A Petri net based reachability tool for open multi agent systems

Shabana AT, Alphy George and S Sheerazuddin

CSE Department, NIT Calicut, Kozhikode, 673601, Kerala, India

ARTICLE INFO

ABSTRACT

We study open multi-agent systems where agents may join or leave the system at runtime. Kouvaros et al. [1] have defined the verification problem for such systems and showed that it is undecidable, in general. Also, they have found one decidable class of open multi-agent systems and provided a partial decision procedure for another. In the same vein, we present a subclass of open MAS, called **Rutwiya** systems, wherein agents communicate exclusively (directly) with the environment and there is no direct communication among the agents. We design an encoding scheme for Rutwiya systems into Petri nets, which are a standard formalism for modeling and analysis of such unbounded distributed systems. This encoding provides a reduction from Rutwiya systems to Petri nets, thereby making reachability (and, therefore, safety checking) in Rutwiya systems decidable. We also develop a tool for reachability in Rutwiya systems that uses any of the already existing Petri net reachability tools, like KReach or ITS, as a back-end to solve the reachability problem in Rutwiya systems.

1. Introduction

Robotic systems represent tangible entities designed to engage with and navigate the physical world. These advanced systems seamlessly combine mechanical, electrical, and computational elements, offering a versatile platform for autonomous or semi-autonomous operations. Instances of robotic systems are found in various applications, such as collaborative robotic MAS in warehouse logistics, where multiple robots work together to optimize inventory management, transportation, and order fulfillment [2]. Another example is the use of swarm robotics in environmental monitoring, where a group of robots collaborates to collect data in large-scale outdoor areas [3].

Robotic systems comprising multiple robotic agents introduce challenges that extend beyond those encountered in the development of single-robot systems. Multi-robot systems can be effectively represented and modelled as multi-agent systems (MAS) [4]. Within the framework of multi-agent systems, each robot is considered an agent capable of sensing its environment, making decisions, and interacting with other agents to accomplish common goals. There are two main types of multi-robot systems: *swarms* [5], in which every robot has the same behaviour, and *MAS*, in which robot can have different behaviours. To achieve sophisticated, global behaviour, a swarm or MAS of robots in a multi-robot system often follow simple, local, nature-inspired protocols to communicate with each other and the environment [6].

Research interests in multi-agent systems (MAS) encompass diverse domains: formal verification to mathematically ensure system correctness [7], distributed decision-making strategies [8], swarm intelligence [5], system learning integration [9], using agent-based models to simulate and analyze the behaviour of complex systems [10] and so on.

✉ shabana_p200052cs@nitc.ac.in (S. AT);
alphy_p200059cs@nitc.ac.in (A. George); sheeraz@nitc.ac.in (.S. Sheerazuddin)
ORCID(s):

Several techniques have been put forth in the last decade to ensure that multi-agent systems (MAS) fulfill the requirements they are meant to. These approaches encompass verification techniques utilizing both model checking [11, 12, 13] and theorem proving [14, 15]. In traditional methods for modelling and verifying MAS, the number of agents will be fixed at the design time (*closed MAS*) to build and verify models [16, 13, 17, 14]. However, in several real-time applications of MAS, the number of agents cannot be known until runtime. A multi-agent system characterized by agents joining or departing from the system during runtime is referred to as *open MAS (OMAS)* [18, 19]. Prominent instances of OMAS involve scenarios like robotic swarms or MAS with agents capable of departing or experiencing malfunctions, auctions- where participants can dynamically enter or exit the bidding process, and, more broadly, any system where the count of agents is unbounded and fluctuates during runtime.

In Kouvaros et al. [1], they proposed a method for verifying OMAS, which consists of homogeneous agents. They introduced semantics for the analysis of OMAS based on interpreted systems, specifically Open Interpreted Systems (OIS). An OIS models the behavior of agents entering and leaving the system, alongside the environment that represents the rest of the system's state. They showed that given an OIS and a formula ϕ in temporal-epistemic logic, the open model checking problem (OMCP) is undecidable in general. This report aims to extend the semantics of OIS for MAS and describe a fragment called *Rutwiya Systems* for which reachability is shown to be decidable using a counter abstraction based encoding technique.

Observe that agents in open MAS are often simply *replications* of a fixed number of template (finite-state) systems defining the behavior of agents. One way to simplify the system is to take advantage of the regularity that occurs in the induced system model. Counter abstraction can be naturally applied in this context. The concept involves representing the global state of an open MAS as a set of counters, with one counter assigned to each local state. These counters

keep track of the current count of agents residing in the respective local state. Originally proposed as a method to achieve *symmetry reduction* for fixed-size systems [20], this approach transforms a *n*-process model, initially exponential in *n*, into one with a polynomial size in *n*. We follow the lead of [20] and abstract a subclass of open MAS as Petri nets to show that reachability is decidable for this subclass.

The next sections of the report are organized in the following manner. In Section 2, we provide a review of the syntax and semantics of open MAS. In Section 3, we define a new class of open MAS called Rutwiya Systems, and in Section 4 we show that the reachability of Rutwiya Systems is decidable by encoding it into Petri nets. In Section 5, we discuss the related work. In Section 6 we provide a conclusion and future directions.

2. Open MAS

The system in which agents with different behaviours enter or leave the system at runtime is known as *open MAS*. In this section, we review the syntax and semantics of Open Interpreted System (OIS), proposed by Kouvaros et al. [1]. OIS is based on interpreted systems, originally proposed by Fagin et al. [21] to provide computationally grounded semantics to model temporal and epistemic properties of multi-agent and distributed computer systems. Interpreted systems are defined over a set of agents $\{1, 2, \ldots, n\}$ with local states for each agent and the environment. A snapshot of the system is represented using a *global state* containing local states of each agent and the environment. Further, the evolution of the system over time can be modelled by the notion of *run*.

2.1. Syntax and Semantics of Open MAS

An agent template defines the behaviour of agents in the open MAS. This is given as follows.

Definition 1 (Agent). Agent template is a tuple $AT = \langle L, \iota, Act, P, tr \rangle$ where:

- $L \neq \emptyset$ is the set of local states of the agents.

- $\iota \in L$ is the unique initial local state of agents.

- Act is the non-empty set of actions the agents executes. Act contains a special symbol τ that corresponds to "no action".

- $P : L \to \mathcal{P}(Act)$ is a protocol function that gives the actions enabled at the current state for the agents. Note that $\tau \in P(l)$, for every $l \in L$.

- $tr : L \times Act \times \mathcal{P}(Act) \times Act_E \to L$ is the local transition function returning the next state of the agent from its current state based on the action executed by that agent, set of actions executed by other agents in the system and the action executed by the environment. Note, that for any $l \in L$, $tr(l, \tau, A, b) = l$, where $A \subseteq Act$ and $b \in Act_E$.

The environment behaviour is defined as follows.

Definition 2 (Environment). The environment is a tuple $E = \langle L_E, \iota_E, Act_E, P_E, tr_E \rangle$ where:

- $L_E \neq \emptyset$ is the set of local states of the environment and $L \cap L_E = \emptyset$.

- $\iota_E \in L_E$ is the unique initial state.

- $Act_E \neq \emptyset$ is a set of actions that the environment executes.

- $P_E : L_E \to \mathcal{P}(Act_E)$ is a protocol function that defines which actions are enabled at a given local state.

- $tr_E : L_E \times Act_E \times \mathcal{P}(Act) \to L_E$ is the transition function that gives the next state of the environment given its current state, the action it executes and the set of actions executed by the agents.

We combine the agent templates and environment to obtain open MAS as follows.

Definition 3 (OIS). An open interpreted MAS (OIS) is a $k + 1$-tuple $\mathcal{O} = \langle AT, E, V \rangle$ where:

- AT is the agent template,

- E is the environment, as defined above, and

- $V : L \to \mathcal{P}(AP)$ is a labelling function for the agent's local states where AP is a set of atomic propositions.

Now we define the semantics of open MAS. In any instance, the system (open MAS) comprises the environment and zero or more agents. Each agent is assigned an integer – its identity – which is unique to that particular agent. The identity is assigned when the agent joins the system, and it will not change until the agent leaves the system. The environment state along with the agent states and their corresponding identities, at any point of time, gives the snapshot (global state) of the system.

When n agents are present in the system, the set of tuples that characterize it at a particular instance of time is denoted by G_n where G_n is a set of $n + 1$ tuples of the form $(L \times \mathbb{Z}^+) \times \cdots \times (L \times \mathbb{Z}^+) \times L_E$ that gives the global state of the system. For a global state g, $g.j$ gives the local state of jth agent and $id(g.j)$ give the identity of jth agent, whereas $g.E$ gives the state of the environment. $G \triangleq \bigcup_{n \in \mathbb{N}} G_n$ is the set of all global states of any size.

For a given $n \in \mathbb{N}$, the set of joint actions $ACT_n = Act^n \times Act_E$. The global transition relation R_n, for a given n, depends on the projection of joint action into a set. A joint action in ACT_n is denoted by a and the projection of joint action into a set as $\dot{a} \triangleq \{a.j \mid a.j$ is the action of agent j in the joint action a and $1 \leq j \leq n\}$. For a given $i \in [n]$, we define $\dot{a}_{-i}$ – the set of actions in a except the action of agent i – as follows: $\dot{a}_{-i} \triangleq \{a.j \mid 1 \leq j < i\} \cup \{a.j \mid i < j \leq n\}$. Note, $\dot{a}_{-i}$ may include $a.i$.

We now define the temporal evolution of the global states. First, we define the global transition relation of a

fixed-size open MAS. Then, we extend the global transition relation for the open MAS with agents leaving and joining the system at runtime.

Definition 4 (The global transition relation of size n).
$R_n \subseteq G_n \times G_n$ on a set G_n of global states is defined by $(g, g') \in R_n$ iff there is some $a \in ACT_n$ such that the following hold:

- $a.E \in P_E(g.E)$ and for all $j \in [n], a.j \in P(g.j)$ where $a.E$ is the environment action and P is the protocol function for the agent.

- $g'.E = tr_E(g.E, a.E, \dot{a})$

- for all $j \in [n]$ it is the case that $id(g'.j) = id(g.j)$ and $g'.j = tr(g.j, a.j, \dot{a}_{-j}, a.E)$, for the jth agent in the system.

The first condition ensures that the action of both environment and agents are enabled as per the protocol functions of the environment and agents, respectively. The second condition guarantees that the transition of the environment is consistent with respect to t_E. The last condition ensures that the agent's identity remains the same after the global transition and each agent's transition is consistent with respect to the corresponding transition relation t_i.

Definition 5 (Global transition relation). The global transition relation $R \subseteq G \times G$ on the set G of global states is defined by $(g, g') \in R$ iff any of the following conditions are met:

> **Joint action:** There is $n \in \mathbb{N}$ such that $(g, g') \in R_n$ This corresponds to a global transition of the system of size n, already defined above.

> **Agent joining:** (i) There is $n \in \mathbb{N}$ such that $g \in G_n$ and $g' \in G_{n+1}$. Also, let $\mathbb{D}$ be the set of agent identities that occur in g. (ii) For all $j \in [n]$ it is the fact that $g.j = g'.j$ and $g.E = g'.E$. (iii) $g'.(n+1) = \iota$ where ι is the initial local state of the agent and $id(g'.(n+1)) \in \mathbb{N} \setminus \mathbb{D}$.

> **Agent leaving:** (i) There is $n \in \mathbb{N}$ such that $g \in G_n$ and $g' \in G_{n-1}$; (ii) There is $\kappa \in [n]$ such that $g' = g_{-\kappa}$, where $g_{-\kappa}$ denotes the tuple resulting from removing the κth component from the tuple g.

Now, a model of OIS may be defined.

Definition 6 (Model). The associated model of an OIS $\mathcal{O} = \langle AT, E, V \rangle$ 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 $g_0 = \langle \iota_E \rangle$ and $\mathcal{V} : G \longrightarrow \mathcal{P}(AP \times \mathbb{Z}^+)$ is the labelling function defined for a set of atomic propositions AP as $(p, i) \in \mathcal{V}(g)$ iff there is j with $id(g.j) = i$ and $p \in V(g.j)$. Also, let $Runs(\mathcal{O})$ be the set of all runs of O starting from the initial global state g_0.

3. Rutwiya Systems

A subclass of open MAS called **Rutwiya Systems** is defined in this section. Later, in Section 4, we show that reachability (and, therefore, safety checking) in this newly defined class is decidable.

In Rutwiya Systems, agents can leave the system at the time of agent leaving transition, but only from a fixed **leaving state**. Further, one or more agents can join in the initial state, at the time of agent joining transition. We can modify the agent template as follows: $AT = \langle L, \iota, Act, P, tr, leave \rangle$ where $leave \in L$ is the leaving state. Also, we insist that $P(leave) = \emptyset$, that is, no action is possible from the leave state. Similarly, for the global transition relation R, the agent joining and agent leaving can be redefined as:

- **Agent joining:** (i) There is $n \in \mathbb{N}$ and $r > 0$ such that $g \in G_n$ and $g' \in G_{n+r}$; (ii) For all $j \in [n]$ it is the case that $g.j = g'.j$. Also $g.E = g'.E$; (iii) For all j such that $1 \leq j \leq r$, $g'.(n+j) = \iota$ where ι is the initial local state of the agent and $id(g'.(n+j))$'s are from the set $\mathbb{N} \setminus \mathbb{D}$, where $\mathbb{D}$ is the set of agent identities occurring in g.

- **Agent leaving:** (i) There is $n \in \mathbb{N}$ and $r > 0$ such that $g \in G_n$ and $g' \in G_{n-r}$; (ii) There is $i \in \mathbb{N}$ such that $g.i = leave$ and $g' = g_{-\{i\}}$, where $g_{-\{i\}}$ denotes the tuple resulting from removing the ith components from g.

Due to the fact that $P(leave) = \emptyset$, when an agent reaches its leave state it is either removed from the system or it continues to remain in the *leave* state. Now, we proceed to show the decidability of reachability (and safety checking) in Rutwiya Systems by encoding them into Petri Nets. Note that we can have more that one leave states in an agent without affecting the decidability of Rutwiya Systems, but we do not consider the case here for the sake of simplicity.

3.1. Inspection Robots - An Example of Rutwiya Systems

Inspection robots play a critical role in various industries, ensuring the safety and integrity of infrastructure in hazardous environments. The need for effective verification and validation of these systems has been highlighted in numerous studies, such as in the work by Fisher et al. [22], which outlines the challenges faced by inspection robots in industries ranging from oil and gas to nuclear power. Within this context, the application of Rutwiya Systems becomes particularly relevant.

In a regular inspection scenario, robots must operate with a designated leave state, meaning they can only exit the operational area after reaching this state, ensuring that all necessary inspections are completed. For example, in nuclear waste management, a MAS of heterogeneous robots—including aerial drones, ground-based radiation monitors, and wall-climbing inspectors—must coordinate their actions through implicit communication. This coordination, or joint action, means that each robot's decisions

are influenced by the actions of others. For instance, a ground robot might only initiate radiation scanning if it detects that the aerial drone has signaled the structural integrity of the area. Such scenarios underscore the importance of managing these complex, interdependent actions efficiently, which is where the concept of Rutwiya Systems proves invaluable.

4. Decidability of Rutwiya Systems

We first revisit the notion of Petri nets. A Petri net N is defined formally as a 5-tuple $N = (P, T, F, W, M_0)$ where

- P is a finite set of **places**

- T is a finite set of **transitions**

- F is a **flow relation** over places and transitions, $F \subseteq (P \times T) \cup (T \times P)$,

- W is the **weight function**, $W : F \longrightarrow \mathbb{N}$ and

- $M_0 : P \longrightarrow \mathbb{N}$ is the **initial marking**.

A firing sequence of Petri net with initial marking M_0 is a sequence of transitions

$$\vec{\sigma} = \langle t_1 \cdots t_n \rangle$$

such that there exist a run:

$$M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \cdots \xrightarrow{t_{n-1}} M_{n-1} \xrightarrow{t_n} M_n.$$

The set of firing sequence is denoted as $\mathcal{L}(N)$. A marking M is reachable from M_0 if there exist a sequence of firing that transforms M_o to M. A set of reachable marking for N is denoted as $R(N)$. The reachability graph of N is denoted as G and it consists of nodes as reachable markings and edges as transitions. Also, we define the set of all possible markings for N as follows: $\mathbb{M} = \{ M \mid M : P \longrightarrow \mathbb{N} \}$

4.1. Encoding of Rutwiya Systems into Petri nets

Given a Rutwiya System $\mathcal{O} = \langle A, E, V \rangle$, we define an encoding of $\mathcal{O}$ as an *interpreted* Petri net $N_{\mathcal{O}} = (P_{\mathcal{O}}, T_{\mathcal{O}}, F_{\mathcal{O}}, W_{\mathcal{O}}, M_{\mathcal{O}}^0, \mathcal{V}_{\mathcal{O}})$ such that for every run ρ in $\mathcal{O}$ there is a corresponding run ρ in the encoding $N_{\mathcal{O}}$ and vice versa. We proceed to describe the various components of $N_{\mathcal{O}}$.

$P_{\mathcal{O}} = L \cup L_E$; For every location in agent template and environment template, it has a corresponding place in $N_{\mathcal{O}}$.

$M_{\mathcal{O}}^0 : P_{\mathcal{O}} \longrightarrow \mathbb{N}$ such that $M_{\mathcal{O}}^0(\iota_E) = 1$ and for all $l \in (P_{\mathcal{O}} - \{\iota_E\})$, $M_{\mathcal{O}}^0(l) = 0$; Initially there is a token in the place corresponding to ι_E, whereas all other places are empty.

We initialize $T_{\mathcal{O}}, F_{\mathcal{O}}, W_{\mathcal{O}}$ as follows:

- $T_{\mathcal{O}} = \{t_{aj}, t_{al}\}$; t_{aj} is the transition in the net corresponding to agent joining action & t_{al} is the transition corresponding to agent leaving action.

- $F_{\mathcal{O}} = \{\langle t_{aj}, \iota \rangle, \langle leave, t_{al} \rangle\}$; t_{aj} is a source transition that adds tokens to the place ι whereas t_{al} is a sink transition that removes tokens from the place $leave$.

- $W_{\mathcal{O}} = \{\langle t_{aj}, \iota \rangle \mapsto 1, \langle leave, t_{al} \rangle \mapsto 1\}$

Now for every $(l_E, b, A, l'_E) \in tr_E$, we augment $T_{\mathcal{O}}, F_{\mathcal{O}}, W_{\mathcal{O}}$ as follows:

Let $A = \{a_1, a_2, \ldots, a_k\}$ be the set of all agent actions executed in a joint action corresponding to the aforementioned environment transition.

We enumerate all location-action tuples for the pair $\langle A, b \rangle$ as follows:
$$\mathcal{LA}_{\langle A, b \rangle} = \Big(\prod_{i \in k} \mathcal{P}(P^{-1}(a_i) \times a_i) \setminus \emptyset \Big) \times (P_E^{-1}(b) \times b)$$
where $P^{-1}(a_i)$ is the set of agent locations with a_i being enabled, and $P_E^{-1}(b)$ is the set of environment locations wherein b is enabled.

Now, for each location-action tuple
$\mathfrak{la} = \langle (l_1, \mathfrak{a}_1), \ldots, (l_\Bbbk, \mathfrak{a}_\Bbbk), (l_e, b) \rangle \in \mathcal{LA}_{\langle A, b \rangle}$,
we define a set of transition labels
$\mathcal{TR}_{\mathfrak{la}} = \langle (l_1, \mathfrak{a}_1, A_1, l'_1), \ldots, (l_\Bbbk, \mathfrak{a}_\Bbbk, A_\Bbbk, l_{\Bbbk'}), (l_e, b, l'_e) \rangle$
where

- $l'_1 = tr(l_1, \mathfrak{a}_1, A_1, b), \ldots, l'_\Bbbk = tr(l_\Bbbk, \mathfrak{a}_\Bbbk, A_\Bbbk, b)$ such that for all $1 \le i \le \Bbbk$, $A_i = A$ or $A_i = A - \{\mathfrak{a}_i\}$

- $l'_e = tr_E(l_e, b, A)$.

Note, that $\{\mathfrak{a}_1, \ldots, \mathfrak{a}_\Bbbk\} = A$ and $\Bbbk \ge k$.
$\mathcal{TR}_{\langle A, b \rangle} = \bigcup_{\mathfrak{la} \in \mathcal{LA}_{\langle A, b \rangle}} \mathcal{TR}_{\mathfrak{la}}$ collects together all the transition labels of this kind. Thereafter, we define $T_{\langle A, b \rangle} = \{t_{\mathfrak{tr}} \mid \mathfrak{tr} \in \mathcal{TR}_{\langle A, b \rangle}\}$ as the set of transitions in $N_{\mathcal{O}}$ contributed by $\langle A, b \rangle$ and, therefore, added to $T_{\mathcal{O}}$.

For every $\mathfrak{tr} \in \mathcal{TR}_{\langle A, b \rangle}$, let the corresponding **pre-transition set** $\mathfrak{pre}(\mathfrak{tr})$ and **post-transition set** $\mathfrak{post}(\mathfrak{tr})$ be defined as:

$$\mathfrak{pre}(\mathfrak{tr}) = \{l_1, l_2, \ldots, l_{\Bbbk'}, l_e\}$$

$$\mathfrak{post}(\mathfrak{tr}) = \{l'_1, l'_2, \ldots, l'_{\Bbbk''}, l'_e\}$$

where $\Bbbk' \& \Bbbk'' \le \Bbbk$. $\mathfrak{pre}(\mathfrak{tr}) \, \mathfrak{post}(\mathfrak{tr})$ give the set of all **pre-transition places** and **post-transition places** for the corresponding joint action, respectively.

We add a set $\mathfrak{f}^{\mathfrak{tr}}_{\langle A, b \rangle}$ of (flow) edges to $F_{\mathcal{O}}$ as follows:
$\mathfrak{f}^{\mathfrak{tr}}_{\langle A, b \rangle} = \{(l, t_{\mathfrak{tr}}) \mid l \in \mathfrak{pre}(\mathfrak{tr})\} \cup \{(t_{\mathfrak{tr}}, l') \mid l' \in \mathfrak{post}(\mathfrak{tr})\}$.

For any $l \in \mathfrak{tr}$, let $n_l^{\mathfrak{tr}}$ and $n_{l'}^{\mathfrak{tr}}$ be the number of times l and l' occurs in the tuple $\mathfrak{tr}$. Then, we may add the following set to weight function $W_{\mathcal{O}}$:
$\mathfrak{w}^{\mathfrak{tr}}_{\langle A, b \rangle} = \{(l, t_{\mathfrak{tr}}) \mapsto n_l^{\mathfrak{tr}} \mid l \in \mathfrak{pre}(\mathfrak{tr})\} \cup \{(t_{\mathfrak{tr}}, l') \mapsto n_{l'}^{\mathfrak{tr}} \mid l' \in \mathfrak{post}(\mathfrak{tr})\}$.
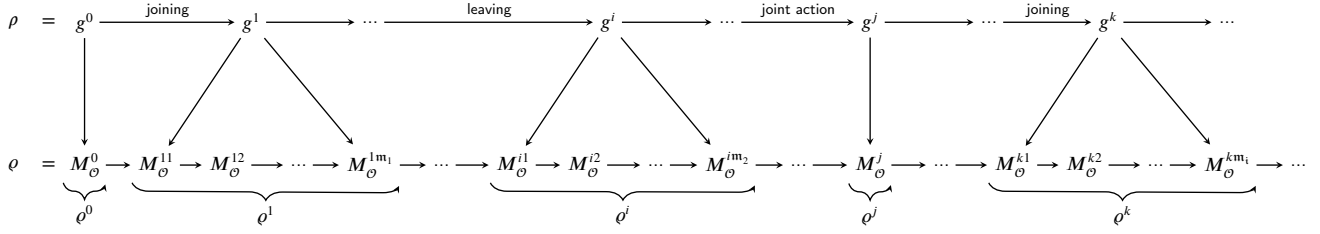
Figure 1: Pictorial representation of construction of ϱ from ρ

$\mathcal{V}_{\mathcal{O}}$, the *interpretation function*, is the same as V.

Definition 7. $mf_1 : G \longrightarrow M_{\mathcal{O}}$ such that for any $g \in G$, there is a $M \in \mathbb{M}_{\mathcal{O}}$ where for each $l \in L$, $M(l) =$ number of times l occurs in g and $M(l_e) = 1$.

Definition 8. $mf_2 : M_{\mathcal{O}} \longrightarrow G$ such that for any $M \in \mathbb{M}_{\mathcal{O}}$, there is a $g \in G$ where for each $l \in L$, $M(l) =$ number of times l occurs in g and $l_e = M(l_e) = 1$.

Now, we assert the following claim:

Claim 1. *There is a map* $\mathfrak{h}_1 : Runs(\mathcal{O}) \longrightarrow Runs(N_{\mathcal{O}})$ *such that for every* $g \in G$ *if there is a run* $\rho \in Runs(\mathcal{O})$ *from* g_0 *to* g *then there is a run* $\varrho \in \mathfrak{h}_1(\rho)$ *from* $M_{\mathcal{O}}^0$ *to* $M = mf_1(g)$.

Proof: Let $\rho = g^0 g^1 g^2 \ldots g^i \ldots g^{\mathbb{m}}$ be the run of $\mathcal{O}$, where $g^0 = g_0$ and $g^{\mathbb{m}} = g$. For every $0 \leq i \leq \mathbb{m}$, we map g^i to a sequence of configurations(markings) in $N_{\mathcal{O}}$, $\varrho^i \in Conf_{\mathcal{O}}^+$, as shown in the Figure 1. The concatenation of all such ϱ^i's gives us a legal run $\varrho^0 \cdot \varrho^1 \cdot \varrho^2 \cdots \varrho^i \cdots \varrho^{\mathbb{m}} = \varrho \in Runs(N_{\mathcal{O}})$.

We construct ϱ inductively:

$n = 0$: $\varrho^0 = M_{\mathcal{O}}^0$ where $M_{\mathcal{O}}(\iota_E) = 1$.

$n > 0$: Assume that ϱ^i is well defined for all $i < n$. Suppose $M_{\mathcal{O}}^*$ is the last configuration in the sequence ϱ^{n-1}. We need to do the following, for some $\mathfrak{m}_n$ that we compute later: define $\varrho^n = M_{\mathcal{O}}^1 M_{\mathcal{O}}^2 \ldots M_{\mathcal{O}}^{\mathfrak{m}_n}$ such that $M_{\mathcal{O}}^* \longrightarrow M_{\mathcal{O}}^1$ and for all $j \in \{1, 2, \ldots, \mathfrak{m}_n - 1\}$, $M_{\mathcal{O}}^j \longrightarrow M_{\mathcal{O}}^{j+1}$.

As mentioned above, we define ϱ^n as a sequence of configurations $M_{\mathcal{O}}^1 M_{\mathcal{O}}^2 \ldots M_{\mathcal{O}}^{\mathfrak{m}_n}$. We consider the three cases depending on the type of the transition (g^{n-1}, g^n) and define $M_{\mathcal{O}}^j$ for all $j \in \{1, 2, \ldots, \mathfrak{m}_n\}$:

- (g^{n-1}, g^n) is an **agents joining transition**: Let r' be the number of times ι occurs in g^{n-1} and r'' be the number of times ι occurs in g^n. Let $r = r'' - r'$. Let $\mathfrak{m}_n = r$. First we fix $\mathfrak{t}^j = t_{aj}$, for all $j \in \{1, 2, \ldots, \mathfrak{m}_n\}$. Thereafter, we use Algorithm 1 to compute $M_{\mathcal{O}}^j$ for all $j \in \{1, 2, \ldots, \mathfrak{m}_n\}$.

- (g^{n-1}, g^n) is an **agents leaving transition**: Let r' be the number of times *leave* occurs in g^{n-1} and r'' be the number of times *leave* occurs in

---

Algorithm 1: $M_{\mathcal{O}}^j$ for ϱ^n (Agent joining)

1   $j \leftarrow 1$
2   **while** $\mathfrak{m}_n > 0$ **do**
3     $M_{\mathcal{O}}^j(\iota) = M_{\mathcal{O}}^{j-1}(\iota) + 1$
4     $\forall l \in L - \{\iota\}, M_{\mathcal{O}}^j(l) = M_{\mathcal{O}}^{j-1}(l)$
5     $j \leftarrow j + 1$
6     $\mathfrak{m}_n \leftarrow \mathfrak{m}_n - 1$

g^n. Let $r = r'' - r'$. Let $\mathfrak{m}_n = r$. First, we fix $\mathfrak{t}^j = t_{al}$, for all $j \in \{1, 2, \ldots, \mathfrak{m}_n\}$, as in the case of agent joining. Thereafter, we use Algorithm 2 to compute $M_{\mathcal{O}}^j$, for all $j \in \{1, 2, \ldots, \mathfrak{m}_n\}$.

Algorithm 2: $M_{\mathcal{O}}^j$ for ϱ^n for ϱ^n (Agent Leaving)

1   $j = 1$
2   **while** $\mathfrak{m}_n > 0$ **do**
3     $M_{\mathcal{O}}^j(leave) = M_{\mathcal{O}}^{j-1}(leave) - 1$
4     $\forall l \in L - \{leave\}, M_{\mathcal{O}}^j(l) = M_{\mathcal{O}}^{j-1}(l)$
5     $j \leftarrow j + 1$
6     $\mathfrak{m}_n \leftarrow \mathfrak{m}_n - 1$

- (g^{n-1}, g^n) is a **transition with joint action** a^n: Let $g^{n-1}, g^n \in G_z$. That is there are z agents in the system at the time of joint action. Let q be the total number of local states of agents in the system. For every $1 \leq i \leq q$, r_i be the number of times l_i occurs in g^{n-1} and r_i' be the number of times l_i occurs in g^n. Let $\mathfrak{r}_i = r_i' - r_i$. In the case of joint actions, $\mathfrak{m}_n = 1$. Algorithm 3 is designed to compute marking $M_{\mathcal{O}}$ from g and g^{n-1}. Now we can select a $\mathfrak{t} \in T_{\mathcal{O}}$ such that $\mathfrak{t} = \mathfrak{t}_{\langle g^{n-1}.i, a^n.i, \dot{a}^n_i, g^n.i \rangle \mid i \in [z] \cup \langle g^{n-1}.E, a^n.E, g^n.E \rangle}$. Note that, if $a^n.i = \tau$ then there wont be any corresponding element in $\mathfrak{t}$.

The way ϱ is constructed, as defined above, makes sure that it is a run in $N_{\mathcal{O}}$.

We complete the proof of Claim 1 by proving the following inductive invariant:

$\forall n : 0 \leq n \leq \mathbb{m}, \forall l \in L$, *Number of agents in local state l in* g^n = *Value of $M_{\mathcal{O}}(l)$ in* $M_{\mathcal{O}}$.

Algorithm 3: $\mathfrak{c}$ for ϱ^n (Joint action)

1 **for** i in q **do**
2   **if** $\mathfrak{r}_i > 0$ **then**
3      $M_{\mathcal{O}}[l_i] = M_{\mathcal{O}}^*[l_i] + \mathfrak{r}_i$
4   **if** $\mathfrak{r}_i = 0$ **then**
5      $M_{\mathcal{O}}[l_i] = M_{\mathcal{O}}^*[l_i]$
6   **if** $\mathfrak{r}_i < 0$ **then**
7      $M_{\mathcal{O}}[l_i] = M_{\mathcal{O}}^*[l_i] - \mathfrak{r}_i$

Proof of invariant: By induction on n

n=0: Trivially holds as $g^0 = \langle \iota_E \rangle$, i.e., there are no agents in the system initially. Hence, all the number of tokens in $M_{\mathcal{O}}^0$ are set to zero whereas $M_{\mathcal{O}}(\iota_E) = 1$.

n>0: Inductively, let us assume that invariant holds for $i = n - 1$. We consider the three cases looking at the type of transition (g^{n-1}, g^n):

  agent joining: By inductive hypothesis, $\forall l \in L$, number of agents in local state l in $g^{n-1} = $ number of tokens in $M_{\mathcal{O}}(l)$ in $M_{\mathcal{O}}^{n-1}$. Note that in line number 3 of Algorithm 1, we make sure to increment number of tokens corresponding to ι state by this value, while rest of the local state tokens remain unchanged from $M_{\mathcal{O}}^{n-1}$ to $M_{\mathcal{O}}^n$. Therefore, the invariant holds for $i = n$.

  agent leaving: By inductive hypothesis, $\forall l \in L$, number of agents in local state l in $g^{n-1} = $ number of tokens in $M_{\mathcal{O}}(l)$ in $M_{\mathcal{O}}^{n-1}$. Note that in line numbers 3 of Algorithm 2, we make sure to decrement tokens corresponding to $leave$ state by this value, while rest of the local state tokens remain unchanged from $M_{\mathcal{O}}^{n-1}$ to $M_{\mathcal{O}}^{n-1}$. Therefore, the invariant holds for $i = n$.

  joint action: By inductive hypothesis, $\forall l \in L$, number of agents in local state l in $g^{n-1} = $ number of tokens in $M_{\mathcal{O}}(l)$ in $M_{\mathcal{O}}^{n-1}$. Consider an arbitrary $l \in L$, which could be an initial state or leave state. As already mentioned, $\mathfrak{r}$ is the difference between the number of times l occurs in g^n and g^{n-1}. Note that this $\mathfrak{r}$ will match with the corresponding $\mathfrak{r}$ in the definition of joint action in $\Delta_{\mathcal{O}}$. In Algorithm 3, we make sure that $M_{\mathcal{O}}(l_i)$ is decremented $\mathfrak{r}$ by if $\mathfrak{r} < 0$, and $M_{\mathcal{O}}(l_i)$ is incremented $\mathfrak{r}$ by if $\mathfrak{r} > 0$, and remains unchanged if $\mathfrak{r} = 0$. Therefore, $\mathfrak{r}$ is the difference between number of tokens of $M_{\mathcal{O}}(l_i)$ in $M_{\mathcal{O}}^n$ and $M_{\mathcal{O}}^{n-1}$. Hence, the invariant holds for $i = n$.

$\square$

Claim 2. *There is a map* $\mathfrak{h}_2 : Runs(N_{\mathcal{O}}) \longrightarrow Runs(\mathcal{O})$ *such that for every* $M \in \mathcal{M}_{\mathcal{O}}$ *if there is a run* $\varrho \in Runs(N_{\mathcal{O}})$ *from* $M_{\mathcal{O}}^0$ *to* M *then there is a run* $\rho \in \mathfrak{h}_2(\varrho)$ *from* g_0 *to* $g = mf_2(M)$.

Proof: Let $\varrho = M^0 \xrightarrow{t_0} M^1 \xrightarrow{t_1} M^2 \xrightarrow{t_2} \ldots M^i \ldots M^{\mathbb{m}} = M$ be a run of $N_{\mathcal{O}}$ where M^0 is the initial marking of $N_{\mathcal{O}}$, where $M^0(\iota_E) = 1$. For every $0 \le i \le \mathbb{m}$, we map a sequence of configurations, $\varrho^i \in Conf_{\mathcal{O}}^+$ to g^i as shown in the Figure 2. We can split the run ϱ to find such sequences.

Algorithm 4 gives us one particular split $\varrho^0, \varrho^1, \ldots, \varrho^i, \ldots \varrho^{\mathbb{m}}$, where $\mathbb{m} \le \mathbb{m}$. The split is primarily based on the transition t. The sequence of M^i's where the transition $t = t_{aj}$ are added to ϱ^i, for the given i in consideration. Similarly, sequences where $t = t_{al}$ are also added to ϱ^i. Otherwise, the algorithm directly adds the M^i to ϱ^i.

Algorithm 4: Splitting the run ϱ

1 $\varrho^0 = M_{\mathcal{O}}^0$
2 $j \leftarrow 0$
3 **for** i in $\mathbb{m}$ **do**
4   **if** $t = t_{aj}$ **then**
5      $j \leftarrow j + 1$
6   **while** $t = t_{aj}$ **do**
7        $\varrho^j \leftarrow \varrho^j.M^i$
8        $i \leftarrow i + 1$
9   **else if** $t = t_{al}$ **then**
10      $j \leftarrow j + 1$
11   **while** $t = t_{al}$ **do**
12        $\varrho^j \leftarrow \varrho^j.M^i$
13        $i \leftarrow i + 1$
14   **else**
15      $\varrho^j \leftarrow M^i$
16    $i \leftarrow i - 1$

We construct $\rho = g^0 \cdot g^1 \cdots g^i \cdots g^{\mathbb{m}}$ from the split $\varrho^0, \varrho^1, \ldots, \varrho^i, \ldots \varrho^{\mathbb{m}}$ inductively as follows:
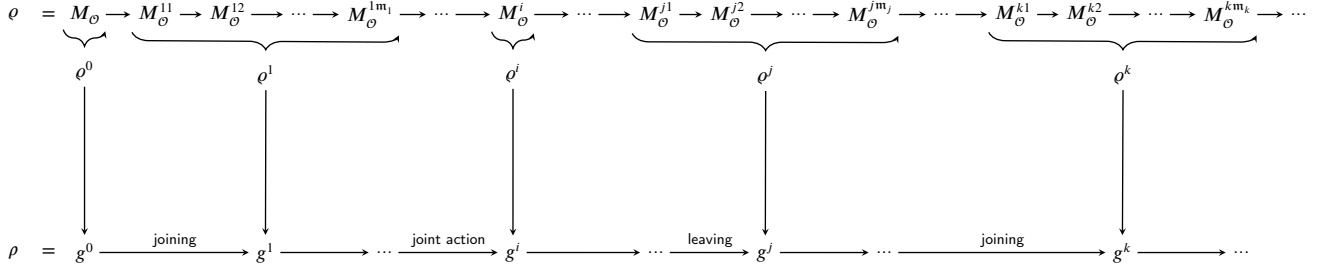
$n = 0$: $g^0 = \langle l \rangle$ where $M_{\mathcal{O}}^0(l) = 1$

$n > 0$: Assume that g^i is well defined for all $i < n$. For all $0 \le i < n - 1$, $(g^i, g^{i+1}) \in R$. We need to do the following: define g^n such that $(g^{n-1}, g^n) \in R$.

  We define $g^n = \langle (L \times \mathbb{Z}^+), (L \times \mathbb{Z}^+) \ldots, (L \times \mathbb{Z}^+), l_E \rangle$ as follows, with $l_E = l$ where $M_{\mathcal{O}}(l) = 1$ and $l \in L_E$. We consider three cases, depending on the type of transition in the first configuration of the sequence ϱ^n:

  t_{aj} : Let M_p and M_q be the starting and ending configurations of ϱ^n. Let $r = M_q(\iota) - M_p(\iota)$. Then g^n is constructed from g^{n-1} and ϱ^n by adding tuples in the way described in Algorithm 5.

  t_{al} : Let M_p and M_q be the starting and ending configurations of ϱ^n. Let $r = M_p(leave) - M_q(leave)$. To construct g^n from g^{n-1}, tuples are removed such that agents

---

$$\varrho \; = \; M_{\mathcal{O}} \rightarrow M_{\mathcal{O}}^{11} \rightarrow M_{\mathcal{O}}^{12} \rightarrow \cdots \rightarrow M_{\mathcal{O}}^{1\mathrm{m}_1} \rightarrow \cdots \rightarrow M_{\mathcal{O}}^{i} \rightarrow \cdots \rightarrow M_{\mathcal{O}}^{j1} \rightarrow M_{\mathcal{O}}^{j2} \rightarrow \cdots \rightarrow M_{\mathcal{O}}^{j\mathrm{m}_j} \rightarrow \cdots \rightarrow M_{\mathcal{O}}^{k1} \rightarrow M_{\mathcal{O}}^{k2} \rightarrow \cdots \rightarrow M_{\mathcal{O}}^{k\mathrm{m}_k} \rightarrow \cdots$$

$$\varrho^0 \qquad \varrho^1 \qquad \varrho^i \qquad \varrho^j \qquad \varrho^k$$

$$\rho \; = \; g^0 \xrightarrow{\text{joining}} g^1 \rightarrow \cdots \xrightarrow{\text{joint action}} g^i \rightarrow \cdots \xrightarrow{\text{leaving}} g^j \rightarrow \cdots \xrightarrow{\text{joining}} g^k \rightarrow \cdots$$

Figure 2: Pictorial representation of construction of ρ from ϱ

Algorithm 5: g^n for Agent joining

1   $\mathbb{D} \leftarrow$ set of all identities present in g^{n-1}
2   $id \leftarrow \mathbb{N} \setminus \mathbb{D}$
3   **for** $j \leftarrow 1, \ldots r$ **do**
4      $id \leftarrow id + 1$
5   add $\langle \iota, id \rangle$ to g^{n-1}

---

in the 'leave' state with the smallest id are removed first. This process is detailed in Algorithm 6, which iterates over each agent type, finds the smallest id for agents that are leaving and removes these tuples from g^{n-1}.

Algorithm 6: g^n for Agent leaving

1   **for** $j \leftarrow 1, \ldots r$ **do**
2      $id \leftarrow$ smallest id among all the *leave* in g^{n-1}
3   remove $\langle leave, id \rangle$ from g^{n-1}

$\mathfrak{t}$: Let $M_{\mathcal{O}}^n$ be the configuration in ϱ^n. Our aim is to construct g^n from $M_{\mathcal{O}}^{n-1}$ and $M_{\mathcal{O}}^n$. We extract the information about joint action from $M_{\mathcal{O}}^n$ and $\mathfrak{t}$ and use it to compute the next global state g^n and the label of joint action (a sequence a over Act) for the current state g^{n-1} as described in Algorithm 7.

It is easy to see that ρ as defined above, is a run in $\mathcal{O}$. $\qquad \square$

Combining the above two claims, we can assert the following:

Lemma 1.

$$Runs(\mathcal{O}) \neq \emptyset \text{ iff } Runs(N_{\mathcal{O}}) \neq \emptyset$$

Theorem 1. *Safety checking in Regular OMAS is decidable.*

Proof: As a proof to the theorem, we reduce the problem of safety checking in Regular Systems to the Liveness ($L1 - live$) problem in Petri net.

A transition t in a Petri net (N, M_0) is said to be $L1 - live$ if t occurs at least once in some firing sequence in $\mathcal{L}(M_0)$.

Algorithm 7: g^n for joint action, given g^{n-1} and $\mathfrak{t}$

1   $n \leftarrow$ number of agents in g^{n-1}
2   Copy g^{n-1} to g^n
3   Initialize a with n number of τ actions
     // $a = [\tau, \tau, \cdots \tau]$
4   $t_e \leftarrow$ environment tuple from $\mathfrak{t}$
5   Remove t_e from $\mathfrak{t}$
6   Change the state $t_e[1]$ in g^n to $t_e[3]$
7   Add $t_e[2]$ to the $n+1^{th}$ position of a
8   **for** *each* $t \in \mathfrak{t}$ **do**
9   **for** $i \leftarrow 1, 2 \cdots n$ **do**
10   **if** $g^n.i == t[1]$ **then**
11          $g^n.i \leftarrow t[4]$
12          $a[i] \leftarrow t[2]$
13   break

14   **return** g^n, a

Safety checking problem in Regular OMAS can be defined as follows: Given a regular MAS $\mathcal{O}$, we first define the notion of a global state g being *safe*. Let $g = \langle l_1, l_2, \ldots l_e \rangle$ be a global state in $\mathcal{O}$, g is safe if $\forall l \in g \setminus l_e$, $V(l) = \{safe\}$. If there is a location $l \in g \setminus l_e$ such that $V(l) = \{\}$ then g is labeled *unsafe*. A regular MAS $\mathcal{O}$ with initial global state g_0 is said to be *safe* if every global state g reachable from g_0 is *safe*. Alternatively, no global global state labeled *unsafe* is reachable.

Now the reduction of safety checking to *liveness* is as follows: Recall that a transition $\mathfrak{t}$ in $N_{\mathcal{O}}$ is in the form $\langle (l_1, \mathfrak{a}_1, A_1, l'_1), (l_2, \mathfrak{a}_2, A_2, l'_2), \ldots, (l_e, b, l'_e) \rangle$. First we compute the set of unsafe transitions T_{unsafe} as given in the Algorithm 8. Now for every $\mathfrak{t} \in T_{unsafe}$, we check whether $\mathfrak{t}$ is $L1 - live$ in $(N_{\mathcal{O}}, M_{\mathcal{O}}^0)$ or not. If any of the $\mathfrak{t}$ is $L1 - live$, then we conclude that $\mathcal{O}$ is *unsafe*. Otherwise, if all the $\mathfrak{t}$ are not $L1 - live$ then $\mathcal{O}$ is *safe*.

$\qquad \square$

5. Related Work

In Kouvaros et al. [1] the undecidability of verification in open multi-agent systems is proved by encoding it into the problem of uniform termination in lossy counter systems [23]. Further, they have proposed two classes of OMAS,

---

Algorithm 8: T_{unsafe} from $T_{\mathcal{O}}$

1   $T_{unsafe} = \emptyset$
2   **for** *each* $\mathfrak{t} \in T_{\mathcal{O}}$ **do**
3   **for** *each* $tp \in \mathfrak{t}$ **do**
4   **if** $V(tp[3]) == \{\}$ **then**
5         $T_{unsafe} = T_{unsafe} \cup \mathfrak{t}$
6   Break

7   **return** T_{unsafe}

---

collective open interpreted systems (COIS) and interleaved open interpreted systems (IOIS), that admit verification procedures. It turns out that the first class, COIS, is particularly interesting in the context of our work on Rutwiya Systems. COIS are obtained from OMAS by introducing a constraint on the transition function of agent templates wherein an individual agent's local transitions depend on the state of the agent and the *collective action* performed by the system at a particular instance. They borrow techniques from [24] to implement a decision procedure for COIS verification. Even though verification of COIS is shown to be decidable only for swarms, the technique can be extended to MAS, as already claimed in [24]. It may be observed that the constraints introduced in our work, to obtain Rutwiya Systems, are orthogonal to those which are used to define COIS. In Rutwiya Systems we designate special leave states in each agent template and allow no arbitrary exits of agents. This makes sure that Rutwiya Systems become naturally encodable into multi-counter systems and, in turn, Petri nets. On the other hand, the constraints in COIS are defined on transitions and, therefore, Rutwiya Systems are not amenable to encoding into COIS.

The concept of utilising process counters was initially introduced by Lubachevsky [25] in 1984. Emerson and Trefler [20] proposed the concept of *generic representatives* to handle the complexity of symbolically represented systems with symmetry reduction.

Parameterized verification is the domain where the term *counter abstraction* is first used. Pnueli et al. [26] proposed a counter abstraction method, which involves abstracting a parameterized system with an unbounded size into a finite-state system. The application of counter abstraction methods for verifying robot swarms is detailed in [27].

The counter abstraction method is effectively utilized in other fields such as software model checking, as demonstrated by Basler et al. [28]. They illustrated its application to real-world concurrent programs, showcasing its ability to factor out redundancy arising from thread replication. Emerson and Namjoshi [29] introduced a fully automated method for the parameterized model checking problem in synchronous systems, demonstrating its decidability for properties specified in an indexed propositional temporal logic within the context of systems comprised of a control process and numerous homogeneous user processes. Emerson and Namjoshi [30] as well as German and Sistla [31]

have demonstrated the feasibility of automated solutions to the parameterized model checking problem in some special cases.

A similar use of counters can be found in the non-negotiating distributed computing model introduced by Delporte-Gallet et al. [32]. In their work, counters are employed in a shared memory model to track process progress, enabling solutions to complex distributed tasks such as multidimensional approximate agreement. They propose a very weak asynchronous crash failure model where each process writes its input to shared memory and then repeatedly writes a private counter to its register, reads the counters from other registers, and increments its counter until a stopping condition is met. While their focus is on concurrent programming, the concept of associating counters with process states resonates with our approach in multi-agent systems.

Additionally, Alcantara et al. [33] explore the coordination of autonomous robots under the Look-Compute-Move model in an asynchronous environment. Their work addresses the challenge of ensuring task completion without global synchronization, where each robot operates independently based on local observations.

6. Conclusion and Future Work

In this report, we defined a subclass of open MAS called Rutwiya Systems where the agents leave the system only from its leaving state. We then showed that reachability in Rutwiya Systems is decidable by encoding it into multi-counter systems.

A natural future work would be to implement a tool that uses the encoding method described above to convert Rutwiya Systems into multi-counter systems that can be further integrated with Petri net checking tools such as **KReach** [34] to check reachability in Rutwiya Systems. Furthermore, following the work of Basler et al. [28], we may explore the possibility of enhancing our automatic reachability checker by interleaving the translation process (to Petri net) and reachability checking process.

References

[1] P. Kouvaros, A. Lomuscio, E. Pirovano, H. Punchihewa, Formal verification of open multi-agent systems, in: E. Elkind, M. Veloso, N. Agmon, M. E. Taylor (Eds.), Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019, International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 179–187.

[2] A. Sales, P. Mira, A. M. Nascimento, A. Brandão, M. Saska, T. Nascimento, Heterogeneous multi-robot systems approach for warehouse inventory management, in: 2023 International Conference on Unmanned Aircraft Systems (ICUAS), IEEE, 2023, pp. 389–394.

[3] M. Dunbabin, L. Marques, Robots for environmental monitoring: Significant advancements and applications, IEEE Robotics Autom. Mag. 19 (2012) 24–39.

[4] M. Luckcuck, M. Farrell, L. A. Dennis, C. Dixon, M. Fisher, Formal specification and verification of autonomous robotic systems: A survey, ACM Comput. Surv. 52 (2019) 100:1–100:41.

---

[5] G. Beni, From swarm intelligence to swarm robotics, in: E. Sahin, W. M. Spears (Eds.), Swarm Robotics, SAB 2004 International Workshop, Santa Monica, CA, USA, July 17, 2004, Revised Selected Papers, volume 3342 of *Lecture Notes in Computer Science*, Springer, 2004, pp. 1–9.

[6] A. P. Engelbrecht, Fundamentals of Computational Swarm Intelligence, Wiley, 2005.

[7] M. Bourahla, M. Benmohamed, Formal specification and verification of multi-agent systems, in: R. J. G. B. de Queiroz, P. Cégielski (Eds.), Proceedings of the 11th Workshop on Logic, Language, Information and Computation, WoLLIC 2004, Fontainebleau, France, July 19-22, 2004, volume 123 of *Electronic Notes in Theoretical Computer Science*, Elsevier, 2004, pp. 5–17.

[8] Y. Rizk, M. Awad, E. W. Tunstel, Decision making in multiagent systems: A survey, IEEE Trans. Cogn. Dev. Syst. 10 (2018) 514–529.

[9] I. Czarnowski, P. Jedrzejowicz, Machine learning and multiagent systems as interrelated technologies, in: I. Czarnowski, P. Jedrzejowicz, J. Kacprzyk (Eds.), Agent-Based Optimization, volume 456 of *Studies in Computational Intelligence*, Springer, 2013, pp. 1–28.

[10] R. Siegfried, Modeling and Simulation of Complex Systems - A Framework for Efficient Agent-Based Modeling and Simulation, Springer, 2014.

[11] P. Gammie, R. van der Meyden, MCK: model checking the logic of knowledge, in: R. Alur, D. A. Peled (Eds.), Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004, Proceedings, volume 3114 of *Lecture Notes in Computer Science*, Springer, 2004, pp. 479–483.

[12] M. Kacprzak, W. Nabialek, A. Niewiadomski, W. Penczek, A. Pólrola, M. Szreter, B. Wozna, A. Zbrzezny, Verics 2007 - a model checker for knowledge and real-time, Fundam. Informaticae 85 (2008) 313–328.

[13] A. Lomuscio, H. Qu, F. Raimondi, MCMAS: an open-source model checker for the verification of multi-agent systems, Int. J. Softw. Tools Technol. Transf. 19 (2017) 9–30.

[14] N. Alechina, M. Dastani, F. Khan, B. Logan, J.-J. C. Meyer, Using theorem proving to verify properties of agent programs, Specification and verification of multi-agent systems (2010) 1–33.

[15] A. S. Rao, Agentspeak(l): BDI agents speak out in a logical computable language, in: W. V. de Velde, J. W. Perram (Eds.), Agents Breaking Away, 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, Eindhoven, The Netherlands, January 22-25, 1996, Proceedings, volume 1038 of *Lecture Notes in Computer Science*, Springer, 1996, pp. 42–55.

[16] R. H. Bordini, M. Fisher, W. Visser, M. J. Wooldridge, Verifying multi-agent programs by model checking, Auton. Agents Multi Agent Syst. 12 (2006) 239–256.

[17] M. Venkatraman, M. P. Singh, Verifying compliance with commitment protocols, Auton. Agents Multi Agent Syst. 2 (1999) 217–236.

[18] F. Belardinelli, D. Grossi, A. Lomuscio, Finite abstractions for the verification of epistemic properties in open multi-agent systems, in: Q. Yang, M. J. Wooldridge (Eds.), Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015, AAAI Press, 2015, pp. 854–860.

[19] P. Kouvaros, A. Lomuscio, Parameterised verification for multi-agent systems, Artif. Intell. 234 (2016) 152–189.

[20] E. A. Emerson, R. J. Trefler, From asymmetry to full symmetry: New techniques for symmetry reduction in model checking, in: L. Pierre, T. Kropf (Eds.), Correct Hardware Design and Verification Methods, 10th IFIP WG 10.5 Advanced Research Working Conference, CHARME '99, Bad Herrenalb, Germany, September 27-29, 1999, Proceedings, volume 1703 of *Lecture Notes in Computer Science*, Springer, 1999, pp. 142–156.

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

[22] M. Fisher, R. C. Cardoso, E. C. Collins, C. Dadswell, L. A. Dennis, C. Dixon, M. Farrell, A. Ferrando, X. Huang, M. Jump, G. Kourtis, A. Lisitsa, M. Luckcuck, S. Luo, V. Pagé, F. Papacchini, M. Webster, An overview of verification and validation challenges for inspection robots, Robotics 10 (2021) 67.

[23] P. Schnoebelen, Lossy counter machines decidability cheat sheet, in: A. Kucera, I. Potapov (Eds.), Reachability Problems, 4th International Workshop, RP 2010, Brno, Czech Republic, August 28-29, 2010. Proceedings, volume 6227 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 51–75.

[24] P. Kouvaros, A. Lomuscio, Verifying emergent properties of swarms, in: Q. Yang, M. J. Wooldridge (Eds.), Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015, AAAI Press, 2015, pp. 1083–1089.

[25] B. D. Lubachevsky, An approach to automating the verification of compact parallel coordination programs I, Acta Informatica 21 (1984) 125–169.

[26] A. Pnueli, J. Xu, L. D. Zuck, Liveness with (0, 1, infty)-counter abstraction, in: E. Brinksma, K. G. Larsen (Eds.), Computer Aided Verification, 14th International Conference, CAV 2002, Copenhagen, Denmark, July 27-31, 2002, Proceedings, volume 2404 of *Lecture Notes in Computer Science*, Springer, 2002, pp. 107–122.

[27] P. Kouvaros, A. Lomuscio, A counter abstraction technique for the verification of robot swarms, in: B. Bonet, S. Koenig (Eds.), Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA, AAAI Press, 2015, pp. 2081–2088.

[28] G. Basler, M. Mazzucchi, T. Wahl, D. Kroening, Symbolic counter abstraction for concurrent software, in: A. Bouajjani, O. Maler (Eds.), Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings, volume 5643 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 64–78.

[29] E. A. Emerson, K. S. Namjoshi, Automatic verification of parameterized synchronous systems (extended abstract), in: R. Alur, T. A. Henzinger (Eds.), Computer Aided Verification, 8th International Conference, CAV '96, New Brunswick, NJ, USA, July 31 - August 3, 1996, Proceedings, volume 1102 of *Lecture Notes in Computer Science*, Springer, 1996, pp. 87–98.

[30] E. A. Emerson, K. S. Namjoshi, Reasoning about rings, in: R. K. Cytron, P. Lee (Eds.), Conference Record of POPL'95: 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Francisco, California, USA, January 23-25, 1995, ACM Press, 1995, pp. 85–94.

[31] S. M. German, A. P. Sistla, Reasoning about systems with many processes, J. ACM 39 (1992) 675–735.

[32] C. Delporte-Gallet, H. Fauconnier, P. Fraigniaud, S. Rajsbaum, C. Travers, Non-negotiating distributed computing, in: Y. Emek (Ed.), Structural Information and Communication Complexity - 31st International Colloquium, SIROCCO 2024, Vietri sul Mare, Italy, May 27-29, 2024, Proceedings, volume 14662 of *Lecture Notes in Computer Science*, Springer, 2024, pp. 208–225.

[33] M. Alcantara, A. Castañeda, D. Flores-Peñaloza, S. Rajsbaum, The topology of look-compute-move robot wait-free algorithms with hard termination, Distributed Comput. 32 (2019) 235–255.

[34] A. Dixon, R. Lazic, Kreach: A tool for reachability in petri nets, in: A. Biere, D. Parker (Eds.), Tools and Algorithms for the Construction and Analysis of Systems - 26th International Conference, TACAS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings, Part I, volume 12078 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 405–412.