systems* do not meet this assumption.

Open MAS, or *OMAS*, are multi-agent systems in which agents may enter or leave the system at run-time. Noteworthy examples of OMAS include auctions where participants may join or leave the bidding scene, robotic swarms where agents may leave the group or malfunction, emerging decentralised applications such as IoT and blockchain, and in general any system in which the number of agents is unbounded and varies at run-time. The aim of this paper is to contribute both to our understanding of the verification problem for OMAS and to the practical challenges of verifying OMAS.

The rest of the paper is organised as follows. After discussing related work, in Section 2 we propose a semantics for OMAS, show how expressive temporal-epistemic specifications can be evaluated on them, define the verification problem and show its undecidability. In Sections 3 and 4 we identify classes of synchronous and asynchronous OMAS and present decision procedures that address the verification problem against the specifications defined in Section 2. In Section 5 we present an open-source toolkit implementing the procedures of Sections 3 and 4, and apply it to scenarios from MAS. We conclude in Section 6.

Related Work. Much of the work in verification of MAS assumes a fixed number of agents given at design-time [1, 9, 25, 36]. This makes it unsuitable for verifying systems where the number of agents is not known until run-time.

This limitation is overcome in work on parameterised verification for MAS [21], which aims to determine whether a property is satisfied irrespective of the number of components in the system. However, this line of work assumes that, while the number of agents in the system can be arbitrarily large, it remains fixed during each execution of the system. Thus, while arbitrarily large systems may be considered, open systems where the number of agents varies at run-time cannot be analysed in that framework.

One of the very few formal approaches to open MAS where this assumption is not made is [3], where a formal semantics and a first-order temporal-epistemic specification language to reason about open MAS was given. While their semantics is different from

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

ours, the verification problem they consider is also undecidable. The class of decidable systems identified in their work relies on active domains semantics from database theory. This has correspondences to our approach as we also assume that no run exists with an unbounded number of agents along the run. Indeed, this is likely to be a requirement for any decidability result. However, while [3] focuses on theoretical results only, we here provide concrete procedures and an implementation. Moreover, [3] does not identify the decidable classes of open interpreted systems which, as we will see in later sections, are useful in applications.

Formal approaches to open distributed systems have been explored more broadly in theoretical computer science. For example, variants of the PI calculus target open systems [34] and model checking approaches tackle dynamic networks of processes [30]. However, the emphasis of this work is quite different from ours. Our specifications include aspects such as epistemic modalities that are tailored to the verification of AI systems, and our approach is focused on the semantics of MAS, rather than a calculus for the interactions between the processes.

Finally, from an epistemic logic angle, quantification over the epistemic modalities, as we do here, was put forward in a number of papers, including [5]. However, these treatments are focused on the resulting axiomatisations and are not concerned with verification.

A related line of work deals with MAS that manipulate unbounded data during systems runs [2, 4, 6, 7, 11]. While there are some similarities between these works and the present contribution, particularly in the study of the decidability of the verification problem, these approaches are all centred on issues pertaining to the unbounded nature of relational data, sometimes combined with an unbounded number of agents. Instead, the aim here is to study systems where agents enter and leave the system at run-time.

2 OPEN MULTI-AGENT SYSTEMS

In this section we present and exemplify a formal semantics for Open Multi-Agent Systems (OMAS), and a specification language for expressing their properties.

Semantics. We begin by defining *Open Interpreted Systems* (OIS), a semantics for reasoning about OMAS that is based on interpreted systems, the standard framework for modelling multi-agent systems [15]. An OIS consists of agents, which capture the behaviours of the individuals that are joining and leaving the system, and an environment capturing the rest of the state of the system.

Definition 2.1 (Agent). An *agent* is defined by a tuple $A = \langle L, \iota, Act, P, t\rangle$ consisting of a set of local states L, a unique initial state $\iota \in L$, a non-empty set of actions Act, a protocol $P : L \rightarrow \mathcal{P}(Act)$ that selects which actions may be performed at a given state, and a transition function $t : L \times Act \times \mathcal{P}(Act) \times Act_E \rightarrow L$ that gives the agent's next state given its current state, the action performed by the agent, the set of actions performed by the other agents, and the action performed by the environment.

Notice that unlike in interpreted systems where every agent can have a unique behaviour, we here assume that the agents are homogeneous. It is possible to extend the semantics up to a finite number of different agent behaviours while preserving all our results. However, for simplicity we do not pursue this here.

Notice also that the agents' transition function depends on the projection of the joint action into a set rather than the joint action itself. Formalising it via the latter would imply having to consider arbitrarily large joint actions thereby limiting the possibility of implementing it. In many scenarios, it may be desirable to count how many copies of each action are performed. This is possible, provided we only count up to some fixed number. Again, we do not pursue this extension here for simplicity.

Having defined agents, we now proceed to define the environment. The environment is defined similarly to the agents and is intended to model the context that the agents are acting in.

Definition 2.2 (Environment). The *environment* is given by a tuple $E = \langle L_E, \iota_E, Act_E, P_E, t_E\rangle$ that defines 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 protocol $P_E : L_E \rightarrow \mathcal{P}(Act_E)$ that defines which actions are enabled at each local state, and a transition function $t_E : L_E \times Act_E \times \mathcal{P}(Act) \rightarrow L_E$ that gives the environment's next state given its current state, the action it performed and the set of actions performed by the agents.

Having defined the agents and environment, we now define an open interpreted system.

Definition 2.3 (OIS). An *open interpreted system* (OIS) is a tuple $O = \langle A, E, V\rangle$, where $V : L \rightarrow \mathcal{P}(AP)$ is a labelling function for the agents' local states.

Hereafter we associate with each agent an integer giving it an identity. The identity of an agent is uniquely assigned to it when it first joins the system, and never changes for as long as the agent remains in the system. This will allow us to express properties that track the evolution of a specific agent.

We now proceed to describe the behaviour of an OIS over time. Towards this we introduce some notation. We use $[n]$ to denote the set $\{1, \ldots, n\}$. We denote by G_n the set of tuples that describe the system at a particular instance of time when precisely n agents are present in the system. In other words, G_n is the set of $(n+1)$-tuples of the form $(L \times \mathbb{Z}^+) \times \cdots \times (L \times \mathbb{Z}^+) \times L_E$ giving the local state and identity for each of the agents, and the local state of the environment. We call such a tuple a *global state*. For a global state g we write $g.i$ and $id(g.i)$ to denote the local state and identity of the i-th agent in g, $s_i(g)$ to represent the local state of the agent with identity i in g, and $g.E$ to represent the state of the environment in g. We use $G \triangleq \cup_{n\in\mathbb{N}} G_n$ for the set of all global states of any size.

Similarly, for a joint action a, we use $a.i$ to denote the action of agent i and $a.E$ to denote the action of the environment. We use $\dot{a}$ to denote the projection of a into a set, i.e., $\dot{a} \triangleq \{a.1, \ldots, a.n\}$. We also use $\dot{a}_{-i}$ to denote the set of actions in a performed by the agents excluding i: $\dot{a}_{-i} \triangleq \{a.1, \ldots, a.(i-1), a.(i+1), \ldots, a.n\}$. Finally, we write ACT_n for the set of all possible joint actions for n agents and the environment, i.e., the set of $(n+1)$-tuples of the form $Act \times \cdots \times Act \times Act_E$.

We now define how the global states evolve over time. The evolution is in compliance with a *global transition relation*. As a stepping stone, we first define a family of subsets of this relation defining how the MAS of a fixed size n evolves. We will then extend this to add transitions for agents joining or leaving the system.

Definition 2.4 (Global transition relation of size n). For each $n \in \mathbb{N}$, we define the *global transition relation of size n*, $R_n \subseteq G_n \times G_n$ on a set G_n of global states by $(g, g') \in R_n$ iff there is some $a \in ACT_n$ s.t. the following hold:

(i) $a.E \in P_E(g.E)$ and for all $i \in [n]$, $a.i \in P_i(g.i)$;
(ii) $g'.E = t_E(g.E, a.E, \dot{a})$;
(iii) for all $i \in [n]$ it is the case that $id(g'.i) = id(g.i)$ and $g'.i = t(g.i, g.E, \dot{a}_{-i}, a.E)$.

Condition (i) ensures that the action is possible, in the sense that it respects the protocol function of both the environment and all the agents. Condition (ii) ensures that the transition respects the transition function of the environment, whilst condition (iii) ensures that no agents change their identity and the transition function of all agents is respected.

Having defined this family of relations which describe how systems of a fixed size would evolve, we extend this to a global transition relation for open systems. This will include all transitions for systems of a fixed size, and also additional transitions for agents joining or leaving the system.

Definition 2.5 (Global transition relation). The *global transition relation* $R \subseteq G \times G$ on a set G of global states is defined by $(g, g') \in R$ iff one of the following holds:

- *(Joint action).* There is $n \in \mathbb{N}$ s.t. $(g, g') \in R_n$. This represents a transition of the system of size n, as defined in Definition 2.4.
- *(Agent joining).* (i) There is $n \in \mathbb{N}$ s.t. $g \in G_n$ and $g' \in G_{n+1}$; (ii) For all $i \in [n] \cup \{E\}$ it is the case that $g.i = g'.i$; (iii) $g'.(n+1) = \iota$; $id(g'.(n+1))$ is a freshly generated natural number (i.e., a number that has not at any point in the execution of the system been used as an identity for an agent).
- *(Agent leaving).* (i) There is $n \in \mathbb{N}$ s.t. $g \in G_n$ and $g' \in G_{n-1}$; (ii) There is $k \in [n]$ s.t. $g' = g_{-k}$, where g_{-k} denotes the tuple resulting from removing the k-th component from g.

Notice that in the above we assume that agents may join and leave at any moment in the execution of the system. This is a reasonable assumption in many cases such as IOT applications where it may be possible for a device to lose or gain connectivity at any point. In applications where it is not a reasonable assumption, we can still model agents not immediately being able to join in certain states by requiring agents in the initial state to perform some "special" joint action with the rest of the system before they can actively participate in the system.

We now associate a *model* to each OIS, which we will use to interpret temporal-epistemic formulae.

Definition 2.6 (Model). The associated *model* of an OIS $O = \langle A, E, V \rangle$ is a tuple $O = \langle G, g_0, R, \mathcal{V} \rangle$, where G is the set of global states reachable via R from the initial global state $g_0 = (\iota_E)$, and $\mathcal{V} : G \to \mathcal{P} \times \mathbb{Z}^+$ is the labelling function defined for a set of atomic propositions $\mathcal{P}$ as $(p, i) \in \mathcal{V}(g)$ iff there is j with $id(g.j) = i$ and $p \in V(g.j)$.

The atomic propositions are indexed by the agents' identities so that (p, i) is true in a state iff there is an agent with identity i in the state whose local state is labelled with p. This will enable us to express specifications irrespectively of how many agents will eventually join the system by quantifying over the agents. Note that our specifications will only express properties of the state of agents, and not of the global state of the system or the environment.

A *path* π is an infinite sequence $\pi = g^0 g^1 g^2 \dots$ of states such that $(g^i, g^{i+1}) \in R$ for every $i \geq 0$. We write $\pi(i)$ for the i-th state in π and $\Pi(g)$ for the set of all paths originating from a state g.

Following the typical treatment for closed unbounded systems whereby an unbounded but finite number of agents are participating in the system [8, 21], we here consider their open variant to adhere to the same condition. In particular, we assume that along any path there is a finite but unbounded number of agents participating in the system. We thus exclude from $\Pi(g)$ all paths with infinitely many *agent joining* transitions. Note the restriction does not prevent the description of agents exhibiting infinite alternations between active and inactive states (indeed, an agent template can be constructed to represent this), but it only prohibits the participation of an infinite set of pairwise distinct individuals in the system. This is in line with any physical system, such as drone swarms or automatic auctions, which will necessarily have at most as many distinct agents participating in the protocol as those that were constructed.

We similarly define the model $O(n) = \langle G_n, g_{n,0}, R_n, \mathcal{V} \rangle$ for the closed system with a constant n agents present and initial global state $g_{n,0} = ((\iota, 1), \dots, (\iota, n), \iota_E)$.

Example 2.7 (Train-gate controller). To illustrate the semantics we describe the OIS of the train-gate controller [17]. In this problem, a number of trains wish to enter a tunnel and a controller has to ensure that at most one train is in the tunnel at any time. We will encode our trains as agents and our controller as the environment.

The agent has states $L \triangleq \{entering, tunnel, outside\}$, with *outside* initial. It has actions $Act \triangleq \{enter, exit, approach, wait\}$ and protocol $P : L \to \mathcal{P}(Act)$ given by $P(entering) = \{enter\}$, $P(tunnel) = \{wait, exit\}$, and $P(outside) = \{wait, approach\}$. Finally, we define the transition function $t : L \times Act \times \mathcal{P}(Act) \times Act_E \to L$ by:

$$(l, a, X, a_E) \mapsto \begin{cases} tunnel & \text{if } a = enter \\ outside & \text{if } a = exit \\ entering & \text{if } a = approach, a_E = go \text{ and } X = \{wait\} \\ l & \text{otherwise} \end{cases}$$

The environment has two states $L_E \triangleq \{green, red\}$, with *green* initial. It has actions $Act_E \triangleq \{go, stop\}$ and protocol $P : L_E \to \mathcal{P}(Act_E)$ given by $P(green) = \{go\}$ and $P(red) = \{stop\}$. Its transition function $t_E : L_E \times Act_E \times \mathcal{P}(Act) \to L_E$ is given by:

$$(l_E, a_E, X) \mapsto \begin{cases} red & \text{if } enter \in X \\ green & \text{if } exit \in X \\ l_E & \text{otherwise} \end{cases}$$

Specifications. 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 agents. Excluding the next time operator and the restriction to the universal fragment are typical in parameterised verification and necessary in order to obtain decidability [21]. Given a set *IND* of indices, and a set *AP* of atomic propositions, IACTLK$\backslash X$ formulae are defined by the following BNF grammar:

$$\begin{aligned}\phi &::= \forall v\colon \phi \mid \psi \\ \psi &::= (p,v) \mid \neg(p,v) \mid \psi \wedge \psi \mid \psi \vee \psi \mid A(\psi U \psi) \\ &\quad \mid A(\psi R \psi) \mid K_v \psi\end{aligned}$$

where $p \in AP \cup \{alv\}$, and $v \in IND$. We introduce a fresh atomic proposition alv (for alive) that is true in a state if the agent that it is indexed by is present in the state. The epistemic modality $K_v\psi$ is read as "agent v knows that ψ". 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". Notice we assume that universal quantifiers only appear at the beginning of a formula. We further assume that formulas are *sentences*, i.e., every variable appearing in the formula is in the scope of a universal quantifier. We will abbreviate $\forall v_1 : \ldots \forall v_n : \psi$ to $\forall_{v_1,\ldots,v_n} : \psi$. We now define the satisfaction relation.

Definition 2.8 (Satisfaction of IACTLK$\backslash X$). The satisfaction relation $\models$ is inductively defined for an OIS O, a global state g and an IACTLK$\backslash X$ sentence ϕ as follows (connectives are clear and omitted):

$O, g \models \forall_{v_1,\ldots,v_n} : \psi$	iff	$O, g \models \psi[v_1 \mapsto a_1]\ldots[v_n \mapsto a_n]$ for all pairwise disjoint choices of $a_1, \ldots, a_n \in \mathbb{N}$.
$O, g \models (alv, i)$	iff	$\exists j\colon id(g.j) = i$
$O, g \models (p, i)$	iff	$O, g \models (alv, i)$ and $p \in \mathcal{V}(s_i(g))$;
$O, g \models \neg(p, i)$	iff	$O, g \not\models (p, i)$;
$O, g \models A(\psi_1 U \psi_2)$	iff	for every $\pi \in \Pi(g)$, for some $i \geq 0$ $O, \pi(i) \models \psi_2$ and for all $0 \leq j < i$, $O, \pi(j) \models \psi_1$;
$O, g \models A(\psi_1 R \psi_2)$	iff	for every $\pi \in \Pi(g)$, for all $i \geq 0$, if $O, \pi(j) \not\models \psi_1$, for all $0 \leq j < i$, then $O, \pi(i) \models \psi_2$;
$O, g \models K_i \psi$	iff	$O, g \models alv(i)$ and $O, g' \models \psi$ for all g' with $O, g' \models alv(i)$ and $s_i(g) = s_i(g')$;

We say that an OIS O satisfies an IACTLK$\backslash X$ sentence ϕ if $O, g_0 \models \phi$ and we denote this by $O \models \phi$. We further define $\top \triangleq (p, i) \vee \neg(p, i)$, $\bot \triangleq (p, i) \wedge \neg(p, i)$, $AF\phi \triangleq A(\top U \phi)$ with the usual meaning of "for all paths, ϕ eventually holds" and $AG\phi \triangleq A(\bot, \phi)$ standing for "for all paths, ϕ always holds". Above we define universal quantification only over pairwise disjoint choices of agents, so that each variable is mapped to a different concrete agent when evaluating the formula. Note this does not restrict the expressive power of the language; for instance, any formula $\phi(i, j)$ for which the assumption is lifted, can be expressed in our language by $\phi(i, i) \wedge \phi(i, j)$. Nevertheless, it enables the formalisation of properties of interest in a concise manner, as the following examples show.

For the train-gate controller we would like to check that whenever an agent is alive and in the tunnel, it knows that no other agent is also alive and in the tunnel. This is expressed by the following IACTLK$\backslash X$ formula:

$$\phi_1 \triangleq \forall_{i,j} : AG((in\_tunnel, i) \rightarrow K_i(\neg(in\_tunnel, j)))$$

where *in_tunnel* is an atomic proposition that holds in the states where the trains are in a tunnel.

We could also consider a pattern formation protocol [32, 35], where a swarm of drones aims to collaborate to form a pattern. If we assume a label *in_position* on the agent's states that expresses when the agent has reached its desired position then we can write the following sentence:

$$\phi_2 \triangleq \forall_i : AG((alv, i) \rightarrow K_i(AF((in\_position, i))))$$

This property expresses that when an agent is alive, then it knows that it will eventually reach its desired position.

We can now define the decision problem which will constitute the focus of this paper, i.e., determining whether an open system satisfies a given specification. We formalise this below.

Definition 2.9 (OMCP). Given an OIS O and an IACTLK$\backslash X$ formula ϕ, the *open model checking problem* (OMCP) is the decision problem of determining whether $O \models \phi$.

As typical of decision problems considered in parameterised verification [8], this problem is undecidable. We show this below.

Theorem 2.10. *The OMCP is undecidable.*

Proof sketch. A lossy counter machine [33] can be encoded as an OIS, by using the agents to store the values of the lossy counters and the environment to capture the state of the machine. Counters can be incremented by the environment by jointly performing an action with an agent that is not yet storing a value (if no such agent is available, the environment waits for one to join). The spontaneous decreasing of a counter is captured by agents leaving the system.

We can add an initialisation phase to our environment that starts the system off from an arbitrary state by first performing an arbitrary number of increment events on any of the counters and then transitioning non-deterministically to any of the machine states. Finally, we can label halting states. This means that we can write a specification that encodes the uniform termination problem (from an arbitrary state, the lossy counter machine will always reach a halting state) in IACTLK$\backslash X$. Thus, since this problem is known to be undecidable [27], it follows that the OMCP is also undecidable. □

3 VERIFICATION OF COLLECTIVE OMAS

In the previous section we showed that the verification of OMAS is undecidable in general. The aim of this and the next section is to identify noteworthy classes of OMAS which admit verification procedures.

We begin by studying a class of OMAS called collective open multi-agent systems. The term *collective* refers to decentralised systems whereby the identity of an agent as well as the precise number of agents participating in a joint action does not affect the outcome of the action on the agent's local state. Several applications in robot swarms operate in this manner [10]. For instance, a number of opinion formation protocols [13] involve robots communicating with neighbourhoods of bounded size in order to agree on the best choice of action. In these scenarios, it is sufficient to only consider a bounded number of actions in the transition rather than the entire (arbitrarily large) joint action.

To model this class of systems we introduce *collective open interpreted systems* (COIS). Informally, agents' local transitions in collective open interpreted systems depend on the state of the agent and the *collective action* performed by the system at the round. A collective action at a given round is the set of action identifiers performed at the round. For example if for a system of 3 agents the joint action at a given round is the tuple (a, b, a) the collective action in that round is $\{a, b\}$. Formally we define COIS as follows.

Definition 3.1 (COIS). A *collective open interpreted system* (COIS) is an OIS $O = \langle A, E, \mathcal{V} \rangle$ where the transition function of A satisfies the condition

$$t(l, a, X \cup \{a\}, a_E) = t(l, a, X, a_E)$$

for all values of l, a, X and a_E.

It should be clear that not all OIS are COIS. For example models of mutual exclusion protocols cannot be COIS as the execution depends on handshaking between more than one agent.

We now proceed to show that COIS, differently from OIS, have a decidable open model checking problem. To do so we translate this problem to the parameterised model checking problem from [20]. We first define the parameterised model checking problem in the context of open interpreted systems.

Definition 3.2 (PMCP). Given an OIS O and an IACTLK$\backslash X$ formula ϕ, the *parameterised model checking problem* (PMCP) is the decision problem of determining whether the following holds for all values of $n \in \mathbb{N}$:

$$O(n) \models \phi.$$

If this holds, then ϕ is said to be satisfied by the closed multi-agent system O; this is denoted by $O \models_c \phi$.

Note that if we omit the transitions corresponding to agents joining and leaving the system, COIS can be recast in the semantics of swarm systems defined in [20] from which we omit the notion of neighbourhoods. Since the PMCP for swarm systems is decidable [20], it follows from this mapping that the PMCP for COIS is also decidable. We will take advantage of this observation to define a procedure that solves the OMCP for COIS. To do this, we first define a translation of the OMCP into the PMCP.

Definition 3.3 (Transformed agent template). Let $A = \langle L, \iota, Act, P, t \rangle$ be an agent template for a COIS. We define the *transformed agent template* $\bar{A} = \langle \bar{L}, \bar{\iota}, \bar{Act}, \bar{P}, \bar{t} \rangle$ by:

- $\bar{L} \triangleq L \cup \{wait, dead\}$ where *wait* and *dead* are two new states encoding, respectively, that the agent has not joined the system yet, and that the agent has exited the system;
- $\bar{\iota} \triangleq wait$ representing that initial state of the agent.
- $\bar{Act} \triangleq Act \cup \{join, leave, null\}$ where *join* and *leave* are new actions that represent an agent joining and leaving the system, and *null* is a new action representing an agent that is not part of the system not doing anything;
- $\bar{P} : \bar{L} \to \mathcal{P}(\bar{Act})$ is given by:

$$l \mapsto \begin{cases} \{null, join\} & \text{if } l = wait \\ \{null\} & \text{if } l = dead \\ P(l) \cup \{leave\} & \text{otherwise} \end{cases}$$

- $t : \bar{L} \times \bar{Act} \times \mathcal{P}(\bar{Act}) \times Act_E \to \bar{L}$ is given by:

$$(l, a, X, a_E) \mapsto$$

$$\begin{cases} t(l, a, X, a_E) & \text{if } l \in L, a \in Act \text{ and } X \subseteq Act \\ \iota & \text{if } l = wait \text{ and } a = join \\ dead & \text{if } l = dead \text{ or } a = leave \\ l & \text{otherwise} \end{cases}$$

Note that as in Definition 2.5 agents that have left cannot rejoin. We will denote by $\bar{O}$ the transformed COIS obtained by replacing A by $\bar{A}$ in O. We now show that the PMCP for $\bar{O}$ is equivalent to the OMCP for O.

Theorem 3.4. *Let O be a COIS and ϕ an IACTLK$\backslash X$ formula. Then, $\bar{O} \models_c \phi$ iff $O \models \phi$.*

Proof sketch. ($\Rightarrow$) Suppose $\bar{O} \models_c \phi$ and assume for a contradiction that $O \not\models \phi$. So, there is some path ρ in the open system falsifying ϕ. By our restriction on $\Pi(g)$, this path has a finite number n of agents joining the system in it. Now, consider a path $\bar{\rho}$ in $\bar{O}(n)$ that performs all the same actions as ρ, with *join* and *leave* actions being used to replace the transitions of agents joining and leaving the system. This path satisfies at each state the same atomic propositions as ρ and thus also falsifies ϕ. But then $\bar{O}(n) \not\models_c \phi$, giving a contradiction as desired.

($\Leftarrow$) Suppose $O \models \phi$ and assume for a contradiction that $\bar{O} \not\models_c \phi$. Then, there is a path $\bar{\rho}$ in $\bar{O}(n)$ that falsifies ϕ. Similarly to before, we can define an equivalent path ρ in the open system O, by using an agent joining transition to simulate the *join* action and an agent leaving transition to simulate the *leave* action. Then $O \not\models \phi$, giving a contradiction as desired. □

Having proved this equivalence, we use this to show that the OMCP for COIS is decidable.

Theorem 3.5. *The OMCP for COIS is decidable.*

Proof sketch. Notice that by Theorem 3.4, if we want to check $O \models \phi$, we can instead construct $\bar{O}$ according to Definition 3.3 and check $\bar{O} \models_c \phi$. This can be done using the decision procedure described in [20]. □

4 VERIFICATION OF INTERLEAVED OMAS

In this section we introduce and study *interleaved open interpreted systems* (IOIS) to model and reason about asynchronous open MAS. IOIS are inspired by Interleaved Interpreted Systems, a semantics for asynchronous MAS [24], where only one local action is performed in a global transition at each time step. In particular, IOIS is a fragment of OIS in which only one action label appears in any single joint action. The agents that participate in the transition perform that same action, whereas the ones not participating in the transition perform a null ε action. Following [21], where communication patterns for interleaved interpreted systems with arbitrarily many agents were put forward, each action label pertains to one of three synchronisation schemes: asynchronous evolution, pairwise synchronisation between an agent and the environment, or global synchronisation between all the agents and the environment.

We now formally describe IOIS. We assume that the agents' action set can be partitioned into three types of actions: *asynchronous*, *agent-environment* and *global synchronous*. We write this as $Act = A \cup AE \cup AE' \cup GS$, where actions from $AE' \triangleq \{a' \mid a \in AE\}$ will be used to model agent-environment synchronisations (see below). Further, we assume that there is a *null* asynchronous action $\varepsilon \in A$ that is enabled at all the states where an agent-environment action is not. Any transition for this action does not result in a state change, i.e., $t(l, \varepsilon, X, a_E) = l$ for all values of l, X, and a_E. Similarly, we assume a set of agent-environment actions AE, a set of global synchronous actions GS, and a null environment action ε_E.

Given the above we now define the agents' and environment's transition functions so that exactly one action is performed at a time. The conditions we impose below ensure that the state of the system can only change when precisely one action is performed and, depending on the type of action and in correspondence with the synchronisation patterns described above, only certain parts of the system can change state.

Asynchronous. If the action is an asynchronous action, then the action is performed by precisely one agent and only this agent can change state. To encode this we require that the transition function satisfies the following condition:

C1 If $a \in A$, $t(l, a, X, a_E) = l'$ and either $X \neq \{\varepsilon\}$ or $a_e \neq \varepsilon_E$, then $l = l'$.

So, by (C1), an agent cannot transition to another state via an asynchronous action unless it is the only agent performing a non-null action. Recall that the null action never results in a state change, so as a result of this restriction only the agent performing the asynchronous action can change state.

Agent-environment. If the action is an agent-environment action, then the action is performed by exactly one agent in conjunction with the environment. The representation of this is given in two steps. In the first step the agent that will perform the action moves to a state enabling the action by performing the action's corresponding copy. As we explain below, this ensures that only one agent participates in the transition. It is expressed by the two conditions below.

C2 If $a \in P(l) \cap AE$ then for all $t(l', x, X, a_E) = l$ we have that $x = a'$.

C3 If $a' \in AE'$, $t(l, a', X, a_E) = l'$, and either $X \neq \{\varepsilon\}$ or $a_E \neq a$, then $l = l'$.

By (C2) a state enabling an agent-environment action a can only be reached via its corresponding a' action. By (C3) this transition can only occur if the environment is willing to perform a, and no other agent decides it wants to perform a. After transitioning to this intermediate state then the agent will insist on completing the action a. We encode this in the condition below.

C4 If $a' \in AE'$ and $t(l, a', \{\varepsilon\}, a) = l'$ then $P(l') = \{a\}$.

The (C4) condition forces the agent to perform the agent-environment action after transitioning to the intermediate state enabling it. However, we still need to enforce that if any other agents try to perform another action while an agent is in the intermediate state then the state of the system does not change state. This is captured by the two conditions below.

C5 If $a \in AE$, $t(l, a, X, a_E) = l'$ and either $X \neq \{\varepsilon\}$ or $a_E \neq a$, then $l = l'$.

C6 If $a \in AE$, $t_E(l_E, a, X) = l'_E$, $X \neq \{a, \varepsilon\}$, and $X \neq \{a\}$, then $l_E = l'_E$.

Here, (C5) enforces that the agent can only transition if the environment performs the same agent-environment action, and no other agent performs an action. In addition, (C6) enforces that the environment can only transition if an agent performs the same agent-environment action and no agents perform a different action.

Notice that the agent-environment action can only be performed from the intermediate state. So, as by the previous conditions at most one agent can be in this state at a given time, the action can only be performed by exactly one agent.

We now introduce the final condition for agent-environment actions, which will ensure the environment cannot know how many agents are in the system.

C7 For all l_E and $a \in AE$, $t_E(l_E, a, \{a, \varepsilon\}) = t_E(l_E, a, \{a\})$.

Here, (C7) enforces that the environment cannot transition differently upon performing an agent-environment action based on whether there is exactly one or several agents in the system.

Global synchronous. If the action is a global synchronous action, then the action is performed by all the agents and the environment (with every participant being able to change its state). We encode this by means of the following conditions.

C9 If $t(l, a, X, a_E) = l'$, $a \in GS$ and either $X \neq \{a\}$ or $a_E \neq a$, then $l = l'$.

C10 If $t_E(l_E, a, X) = l'_E$, $a \in GS$, and $X \neq \{a\}$, then $l_E = l'_E$.

Here, (C9) enforces that the agents can only change state if a was selected by all agents in the system. Similarly, (C10) enforces that the environment can only change state if this was the case.

Definition 4.1 (IOIS). An *interleaved open interleaved system* (IOIS) is an OIS that satisfies conditions (C1) - (C10).

Decision procedure. Despite the restrictions imposed, the proof for Theorem 2.10 still holds for IOIS. Thus, the OMCP for IOIS is in general undecidable. We will nonetheless explore a partial decision procedure for it in the rest of this section.

We begin by noting that IOIS extend parameterised interleaved interpreted systems (PIIS) [21] with the addition of transitions capturing agents joining and leaving the system. The PMCP for PIIS is undecidable. Nevertheless, decidable fragments have been exploited in a variety of AI-based applications [20, 21] via the development of sound albeit incomplete verification methods. We leverage on these ideas to introduce a verification procedure for IOIS. Specifically, we reduce the OMCP for IOIS to the PMCP for IOIS, and then solve this via existing decision procedures for the PMCP on PIIS [19, 21]. Formally, as in Definition 3.3, given an agent template A we define $\bar{A}$ by adding two additional states representing agents that have not yet joined the system and agents that have left the system, and introduce asynchronous actions *join* and *leave* for the agents to join and leave the system. We denote by $\bar{O}$ the IOIS obtained by replacing A by $\bar{A}$ in an IOIS O.

Theorem 4.2. *Let O be an IOIS and ϕ be an IACTLK$\setminus$X formula. Then, $\bar{O} \models_c \phi$ iff $O \models \phi$.*

Proof Sketch. The proof is similar to that of Theorem 3.4. Once again, for any path in O we first observe that this has a finite number n of agents joining transitions by our assumption. Then we can define an equivalent path in $\bar{O}(n)$ by using the new *join* and *leave* asynchronous actions to simulate an agent joining and leaving. Also, we observe that for any path in $\bar{O}(n)$ we can construct an equivalent path in O by simulating the asynchronous *join* and *leave* actions via agent joining and leaving transitions. The result follows from these observations. □

It follows that we can solve the OMCP by using existing techniques for solving the PMCP on PIIS. A useful notion for solving this decision problem is that of a cutoff. Informally, this is a number of agents that is sufficient to exhibit every possible behaviour of the system. We use a definition similar to that in [19].

Definition 4.3 (Cutoff). An integer $c \in \mathbb{Z}^+$ is said to be a *cutoff* for an OIS O if for all IACTLK$\backslash X$ formulas ϕ we have that $O(c) \models \phi$ iff $O(n) \models \phi$ for all $n \geq c$.

Cutoffs have been extensively studied and, while it is known that they do not exist in general, cutoff identification procedures exist for some classes of systems [19, 21]. In the classes for which a cutoff c can be identified, the PMCP can be solved by checking all concrete instances of the system of size up to c.

The above cutoff procedures can be applied to solve the OMCP for the transformed IOIS. However, even if the original IOIS O admits a cutoff, its transformed version $\bar{O}$ may not have a cutoff in general. To this end we identify a fragment of IOIS, which we call GS-compliant, for which cutoffs are preserved in the transformation.

Definition 4.4 (GS-compliant). We say an IOIS O is *GS-compliant* if it is the case that $P(\iota) = GS$ and for every $g \in GS$, there is some $s \in L \setminus \{\iota\}$ such that $\{s \in L | g \in P(s)\} = \{\iota, s\}$ and $t(\iota, g, \{g\}, g) = t(s, g, \{g\}, g)$.

So, in a GS-compliant IOIS precisely the global synchronous actions are enabled at the initial state and every global synchronous action is enabled in exactly two states with the two corresponding transitions sharing the same target. This restriction arises quite naturally in a number of scenarios. This is because global synchronous actions are most often used to encode different phases of a protocol [20]; this restriction corresponds precisely to new agents joining in at the correct phase.

We can now present a key result of this section.

Theorem 4.5. *Suppose $c \in \mathbb{Z}^+$ is a cutoff for an IOIS O, and that O is GS-compliant. Then c is also a cutoff for $\bar{O}$.*

Proof sketch. Notice that when an agent first joins a system with an agent already present, it cannot block any global transition from occurring, since all global synchronous actions are enabled in the initial state. Further, since only global synchronous actions are enabled at the initial state, the new agent has to wait in this state until it performs one such action. After performing this action, it will be in the same state as all the other agents. Thus, this ensures that the joining agent cannot enable any new behaviours of the system. This intuition can be used to build a stuttering simulation [21] and show the desired result. □

Algorithm 1 IOIS Decision Procedure

```
   Input: IOIS $O$, IACTLK$\backslash X$ formula $\phi$
   Output: Whether or not $O \models \phi$
 1: if ModelCheckClosed ($\bar{O}, \phi$) $\neq$ undefined then
 2:    return ModelCheckClosed ($\bar{O}, \phi$)
 3: end if
 4: $c \leftarrow$IdentifyCutoff($O$)
 5: if $c \neq$ undefined $\wedge$CheckGSCompliant ($O$) then
 6:    for $k \leftarrow 0$ to $c$ do
 7:       if $\bar{O}(k) \not\models \phi$ then
 8:          return false
 9:       end if
10:    end for
11:    return true
12: end if
13: return undefined
```

The combination of Theorem 4.2 and Theorem 4.5 gives us a sound but incomplete decision procedure for solving the OMCP on IOIS, shown in Algorithm 1. Given an IOIS O and a formula ϕ that we wish to check, we construct $\bar{O}$ and attempt to verify $\bar{O} \models_c \phi$ using existing incomplete procedures for parameterised model checking. If we can either verify that $\bar{O} \models_c \phi$ or $\bar{O} \not\models_c \phi$ then by Theorem 4.2 we are done. Otherwise, if we can identify a cutoff for O, e.g. by using existing cutoff identification procedures [19, 21] and also check that it is GS-compliant, then we know that this is also a cutoff for $\bar{O}$. We can use this to check $\bar{O} \models_c \phi$ by checking all concrete systems of increasing size up to the cutoff and return this as a result. If neither of the above checks passes, then the procedure does not return any result. Note that Algorithm 1 is agnostic with respect to the particular cutoff procedure used. It is therefore in general possible that a cutoff could be found for O and not for $\bar{O}$, depending on the choice of cutoff procedure.

We now show that this incomplete decision procedure is correct.

Corollary 4.6. *Algorithm 1 is a sound decision procedure for the OMCP.*

Proof. Suppose we return on line 2. Then, this return value is correct by Theorem 4.2. Now suppose we return on line 8. Then, it is clear that $\bar{O} \not\models_c \phi$ so once again the return value is correct by Theorem 4.2. Finally, suppose we return on line 11. Then, O is GS-compliant by the check on line 5. Thus, since c is a cutoff for O, it follows by Theorem 4.5 that it is also a cutoff $\bar{O}$. So, since we checked all systems $\bar{O}(k)$ for $k \leq c$ it follows that $\bar{O} \models_c \phi$ and, by Theorem 4.2 our return value is once again correct. □

5 EVALUATION

The techniques described in the previous sections were implemented into an experimental open toolkit called MCMAS-OP, which is released as open-source [28]. MCMAS-OP is composed of two parts.

The first part, MCMAS-OP$_{\texttt{COIS}}$, is a checker that extends MCMAS [25] by implementing the (previously purely theoretical and unimplemented) parameterised verification techniques described in [20]. These were further extended to model the open aspect of the system and support COIS by constructing the transformed template (as described in Definition 3.3) and using it to verify the specifications according to Theorem 3.4. The second part, MCMAS-OP$_{\text{IOIS}}$, is built on MCMAS-P [21] to support IOIS. As described in Section 4, MCMAS-OP$_{\text{IOIS}}$ constructs the transformed template and uses Theorem 4.2 to check the specification under analysis.

We tested our toolkit on two different examples, one for COIS (autonomous robots) and one for IOIS (train-gate controller).

Autonomous Robots. In the autonomous robots scenario [15, 21], an arbitrary number of robots move synchronously along a track of length n. For the purposes of testing our toolkit we fixed $n = 4$. Each robot is equipped with a sensor reporting its position with a potential error of up to 1 unit. The robot begins at the start of the track and at each time step may either choose to halt or to continue moving forward. The robot's goal is to stop within the target region (which we fixed to $\{2, 3\}$), and not enter the forbidden region (fixed to $\{4\}$). We encoded this scenario into a COIS, and then considered the following specifications:

$$\phi_1 \equiv \forall_x : AG\ ((goal\_region, x) \rightarrow AG((alv, x) \rightarrow (goal\_region, x)))$$

$$\phi_2 \equiv \forall_{x,y} : AG\ K_x\ AG\ \neg(forbidden\_region, y)$$

$$\phi_3 \equiv \forall_x : AG\ ((halted, x) \rightarrow (goal\_region, x))$$

The first property encodes the fact that once in the goal region, an agent stays in the goal region as long as it remains alive. The second property represents that every agent always knows that every other agent cannot ever be in the forbidden region. The third property encodes that an agent that has halted is in the goal region. We would expect all three properties to hold.

Train-Gate Controller. In the train-gate controller scenario [17], a number of trains wish to enter a tunnel and a controller has to ensure that only one train is in the tunnel at any given point. We encoded this as an IOIS, in a similar way to Example 2.7. In particular, the trains are encoded as agents, and the controller is encoded in an environment with two states: *red* representing that a train is in the tunnel, and *green* representing that the tunnel is empty. We considered the following properties:

$$\begin{aligned}\psi_1 \equiv \forall_x :\ & AG\ ((l_E = green) \rightarrow \neg(in\_tunnel, x) \\ & \wedge (in\_tunnel, x) \rightarrow (l_E = red)) \\ \psi_2 \equiv \forall_{x,y} :\ & AG\ ((in\_tunnel, x) \rightarrow \neg(in\_tunnel, y)) \\ \psi_3 \equiv \forall_x :\ & AG\ ((l_E = red) \rightarrow AF(l_E = green))\end{aligned}$$

The first property expresses that if the environment is in state *green*, then no train is in the tunnel and, conversely, if a train is in the tunnel then it is in state *red*. The second property expresses that two trains cannot be in the tunnel at the same time. The third property encodes the liveness property that if the light is red, it will always eventually turn green again. We would expect the first two properties to hold. However, the third should not hold since if an agent leaves the system while it is in the tunnel, this will cause the light to remain red forever.

Experimental Results. The results obtained when checking these two scenarios (with MCMAS-OP$_{\text{COIS}}$ and MCMAS-OP$_{\text{IOIS}}$, respectively) are recorded in Table 1. We also measured the time and memory usage to obtain these results on a machine with an i7-6700 processor and 16GB of RAM, running Linux kernel version 4.4.0. Note that both the protocols generate an unbounded state space and cannot be tackled using traditional verification techniques.

		Autonomous Robot (MCMAS-OP$_{\text{COIS}}$)	Train-Gate Controller (MCMAS-OP$_{\text{IOIS}}$)
	1	Satisfied	Satisfied
Property	2	Satisfied	Satisfied
	3	Satisfied	Not Satisfied
Model Build Time		30 sec	0.0023 sec
Memory Usage		199MB	31MB
Reachable States		1.49 million	45

Table 1: MCMAS-OP verification results obtained on our two examples, along with the build time, memory usage and number of reachable states for the model.

The results of the experiments confirmed our intuition on the satisfiability or unsatisfiability of the specifications in question on the models. The results confirmed that our toolkit can be used to analyse open multi-agent systems in cases where we have identified the problem to be decidable. For the simple scenarios we considered, the timing results obtained were attractive. No comparison to other toolkits is provided as we are not aware of other methods capable of verifying open multi-agent systems as we do here.

6 CONCLUSIONS

There has been considerable progress over the past decade on verification methods for multi-agent systems. This work has found application in a number of areas including services, robotics, autonomous systems, and security. One key limitation of this body of work concerns the assumption that the number of agents does not change during an execution of the system. As we argued in the Introduction, this is often an unrealistic assumption. Open systems such as auctions, negotiation protocols, robot swarms, etc., have as a key feature precisely the fact that agents may join or leave the system at run-time. With the exception of the work previously discussed [3], we are not aware of existing research in this area.

In this paper we have contributed towards a solution to this problem by defining a semantics for open multi-agent systems and their corresponding verification problem, which we called the open model checking problem. We showed that the problem is undecidable in general. We then proceeded to identify two noteworthy large classes of systems which admit verification procedures. As we discussed, these classes are of interest on their own and can model systems of practical value. We have implemented the procedure for checking the open verification problem into a toolkit and used it to assess specifications on MAS scenarios. The results demonstrated the correctness of the toolkit and its good performance. We are not aware of any other method or toolkit for assessing similar problems formally; so no comparison could be conducted with other tools.

In future work we intend to identify further decidable classes of OMAS, and further apply these results to swarm systems and auction systems. Another possible extension of this work is to consider stronger specification logics that incorporate aspects such as strategic behaviour [12, 29].

ACKNOWLEDGEMENTS

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

REFERENCES

[1] N. Alechina, M. Dastani, F. Khan, B. Logan, and J. J. Ch. Meyer. 2010. Using Theorem Proving to Verify Properties of Agent Programs. In *Specification and Verification of Multi-agent Systems*. Springer, 1–33.

[2] B. Bagheri, D. Calvanese, M. Montali, G. Giacomo, and A. Deutsch. 2013. Verification of Relational Data-centric Dynamic Systems with External Services. In *Proceedings of the 32nd Symposium on Principles of Database Systems (PODS13)*. ACM, 163–174.

[3] F. Belardinelli, D. Grossi, and A. Lomuscio. 2015. Finite Abstractions for the Verification of Epistemic Properties in Open Multi-Agent Systems. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI15)*. AAAI Press, 854–860.

[4] F. Belardinelli, P. Kouvaros, and A. Lomuscio. 2017. Parameterised Verification of Data-aware Multi-agent Systems. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI17)*. AAAI Press, 98–104.

[5] F. Belardinelli and A. Lomuscio. 2009. Quantified epistemic logic for reasoning about knowledge in multi-agent systems. *Artificial Intelligence* 173, 9–10 (2009), 982–1013.

[6] F. Belardinelli, A. Lomuscio, and F. Patrizi. 2012. An Abstraction Technique for the Verification of Artifact-Centric Systems. In *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning (KR12)*. AAAI Press, 319–328.

[7] F. Belardinelli, A. Lomuscio, and F. Patrizi. 2014. Verification of Agent-based Artifact Systems. *Journal of Artificial Intelligence Research* 51 (2014), 333–376.

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

[9] R. H. Bordini, M. Fisher, W. Visser, and M. Wooldridge. 2006. Verifying Multi-agent Programs by Model Checking. *Autonomous Agents and Multi-Agent Systems* 12, 2 (2006), 239–256.

[10] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo. 2013. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence* 7, 1 (2013), 1–41.

[11] D. Calvanese, M. Montali, and G. Delzanno. 2015. Verification of Relational Multiagent Systems with Data Types. In *Proceedings of the 19th AAAI Conference on Artificial Intelligence (AAAI15)*. AAAI Press, 2031–2037.

[12] K. Chatterjee, T. Henzinger, and N. Piterman. 2007. Strategy Logic. In *Proceedings of the 18th International Conference on Concurrency Theory (CONCUR07)*, Vol. 4703. 59–73.

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

[14] J. Ezekiel, A. Lomuscio, L. Molnar, and S. Veres. 2011. Verifying fault tolerance and self-diagnosability of an autonomous underwater vehicle. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI11)*. AAAI Press, 1659–1664.

[15] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. 1995. *Reasoning about Knowledge*. MIT Press, Cambridge.

[16] P. Gammie and R. van der Meyden. 2004. MCK: Model Checking the Logic of Knowledge. In *Proceedings of 16th International Conference on Computer Aided Verification (CAV04) (Lecture Notes in Computer Science)*, Vol. 3114. Springer, 479–483.

[17] W. van der Hoek and M. Wooldridge. 2002. Tractable multiagent planning for epistemic goals. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS02)*. ACM Press, 1167–1174.

[18] M. Kacprzak, W. Nabialek, A. Niewiadomski, W. Penczek, A. Pólrola, M. Szreter, B. Woźna, and A. Zbrzezny. 2008. VerICS 2007 - a Model Checker for Knowledge and Real-Time. *Fundamenta Informaticae* 85, 1 (2008), 313–328.

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

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

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

[22] P. Kouvaros and A. Lomuscio. 2017. Verifying Fault-tolerance in Parameterised Multi-Agent Systems. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI17)*. AAAI Press, 288–294.

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

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

[25] A. Lomuscio, H. Qu, and F. Raimondi. 2017. MCMAS: A Model Checker for the Verification of Multi-Agent Systems. *Software Tools for Technology Transfer* 19, 1 (2017), 9–30.

[26] A. Lomuscio, H. Qu, and M. Solanki. 2008. Towards verifying compliance in agent-based web service compositions. In *Proceedings of the 7th International Conference on Autonomous Agents and Multi-Agent systems (AAMAS08)*. IFAAMAS Press, 265–272.

[27] R. Mayr. 2003. Undecidable problems in unreliable computations. *Theoretical Computer Science* 297, 1 (2003), 337–354.

[28] MCMAS-OP. 2019. Model Checking Multi-Agent Systems (OPen), http://vas.doc.ic.ac.uk/software/extensions. (2019).

[29] F. Mogavero, A. Murano, and M. Vardi. 2010. Reasoning About Strategies. In *Proceedings of the 30th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS10)*, Vol. 8. Schloss Dagstuhl, 133–144.

[30] K. Namjoshi and R Trefler. 2015. Analysis of dynamic process networks. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 164–178.

[31] A. S. Rao. 1996. AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In *Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW96) (LNCS)*, Vol. 1038. Springer-Verlag, 42–55.

[32] E. Sahin, T. H. Labella, V. Trianni, J. L. Deneubourg, P. Rasse, D. Floreano, L. Gambardella, F. Mondada, S. Nolfi, and M. Dorigo. 2002. SWARM-BOT: pattern formation in a swarm of self-assembling mobile robots. In *IEEE International Conference on Systems, Man and Cybernetics*, Vol. 4. IEEE Press.

[33] P. Schnoebelen. 2010. Lossy Counter Machines Decidability Cheat Sheet. In *Proceedings of the 4th International Conference on Reachability Problems (RP10) (Lecture Notes in Computer Science)*, Vol. 6227. Springer, 51–75.

[34] P. Sewell. 2001. Pi Calculus. In *Formal Methods for Distributed Processing, A Survey of Object Oriented Approaches*, H. Bowman and J. Derrick (Eds.). Cambridge University Press, Chapter 9, 177–197.

[35] V. Trianni, R. Grob T. H. Labella, E. Şahin, M. Dorigo, and J. L. Deneubourg. 2012. *Modeling pattern formation in a swarm of self-assembling robots.* Technical Report TR/IRIDIA/2002-12. UniversitÃľ Libre de Bruxelles.

[36] M. Venkatraman and M. P. Singh. 1999. Verifying Compliance with Commitment Protocols. *Autonomous Agents and Multi-Agent Systems* 2, 3 (1999), 217–236.