Reconfigurable Interaction for MAS Modelling*

Yehia Abd Alrahman
University of Gothenburg
Gothenburg, Sweden

Giuseppe Perelli
Sapienza University of Rome
Rome, Italy

Nir Piterman
University of Gothenburg
University of Leicester
Gothenburg, Sweden

ABSTRACT

We propose a formalism to model and reason about multi-agent systems. We allow agents to interact and communicate in different modes so that they can pursue joint tasks; agents may dynamically synchronize, exchange data, adapt their behaviour, and reconfigure their communication interfaces. The formalism defines a local behaviour based on shared variables and a global one based on message passing. We extend LTL to be able to reason explicitly about the intentions of the different agents and their interaction protocols. We also study the complexity of satisfiability and model-checking of this extension.

KEYWORDS

Agent Theories and Models, Logics for Agent Reasoning, Verification of Multi-Agent Systems

1 INTRODUCTION

In recent years formal modelling of multi-agent systems (MAS) and their analysis through model checking has received much attention [30, 41]. Several mathematical formalisms have been suggested to represent the behaviours of such systems and to reason about the strategies that agents exhibit [6, 30]. For instance, modelling languages, such as RM [5, 22] and ISPL [30], are used to enable efficient analysis by representing these systems through the usage of BDDs. Temporal logics have been also extended and adapted (e.g., with Knowledge support [17] and epistemic operators [20]) specifically to support multi-agent modelling [21]. Similarly, logics that support reasoning about the intentions and strategic abilities of such agents have been used and extended [14, 37].

These works are heavily influenced by the formalisms used for verification (e.g., Reactive Modules [4, 5], concurrent game structures [6], and interpreted systems [30]). They rely on shared memory to implicitly model interactions. It is generally agreed that explicit message passing is more appropriate to model interactions among distributed agents because of its scalability [9, 26]. However, the mentioned formalisms trade the advantages of message passing for abstraction, and abstract message exchange by controlling the visibility of state variables of the different agents. Based on an early result, where a compilation from shared memory to message passing was provided [8], it was believed that a shared memory model is a higher level abstraction of distributed systems. However, this result holds only in specific cases and under assumptions that practically proved to be unrealistic, see [2]. Furthermore, the compositionality of shared memory approaches is limited and the supported interaction interfaces are in general not very flexible [10]. Alternatively, message passing formalisms [35] are very compositional and support flexible interaction interfaces. However, unlike shared memory formalisms, they do not accurately support awareness capabilities, where an agent may instantaneously inspect its local state and adapt its behaviour while interacting[1].

To combine the benefits of both approaches recent developments [2, 39] suggest adopting hybrids, to accurately represent actual distributed systems, e.g., [3, 33]. We propose a hybrid formalism to model and reason about distributed multi-agent systems. A system is represented as a set of agents (each with local state), executing concurrently and only interacting by message exchange. Inspired by multi-robot systems, e.g., Kilobot [38] and Swarmanoid [15], agents are additionally able to sense their local states and partially their surroundings. Interaction is driven by message passing following the interleaving semantics of [35]; in that only one agent may send a message at a time while other agents may react to it. To support meaningful interaction among agents [40], messages are not mere synchronisations, but carry data that might be used to influence the behaviour of receivers.

Our message exchange is adaptable and reconfigurable. Thus, agents determine how to communicate and with whom. Agents interact on links that change their utility based on the needs of interaction at a given stage. Unlike existing message-passing mechanisms, which use static notions of network connectivity to establish interactions, our mechanisms allow agents to specify receivers using logical formulas. These formulas are interpreted over the evolving local states of the different agents and thus provide a natural way to establish reconfigurable interaction interfaces (for example, limited range communication [33], messages destined for particular agents [1], etc.).

The advantages of our formalism are threefold. We provide more realistic models that are close to their distributed implementations, and how actual distributed MAS are developed, e.g., [25]. We provide a modelling convenience for high level interaction features of MAS (e.g., coalition formation, collaboration, self-organisation, etc), that otherwise have to be hard-coded tediously in existing formalisms. Furthermore, we decouple the individual behaviour of agents from their interaction protocols to facilitate reasoning about either one separately.

---

[1]They model agents as mathematical expressions over interaction operators. Thus the state of an agent is implicit in the structure of the expression.

In addition, we extend LTL to characterise messages and their targets. This way we allow reasoning about the intentions of agents in communication. Our logic can refer directly to the interaction protocols. Thus the interpretation of a formula incorporates information about the causes of assignments to variables and the flow of the interaction protocol. We also study the complexity of satisfiability and Model-checking for our logic.

The paper's structure is as follows: In Sect. 2, we informally present our model and motivate our design choices. In Sect. 3, we give the background. In Sect. 4 and Sect. 5 we introduce the formalism and a case study. In Sect. 6 we discuss our extension to LTL and in Sect. 7 we conclude.

2 AN INFORMAL OVERVIEW

We use a collaborative-robot scenario to informally illustrate the distinctive features of our formalism and we later formalise it in Section 5. The scenario is based on Reconfigurable Manufacturing Systems (RMS) [27, 31], where assembly product lines coordinate autonomously with different types of robots to produce products.

In our formalism, each agent has a local state consisting of a set of variables whose values may change due to either contextual conditions or side-effects of interaction. The external behaviour of an agent is only represented by the messages it exposes to other agents while the local one is represented by changes to its state variables. These variables are initialised by initial conditions and updated by send- and receive- transition relations. In our example, a product-line agent initiates different production procedures based on the assignment to its product variable "prd", which is set by the operator, while it controls the progress of its status variable "st" based on interactions with other robots. Furthermore, a product-line agent is characterised: (1) externally only by the recruitment and assembly messages it sends to other robots and (2) internally by a sequence of assignments to its local variables.

Before we explain the send- and receive- transition relations and show the dynamic reconfiguration of communication interfaces we need to introduce a few additional features. We assume that there is an agreed set of *channels/links* CH that includes a unique broadcast channel ★. Broadcasts have non-blocking send and blocking receive while multicasts have blocking send and receive. In a broadcast, receivers (if exist) may anonymously receive a message when they are interested in its values and when they satisfy the send guard. Otherwise, the agent does not participate in the interaction either because they cannot (do not satisfy the guard) or because they are not interested (make an idle transition). In multicast, all agents connected to the multicast channel must participate to enable the interaction. For instance, recruitment messages are broadcast because a line agent assumes that there exist enough robots to join the team while assembly messages are multicast because they require that the whole connected team is ready to assemble the product.

Agents dynamically decide (based on local state) whether they can use/connect-to multicast channels while the broadcast channel is always available. Thus, initially, agents may not be connected to any channel, except for the broadcast one ★. These channels may be learned using broadcast messages and thus a structured communication interface can be built at run-time, starting from a (possibly) flat one.

Agents use messages to send selected data and specify how and to whom. Namely, the values in a message specify what is exposed to the others; the channel specifies how to coordinate with others; and a send guard specifies the target. Accordingly, each message carries an assignment to a set of agreed *data variables* D, i.e., the exposed data; a channel ch; and a send guard g^s. In order to write meaningful send guards, we assume a set of *common variables* CV that each agent stores locally and assigns its own information (e.g., the type of agent, its location, its readiness, etc.). Send guards are expressed in terms of conditions on these variables and are evaluated per agent based on their assigned local values. Send guards are parametric to the local state of the sender and specify what assignments to common variables a potential receiver must have. For example, an agent may send a dedicated link name to a selected set of agents by assigning a data variable in the communicated message and this way a coalition can be built incrementally at run-time. In our RMS, the send guard of the recruitment message specifies the types of the targeted robots while the data values expose the number of required robots per type and a dedicated multicast link to be used to coordinate the production.

Targeted agents may use incoming messages to update their states, reconfigure their interfaces, and/or adapt their behaviour. In order to do so, however, agents are equipped with receive guards g^r; that might be parametrised to local variables and channels, and thus dynamically determine if an agent is connected to a given channel. The interaction among different agents is then derived based on send- and receive- transition relations. These relations are used to decide when to send/receive a message and what are the side-effects of interaction. Technically, every agent has a send and a receive transition relation. Both relations are parameterised by the state variables of the agent, the data variables transmitted on the message, and by the channel name. A sent message is interpreted as a joint transition between the send transition relation of the sender and the receive transition relations of all the receivers. For instance, a robot's receive guard specifies that other than the broadcast link it is also connected to a multicast link that matches the current value of its local variable "lnk". The robot then uses its receive transition relation to react to a recruitment message, for instance, by assigning to its "lnk" the link's data value from the message.

Furthermore, in order to send a message the following has to happen. The send transition relation of the sender must hold on: a given state of the sender, a channel name, and an assignment to data variables. If the message is broadcast, all agents whose assignments to common variables satisfy the send guard jointly receive the message, the others discard it. If the message is multicast, all connected agents must satisfy the send guard to enable the transmission (as otherwise they block the message). In both cases, sender and receivers execute their send- and receive-transition relations jointly. The local side-effect of the message takes into account the origin local state, the channel, and the data. In our example, a (broadcast) recruitment message is received by all robots that are not assigned to other teams (assigned ones discard it) and as a side effect they connect to a multicast channel that is specified in the message. A (multicast) assembly message can only be sent when the whole recruited team is ready to receive (otherwise the message is blocked) and as a side effect the team proceeds to the next production stage.

Clearly, the dynamicity of our formalism stems from the fact that we base interactions directly over the evolving states of the different agents rather than over static notions of network connectivity as of existing approaches.

3 TRANSITION SYSTEMS

A *Doubly-Labeled Transition System* (DLTS) is $\mathscr{T} = \langle \Sigma, \Upsilon, S, S_0, R, L \rangle$, where Σ is a *state alphabet*, Υ is a *transition alphabet*, S is a set of states, $S_0 \subseteq S$ is a set of initial states, $R \subseteq S \times \Upsilon \times S$ is a transition relation, and $L : S \to \Sigma$ is a labelling function.

A *path* of a transition system $\mathscr{T}$ is a maximal sequence of states and transition labels $\sigma = s_0, a_0, s_1, a_1, \ldots$ such that $s_0 \in S_0$ and for every $i \geq 0$ we have $(s_i, a_i, s_{i+1}) \in R$. We assume that for every state $s \in S$ there are $a \in \Upsilon$ and $s' \in S$ such that $(s, a, s') \in R$. Thus, a sequence σ is maximal if it is infinite. If $|\Upsilon| = 1$ then $\mathscr{T}$ is a *state-labeled system* and if $|\Sigma| = 1$ then $\mathscr{T}$ is a *transition-labeled system*.

We introduce *Discrete Systems* (DS) that represent state-labeled systems symbolically. A DS is $\mathscr{D} = \langle \mathscr{V}, \theta, \rho \rangle$, where the components of $\mathscr{D}$ are as follows:

- $\mathscr{V} = \{v_1, ..., v_n\}$: A finite set of typed variables. Variables range over discrete domains, e.g., Boolean or Integer. A *state* s is an interpretation of $\mathscr{V}$, i.e., if D_v is the domain of v, then s is in $\prod_{v_i \in \mathscr{V}} D_{v_i}$.
- θ : This is an assertion over $\mathscr{V}$ characterising all the initial states of the DS. A state is called *initial* if it satisfies θ.
- ρ : A *transition relation*. This is an assertion $\rho(\mathscr{V} \cup \mathscr{V}')$, where $\mathscr{V}'$ is a primed copy of variables in $\mathscr{V}$. The transition relation ρ relates a state $s \in \Sigma$ to its $\mathscr{D}$-*successors* $s' \in \Sigma$, i.e., $(s, s') \vDash \rho$, where s is an interpretation to variables in $\mathscr{V}$ and s' is for variables in $\mathscr{V}'$.

The DS $\mathscr{D}$ gives rise to a state-labeled transition system $\mathscr{T}_{\mathscr{D}} = \langle \Sigma, \{1\}, T, T_0, R \rangle$, where Σ and T are the set of states of $\mathscr{T}_{\mathscr{D}}$, T_0 is the set of initial states, and R is the set of triplets $(s, 1, s')$ such that $(s, s') \vDash \rho$. Clearly, the paths of $\mathscr{T}_{\mathscr{D}}$ are exactly the paths of $\mathscr{D}$, but the size of $\mathscr{T}_{\mathscr{D}}$ is exponentially larger than the description of $\mathscr{D}$.

A common way to translate a DLTS into a DS, which we adapt and extend below, would be to include additional variables that encode the transition alphabet. Given such a set of variables $\mathscr{V}_{\Upsilon}$, an assertion $\rho(\mathscr{V} \cup \mathscr{V}_{\Upsilon} \cup \mathscr{V}')$ characterises the triplets (s, v, s') such that $(s, v, s') \vDash \rho$, where s supplies the interpretation to $\mathscr{V}$, v to $\mathscr{V}_{\Upsilon}$ and s' to $\mathscr{V}'$.

4 RECIPE: RECONFIGURABLE COMMUNICATING PROGRAMS

We formally present the RECIPE communication formalism and its main ingredients. We start by specifying agents (or programs) and their local behaviours and we show how to compose these local behaviours to generate a global (or a system) one. We assume that agents rely on a set of common variables CV, a set of data variables D, and a set of channels CH containing the broadcast one $\star$.

Definition 4.1 (Agent). An agent is $A_i = \langle V_i, f_i, g_i^s, g_i^r, \mathcal{T}_i^s, \mathcal{T}_i^r, \theta_i \rangle$, where:

- V_i is a finite set of typed local variables, each ranging over a finite domain. A state s^i is an interpretation of V_i, i.e., if Dom(v) is the domain of v, then s^i is an element in $\prod_{v \in V_i}$ Dom(v). We

use V' to denote the primed copy of V and Id_i to denote the assertion $\bigwedge_{v \in V_i} v = v'$.

- $f_i : \text{CV} \to V_i$ is a function, associating common variables to local variables. We freely use the notation f_i for the assertion $\bigwedge_{cv \in \text{CV}} cv = f_i(cv)$.
- $g_i^s(V_i, \text{CH}, \text{D}, \text{CV})$ is a send guard specifying a condition on receivers. That is, the predicate, obtained from g_i^s after assigning s^i, ch, and $\mathbf{d}$ (an assignment to D) , which is checked against every receiver j after applying f_j.
- $g_i^r(V_i, \text{CH})$ is a receive guard describing the connection of an agent to channel ch. We let $g_i^r(V_i, \star) = \text{true}$, i.e., every agent is always connected to the broadcast channel. We note, however, that receiving a broadcast message could have no effect on an agent.
- $\mathcal{T}_i^s(V_i, V_i', \text{D}, \text{CH})$ is an assertion describing the send transition relation while $\mathcal{T}_i^r(V_i, V_i', \text{D}, \text{CH})$ is an assertion describing the receive transition relation. We assume that an agent is broadcast input-enabled, i.e., $\forall v, \mathbf{d} \; \exists v'$ s.t. $\mathcal{T}_i^r(v, v', \mathbf{d}, \star)$.
- θ_i is an assertion on V_i describing the initial states, i.e., a state is initial if it satisfies θ_i.

Agents exchange messages. A message is defined by the channel it is sent on, the data it carries, the sender identity, and the assertion describing the possible local assignments to common variables of receivers. Formally:

Definition 4.2 (Observation). An observation is a tuple $m = (ch, \mathbf{d}, i, \pi)$, where ch is a channel, $\mathbf{d}$ is an assignment to D, i is an identity, and π is a predicate over CV.

In Definition 4.2 we interpret π as a set of possible assignments to common variables CV. In practice, π is obtained from $g_i^s(s^i, ch, \mathbf{d}, \text{CV})$ for an agent i, where $s^i \in \prod_{v \in V_i}$ Dom(v) and ch and $\mathbf{d}$ are the channel and assignment in the observation. We freely use π to denote either a predicate over CV or its interpretation, i.e., the set of variable assignments c such that $c \vDash \pi$.

A set of agents agreeing on the common variables CV, data variables D, and channels CH define a *system*. We define a DLTS capturing the interaction and then give a DS-like symbolic representation of the same system.

Let Υ be the set of possible observations. That is, let CH be the set of channels, $\mathscr{D}$ the product of the domains of variables in D, $\mathscr{A}$ the set of agent identities, and $\Pi(\text{CV})$ the set of predicates over CV then $\Upsilon \subseteq \text{CH} \times \mathscr{D} \times \mathscr{A} \times \Pi(\text{CV})$. In practice, we restrict attention to predicates in $\Pi(\text{CV})$ that are obtained from $g_i^s(V_i, \text{CH}, \text{D}, \text{CV})$ by assigning to i and V_i the identity and a state of some agent.

Let $S_i = \prod_{v \in V_i}$ Dom(v) and $S = \prod_i S_i$. Given an assignment $s \in S$ we denote by s_i the projection of s on S_i.

Definition 4.3 (Transition System). Given a set $\{A_i\}_i$ of agents, we define a doubly-labeled transition system $\mathscr{T} = \langle \Sigma, \Upsilon, S, S_0, R, L \rangle$, where Υ and S are as defined above, $\Sigma = S$, S_0 are the states that satisfy $\bigwedge_i \theta_i$, $L : S \to \Sigma$ is the identity function, and R is as follows.

A triplet $(s, v, s') \in R$, where $v = (ch, \mathbf{d}, i, \pi)$, if the following conditions hold:

- For the sender i we have that $\pi = g_i^s(s_i, ch, \mathbf{d})$, i.e., π is obtained from g_i^s by assigning the state of i, the data variables assignment $\mathbf{d}$ and the channel ch, and $\mathcal{T}_i^s(s_i, s_i', \mathbf{d}, ch)$ evaluates to true.

- For every other agent i' we have that either (a) $\pi(f_{i'}^{-1}(s_{i'}))$, $\mathcal{T}_{i'}^{r}(s_{i'}, s_{i'}', \mathbf{d}, ch)$, and $g_{i'}^{r}(s_{i'}', ch)$ all evaluate to true, (b) $g_{i'}^{r}(s_{i'}, ch)$ evaluates to false and $s_{i'} = s_{i'}'$, or (c) $ch = \star$, $\pi(f_{i'}^{-1}(s_{i'}))$ evaluates to false and $s_{i'} = s_{i'}'$. By $\pi(f_{i'}^{-1}(s_i))$ we denote the assignment of $v \in \text{cv}$ by $s_i(f_{i'}(v))$ in π.

An observation $(ch, \mathbf{d}, i, \pi)$ labels a transition from s to s' if the sender i determines the predicate (by assigning s_i, $\mathbf{d}$, and ch in g_i^s) and the send transition of i is satisfied by assigning s_i, s_i' and $\mathbf{d}$ to it, i.e., the sender changes the state from s_i to s_i' and sets the data variables in the observation to $\mathbf{d}$. All the other agents either (a) satisfy this condition on receivers (when translated to their local copies of the common variables), are connected to ch (according to $g_{i'}^r$), and perform a valid transition when reading the data sent in $\mathbf{d}$, (b) are not connected to ch (according to $g_{i'}^r$) and all their variables do not change, or (c) the channel is a broadcast channel, the agent does not satisfy the condition on receivers, and all their variables do not change.

We now define a symbolic version of the same transition system. To do that we have to extend the format of the allowed transitions from assertions over an extended set of variables to assertions that allow quantification.

Definition 4.4 (Discrete System). Given a set $\{A_i\}_i$ of agents, a system is defined as follows: $S = \langle \mathscr{V}, \rho, \theta \rangle$, where $\mathscr{V} = \biguplus_i V_i$ and $\theta = \bigwedge_i \theta_i$ and a state of the system is in $\prod_i \prod_{v \in V_i} \text{Dom}(v)$. The transition relation of the system is characterised as follows:

$$\rho : \exists ch \, \exists \text{D} \bigvee_k \mathcal{T}_k^s(V_k, V_k', \text{D}, ch) \wedge \bigwedge_{j \neq k} \exists \text{cv}. f_j \wedge$$
$$\begin{pmatrix} g_j^r(V_j, ch) \wedge \mathcal{T}_j^r(V_j, V_j', \text{D}, ch) \wedge g_k^s(V_k, ch, \text{D}, \text{cv}) \\ \vee \qquad\qquad\qquad \neg g_j^r(V_j, ch) \wedge \text{Id}_j \\ \vee \qquad ch = \star \wedge \neg g_k^s(V_k, ch, \text{D}, \text{cv}) \wedge \text{Id}_j \end{pmatrix}$$

The transition relation ρ relates a system state s to its successors s' given an observation $m = (ch, \mathbf{d}, k, \pi)$. Namely, there exists an agent k that sends a message with data $\mathbf{d}$ (an assignment to D) with assertion π (an assignment to g_k^s) on channel ch and all other agents are either (a) connected, satisfy the send predicate, and participate in the interaction, (b) not connected and idle, or (c) do not satisfy the send predicate of a broadcast and idle. That is, the agents satisfying π (translated to their local state by the conjunct $\exists \text{cv}. f_j$) and connected to channel ch (i.e., $g_j^r(s^j, ch)$) get the message and perform a receive transition. As a result of interaction, the state variables of the sender and these receivers might be updated. The agents that are *not connected* to the channel (i.e., $\neg g_j^r(s^j, ch)$) do not participate in the interaction and stay still. In case of broadcast, namely when sending on $\star$, agents are always connected and the set of receivers not satisfying π (translated again as above) stay still. Thus, a blocking multicast arises when a sender is blocked until all *connected* agents satisfy $\pi \wedge f_j$. The relation ensures that, when sending on a channel that is different from the broadcast channel $\star$, the set of receivers is the full set of *connected* agents. On the broadcast channel agents who do not satisfy the send predicate do not block the sender.

The translation above to a transition system leads to a natural definition of a trace, where the information about channels, data,

senders, and predicates is lost. We extend this definition to include this information as follows:

Definition 4.5 (System trace). A system trace is an infinite sequence $s_0 \xrightarrow{m_0} s_1 \xrightarrow{m_1} \ldots$ of system states and observations such that $\forall t \geq 0$: $m_t = (ch_t, \mathbf{d}_t, k, \pi_t)$, $\pi_t = g_k^s(s_t^k, \mathbf{d}_t, ch_t)$, and:

$$(s_t, \; s_{t+1}) \vDash \mathcal{T}_k^s(s_t^k, s_{t+1}^k, \mathbf{d}_t, ch_t) \wedge \bigwedge_{j \neq k} \exists \text{cv}. f_j \wedge$$
$$\begin{pmatrix} g_j^r(s_t^j, ch_t) \wedge \mathcal{T}_j^r(s_t^j, s_{t+1}^j, \mathbf{d}_t, ch_t) \wedge \pi_t \\ \vee \qquad\qquad \neg g_j^r(s_t^j, ch_t) \wedge s_t^j = s_{t+1}^j \\ \vee \qquad ch_t = \star \wedge \neg \pi_t \wedge s_t^j = s_{t+1}^j \end{pmatrix}$$

That is, we use the information in the observation to localize the sender k and to specify the channel, data values, and the send predicate.

The following lemma relates the traces arising from Definition 4.5 to that of Definition 4.3.

LEMMA 4.6. *The traces of a system composed of a set of agents $\{A_i\}_i$ are the paths of the induced DLTS.*

5 COLLABORATIVE ROBOTS

We complete the details of the RMS example informally described in Section 2. Many aspects of the example are kept very simple on purpose due to lack of space.

The system, in our scenario, consists of an assembly product line agent (line) and several types of task-driven robots. We only give a program for type-1 (t1) because type-2 (t2) and type-3 (t3) are similar. A product line is responsible for assembling the main parts and delivering the final product. Different types of robots are responsible for sub-tasks, e.g., retrieving and/or assembling individual parts. The product line is generic and can be used to produce different products and thus it has to determine the set of resources, to recruit a team of robots, to split tasks and to coordinate the final stage of production.

Every agent has copies of the common variables: @type indicating its type (line, t1, t2, t3), @asgn indicating whether a robot is assigned, and @rdy indicating what stage of production the robot is in. The set of channels includes the broadcast channel $\star$ and multicast channels $\{A, \ldots\}$. For simplicity, we only use the multicast channel A and fix it to the line agent. The set of data variables is $\{\text{MSG}, \text{NO}, \text{LNK}\}$ indicating the type of the message, a number (of robots per type), and a name of a channel. When a data variable is not important for some message it is omitted from the description. We also use the notation $\text{KEEP}(v)$ to denote that a variable is not changed by a transition.

In addition to copies of common variables (e.g., $f_l(\text{@type}) = \text{ltype}$), the line agent has the following state variables: st is a state ranging over $\{\text{pnd}, \text{strt}\}$ (pending and start), lnk is the link of the product line, prd is the id of the active product, and stage is used to follow stages of the production. The initial condition θ_l of a line agent is:

$$\theta_l : \text{st} = \text{pnd} \wedge \text{stage} = 0 \wedge \text{lnk} = A \wedge (\text{prd} = 1 \vee \text{prd} = 2)$$

Thus, the line agent starts with a task of assembling one of two products and uses channel A. If there are multiple lines, then each is initialised with a different channel.

The send guard of Line is of the form:

$$g_l^s : ch= \star \land \neg @\text{asgn} \land (\text{prd}=1 \rightarrow (@\text{type}=\text{t1} \lor @\text{type}=\text{t2})) \land$$
$$(\text{prd}=2 \rightarrow (@\text{type}=\text{t1} \lor @\text{type}=\text{t3})) \lor ch=\text{lnk} \land @\text{rdy} = \text{stage}$$

Namely, broadcasts are sent to robots whose @asgn is false (i.e., free to join a team). If the identity of the product to be assembled is 1, then the required agents are t1 and t2 and if the identity of the product is 2, then the required agents are t1 and t3. Messages on channel A (the value of lnk) are sent to connected agents when they reach a matching stage of production, i.e., @rdy = stage. The receive guard of Line is $ch = \star$, i.e., it is only connected to channel $\star$.

The send transition relation of Line is of the form:

$$\mathcal{T}_l^s : \quad \text{KEEP}(\text{lnk}, \text{prd}, \text{ltype}, \text{lasgn}, \text{lrdy}) \land$$
$$\begin{pmatrix} & \text{st} = \text{pnd} \land \mathbf{d}(\text{MSG} \mapsto \text{team}; \text{NO} \mapsto 2; \text{LNK} \mapsto \text{lnk}) \\ & \qquad \land \text{stage}' = 1 \land \text{st}' = \text{strt} \land ch = \star \\ \lor & \text{st} = \text{strt} \land \mathbf{d}(\text{MSG} \mapsto \text{asmbl}) \land \text{stage} = 1 \land \\ & \qquad \land \text{st}' = \text{strt} \land \text{stage}' = 2 \land ch = \text{lnk} \\ \lor & \text{st} = \text{strt} \land \mathbf{d}(\text{MSG} \mapsto \text{asmbl}) \land \text{st}' = \text{pnd} \\ & \qquad \land \text{stage} = 2 \land \text{stage}' = 0 \land ch = \text{lnk} \end{pmatrix}$$

Line starts in the pending state (see θ_l). It broadcasts a request ($\mathbf{d}(\text{MSG} \mapsto \text{team})$) for two robots ($\mathbf{d}(\text{NO} \mapsto 2)$) per required type asking them to join the team on the multicast channel stored in its lnk variable ($\mathbf{d}(\text{LNK} \mapsto \text{lnk})$). According to the send guard, if the identity of the product to assemble is 1 (prd = 1) the broadcast goes to type 1 and type 2 robots and if the identity is 2 then it goes to type 1 and type 3 robots. Thanks to channel mobility (i.e., $\mathbf{d}(\text{LNK}) = \text{lnk}$) a team on a dedicated link can be formed incrementally at run-time. In the start state, Line attempts an ASSEMBLE (blocking) multicast on A. The multicast can be sent only when the entire team completed the work on the production stage (when their common variable @rdy agrees with stage). One multicast increases the value of stage and keeps Line in the start state. A second multicast finalises the production and Line becomes free again.

We set $\mathcal{T}_l^r$: KEEP(all) as Line's recieve transition relation. That is, Line is not influenced by incoming messages.

We now specify the behaviour of t1-robots and show how an autonomous and incremental one-by-one team formation is done anonymously at run-time. In addition to copies of common variables a t1-robot has the following variables: st ranges over {pnd, strt, end}, step is used to control the progress of individual behaviour, no (resp. lnk) is a placeholder to a number (resp. link) learned at run-time, and f_b relabels common variables as follows: $f_b(@\text{type}) = \text{btype}$, $f_b(@\text{asgn}) = \text{basgn}$ and $f_b(@\text{rdy}) = \text{brdy}$.

Initially, a robot is available for recruitment:
$$\theta_b : (\text{st} = \text{pnd}) \land (\text{btype} = \text{t1}) \land \neg\text{basgn} \land (\text{lnk} = \bot) \land$$
$$(\text{step} = \text{brdy} = \text{no} = 0)$$

The send guard specifies that a robot only broadcasts to unassigned robots of the same type, namely:
$$g_b^s : (ch = \star) \land (@\text{type} = \text{btype}) \land \neg@\text{asgn}.$$

The receive guard specifies that a t1-robot is connected either to a broadcast $\star$ or to a channel matching the value of its link variable:
$$g_b^r : ch = \star \lor ch = \text{lnk}.$$

The send $\mathcal{T}_b^s$ and receive $\mathcal{T}_b^r$ transition relations are:

$$\mathcal{T}_b^s : \quad \text{KEEP}(\text{lnk}, \text{btype}) \land$$
$$\begin{pmatrix} & \text{st} = \text{strt} \land \mathbf{d}(\text{MSG} \mapsto \text{form}; \text{LNK} \mapsto \text{lnk}; \text{NO} \mapsto \text{no} - 1) \\ & \qquad \land (\text{no} \geq 1) \land \text{step} = 0 \land \text{step}' = 1 \land \text{st}' = \text{end} \\ & \qquad \qquad \land \text{basgn}' \land (\text{no}' = 0) \land ch = \star \\ \lor & \text{st} = \text{end} \land \text{step} = 1 \land \text{step}' = 2 \land \dots \\ \vdots & \quad [\text{INDIVIDUAL BEHAVIOR}] \\ \lor & \text{st} = \text{end} \land \dots \land \text{step}' = n \land \text{brdy}' = 1 \end{pmatrix}$$

$$\mathcal{T}_b^r : \quad \text{KEEP}(\text{btype}) \land$$
$$\begin{pmatrix} & \text{st} = \text{pnd} \land \mathbf{d}(\text{MSG} \mapsto \text{team}) \land \text{st}' = \text{strt} \\ & \qquad \land \text{lnk}' = \mathbf{d}(\text{LNK}) \land \text{no}' = \mathbf{d}(\text{NO}) \land ch = \star \\ \lor & \text{st} = \text{st}' = \text{strt} \land \mathbf{d}(\text{MSG} \mapsto \text{form}) \land \mathbf{d}(\text{NO}) > 0 \land ch = \star \\ & \qquad \land \neg\text{basgn} \land \text{lnk}' = \mathbf{d}(\text{LNK}) \land \text{no}' = \mathbf{d}(\text{NO}) \\ \lor & \text{st} = \text{strt} \land \mathbf{d}(\text{MSG} \mapsto \text{form}; \text{NO} \mapsto 0) \land ch = \star \\ & \qquad \land \neg\text{basgn} \land \text{st}' = \text{pnd} \land \text{lnk}' = \bot \land \text{no}' = 0 \\ \lor & \text{st} = \text{end} \land \mathbf{d}(\text{MSG} \mapsto \text{asmbl}) \land \text{brdy} = 1 \land ch = \text{lnk} \\ & \qquad \land \text{basgn}' \land \text{step} = n \land \text{st}' = \text{end} \land \text{brdy}' = 2 \land \text{step}' = 0 \\ \lor & \text{st} = \text{end} \land \mathbf{d}(\text{MSG} \mapsto \text{asmbl}) \land \text{brdy} = 2 \land ch = \text{lnk} \\ & \qquad \land \text{st}' = \text{pnd} \land \text{brdy}' = 0 \land \text{lnk}' = \bot \land \neg\text{basgn}' \end{pmatrix}$$

The team formation starts when unassigned robots are in pending states (pnd) as specified by the initial condition θ_b. From this state they may only receive a team message from a line agent (by the 1st disjunct of $\mathcal{T}_b^r$). The message contains the number of required robots $\mathbf{d}(\text{NO})$ and a team link $\mathbf{d}(\text{LNK})$. The robots copy these values to their local variables (i.e., $\text{lnk}' = \mathbf{d}(\text{LNK})$ etc.) and move to the start state ($\text{st}' = \text{strt}$) where they either join the team (by $\mathcal{T}_b^s$) or step back (by $\mathcal{T}_b^r$) as follows:

- By the 1st disjunct of $\mathcal{T}_b^s$ a robot joins the team by *broadcasting* a form message to t1-robots forwarding the number of still required robots ($\mathbf{d}(\text{NO}) = (\text{no} - 1)$) and the team link ($\mathbf{d}(\text{LNK}) = \text{lnk}$). This message is sent only if no ≥ 1, i.e, at least one robot is needed. The robot then moves to state (end) to start its mission.
- By the 2nd disjunct of $\mathcal{T}_b^r$ a robot *receives* a form message from a robot, updating the number of still required robots (i.e., if $\mathbf{d}(\text{NO}) > 0$), and remains in the start state.
- By the 3rd disjunct of $\mathcal{T}_b^r$ a robot *receives* a form message from a robot, informing that no more robots are needed, i.e., $\mathbf{d}(\text{NO}) = 0$. The robot then moves to the pending state and disconnects from the team link, i.e., $\text{lnk}' = \bot$. Thus it may not block interaction on the team link.

The last disjuncts of $\mathcal{T}_b^s$ specify that a team robot (i.e., with step = 1) starts its mission independently until it finishes ($\text{step}' = n \land \text{brdy}' = 1$) . When all team robots finish they enable an asmbl message on A, to start the next stage of production as specified by the 4th disjunct of $\mathcal{T}_b^r$. When the robots are at the final stage (i.e., brdy' = 2) they enable another asmbl message to finalise the product and subsequently they reset to their initial conditions.

6 LTOL: AN EXTENSION OF LTL

We introduce LTOL, an extension of LTL with the ability to refer and therefore reason about agents interactions. We replace the next operator of LTL with the observation descriptors: *possible* $\langle O \rangle$ and *necessary* $[O]$, to refer to messages and the intended set of receivers.

The syntax of formulas φ and *observation descriptors* O is as follows:

$$\varphi ::= v \mid \neg v \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \, \mathcal{U} \, \varphi \mid \varphi \, \mathcal{R} \, \varphi \mid \langle O \rangle \varphi \mid [O]\varphi$$
$$O ::= cv \mid \neg cv \mid ch \mid \neg ch \mid k \mid \neg k \mid d \mid \neg d \mid \bullet^{\exists} O \mid \bullet^{\forall} O$$
$$\mid O \vee O \mid O \wedge O$$

We use the classic abbreviations $\rightarrow, \leftrightarrow$ and the usual definitions for true and false. We also introduce the temporal abbreviations $\mathsf{F}\varphi \equiv \text{true} \, \mathcal{U} \, \varphi$ (*eventually*), $\mathsf{G}\varphi \equiv \neg\mathsf{F}\neg\varphi$ (*globally*) and $\varphi \, \mathcal{W} \, \psi \equiv \psi \mathcal{R}(\psi \vee \varphi)$ (*weak until*). Furthermore we assume that all variables are Boolean because every finite domain can be encoded by multiple Boolean variables. For convenience we will, however, use non-Boolean variables when relating to our RMS example.

The syntax of LTOL is presented in *positive normal form* to facilitate translation into alternating Büchi automata (ABW) as shown later. We, therefore, use $\overline{\Theta}$ to denote the dual of formula Θ where Θ ranges over either φ or O. Intuitively, $\overline{\Theta}$ is obtained from Θ by switching $\vee$ and $\wedge$ and by applying dual to sub formulas, e.g., $\overline{\varphi_1 \, \mathcal{U} \, \varphi_2} = \overline{\varphi_1} \, \mathcal{R} \, \overline{\varphi_2}$, $\overline{\varphi_1 \wedge \varphi_2} = \overline{\varphi_1} \vee \overline{\varphi_2}$, $\overline{cv} = \neg cv$, and $\overline{\bullet^{\exists} O} = \bullet^{\forall} \overline{O}$.

Observation descriptors are built from referring to the different parts of the observations and their Boolean combinations. Thus, they refer to the channel in CH, the data variables in D, the sender k, and the predicate over common variables in CV. These predicates are interpreted as sets of possible assignments to common variables, and therefore we include existential $\bullet^{\exists} O$ and universal $\bullet^{\forall} O$ quantifiers over these assignments.

The semantics of an observation descriptor O is defined for an observation $m = (ch, \mathbf{d}, k, \pi)$ as follows:

$m \models ch'$	**iff**	$ch = ch'$	$m \models \neg ch'$	**iff**	$ch \neq ch'$
$m \models d'$	**iff**	$\mathbf{d}(d')$	$m \models \neg d'$	**iff**	$\neg\mathbf{d}(d')$
$m \models k'$	**iff**	$k = k'$	$m \models \neg k'$	**iff**	$k \neq k'$

$m \models cv$ **iff**   for all $c \in \pi$ we have $c \models cv$
$m \models \neg cv$ **iff** there is $c \in \pi$ such that $c \not\models cv$
$m \models \bullet^{\exists} O$ **iff** there is $c \in \pi$ such that $(ch, d, k, \{c\}) \models O$
$m \models \bullet^{\forall} O$ **iff** for all $c \in \pi$ it holds that $(ch, d, k, \{c\}) \models O$
$m \models O_1 \vee O_2$ **iff** either $m \models O_1$ or $m \models O_2$
$m \models O_1 \wedge O_2$ **iff** $m \models O_1$ and $m \models O_2$

We only comment on the semantics of the descriptors $\bullet^{\exists} O$ and $\bullet^{\forall} O$ and the rest are standard propositional formulas. The descriptor $\bullet^{\exists} O$ requires that at least one assignment c to the common variables in the sender predicate π satisfies O. Dually $\bullet^{\forall} O$ requires that all assignments in π satisfy O. Using the former, we express properties where we require that the sender predicate has a possibility to satisfy O while using the latter we express properties where the sender predicate can only satisfy O. For instance, both observations $(ch, \mathbf{d}, k, cv_1 \vee \neg cv_2)$ and $(ch, \mathbf{d}, k, cv_1)$ satisfy $\bullet^{\exists} cv_1$ while only the latter satisfies $\bullet^{\forall} cv_1$. Furthermore, the observation descriptor $\bullet^{\forall} \text{false} \wedge ch = \star$ says that a message is sent on the broadcast channel with a false predicate. That is, the message cannot be received by other agents. In our RMS example in Section 5, the descriptor $\bullet^{\exists}(@\text{type} = \text{t1}) \wedge \bullet^{\forall}(@\text{type} = \text{t1})$ says that the message is intended exactly for robots of type-1.

Note that the semantics of $\bullet^{\exists} O$ and $\bullet^{\forall} O$ (when nested) ensures that the outermost cancels the inner ones, e.g., $\bullet^{\exists}(O_1 \vee (\bullet^{\forall}(\bullet^{\exists} O_2)))$ is equivalent to $\bullet^{\exists}(O_1 \vee O_2)$. Thus, we assume that they are written in the latter normal form.

We interpret LTOL formulas over system computations:

Definition 6.1 (System computation). A system computation ρ is a function from natural numbers N to $2^{\mathcal{V}} \times M$ where $\mathcal{V}$ is the set of state variable propositions and $M = \text{CH} \times 2^{D} \times K \times 2^{2^{\text{CV}}}$ is the set of possible observations. That is, ρ includes values for the variables in $2^{\mathcal{V}}$ and an observation in M at each time instant.

We denote by s_i the system state at the i-th time point of the system computation. Moreover, we denote the suffix of ρ starting with the i-th state by $\rho_{\geq i}$ and we use m_i to denote the observation $(ch, \mathbf{d}, k, \pi)$ in ρ at time point i.

The semantics of an LTOL formula φ is defined for a computation ρ at a time point i as follows:

$\rho_{\geq i} \models v$ **iff** $s_i \models v$ and $\rho_{\geq i} \models \neg v$ **iff** $s_i \not\models v$;
$\rho_{\geq i} \models \varphi_2 \vee \varphi_2$ **iff** $\rho_{\geq i} \models \varphi_1$ or $\rho_{\geq i} \models \varphi_2$;
$\rho_{\geq i} \models \varphi_2 \wedge \varphi_2$ **iff** $\rho_{\geq i} \models \varphi_1$ and $\rho_{\geq i} \models \varphi_2$;
$\rho_{\geq i} \models \varphi_1 \, \mathcal{U} \, \varphi_2$ **iff** there exists $j \geq i$ s.t. $\rho_{\geq j} \models \varphi_2$ and, for every $i \leq k < j$, $\rho_{\geq k} \models \varphi_1$;
$\rho_{\geq i} \models \varphi_1 \, \mathcal{R} \, \varphi_2$ **iff** for every $j \geq i$ either $\rho_{\geq j} \models \varphi_2$ or there exists $i \leq k < j$ s.t. $\rho_{\geq k} \models \varphi_1$;
$\rho_{\geq i} \models \langle O \rangle \varphi$ **iff** $m_i \models O$ and $\rho_{\geq i+1} \models \varphi$;
$\rho_{\geq i} \models [O]\varphi$ **iff** $m_i \models O$ implies $\rho_{\geq i+1} \models \varphi$.

Intuitively, the temporal formula $\langle O \rangle \varphi$ is satisfied on the computation ρ at point i if the observation m_i satisfies O *and* φ is satisfied on the suffix computation $\rho_{\geq i+1}$. On the other hand, the formula $[O]\varphi$ is satisfied on the computation ρ at point i if m_i satisfying O *implies* that φ is satisfied on the suffix computation $\rho_{\geq i+1}$. Other formulas are interpreted exactly as in classic LTL.

With observation descriptors we can refer to the intention of senders. For example, $O_1 := \bullet^{\exists}(@\text{type} = \text{t1}) \wedge \bullet^{\exists}(@\text{type} = \text{t2}) \wedge \bullet^{\forall}(@\text{type} = \text{t1} \vee @\text{type} = \text{t2})$ specifies that the target of the message is "exactly and only" both type-1 and type-2 robots. Thus, we may specify that whenever the line agent "l" recruits for a product with identity 1, it notifies both type-1 and type-2 robots: $\mathsf{G}((\text{prd} = \text{1} \wedge \text{st} = \text{pnd} \wedge \langle l \wedge ch = \star \rangle \text{true}) \rightarrow \langle O_1 \rangle \text{true})$. Namely, whenever the line agent is in the pending state and tasked with product 1 it notifies both type-1 and type-2 robots by a broadcast. The pattern "After q have exactly two p until r" [16, 32] can be easily expressed in LTL and can be used to check the formation protocol. Consider the following formulas. Let $\varphi_1 := \langle \text{MSG} = \text{team} \wedge \text{NO} = \text{2} \wedge \bullet^{\exists}(@\text{type} = \text{t1}) \rangle \text{true}$, i.e., a team message is sent to type-1 robots requiring two robots. Let $\varphi_2 := \langle \text{MSG} = \text{form} \wedge \bullet^{\exists}(@\text{type} = \text{t1}) \rangle \text{true}$, i.e., a formation message is sent to type-1 robots. Let $\varphi_3 := \langle ch = A \rangle \text{true}$, i.e., a message is sent on channel A. Then, "After φ_1 have exactly two φ_2 until φ_3" says that whenever a team message is sent to robots of type-1 requiring two robots, then two form messages destined for type-1 robots will follow before using the multicast channel. That is, two type-1 robots join the team before a (blocking) multicast on channel A may become possible.

We can also reason at a local rather than a global level. For instance, we can specify that robots follow a "correct" utilisation of channel A. Formally, let $O_1(t) := \text{MSG=team} \wedge \bullet^{\exists}(@\text{type}=t)$, i.e., a team message is sent to robots of type t; $O_2(k, t) := \text{MSG=form} \wedge \neg k \wedge \text{NO=0} \wedge \bullet^{\exists}(\neg@\text{asgn} \wedge @\text{type}=t)$, i.e., a robot different from k sends a form message saying no more robots are needed and this message is sent to unassigned type t robots; and let $O_3(t) := \text{MSG=asmb1} \wedge ch = A \wedge @\text{rdy=2} \wedge \bullet^{\exists}(@\text{type}=t)$, i.e., an assembly message is sent on

channel A to robots of type t who reached stage 2 of the production. Thus, for robot k of type t, the formulas

(1) $\varphi_1(t) := (\text{lnk} \neq A) \mathcal{W} \langle O_1(t) \rangle \text{true}$
(2) $\varphi_2(k, t) := G([O_2(k, t) \vee O_3(t)] \varphi_1(t))$

state that: (1) robots are not connected to channel A until they get a team message, inviting them to join a team; (2) if either they are not selected $(O_2(k, t))$ or they finished production after selection $(O_3(t))$ then they disconnect again until the next team message. This reduces to checking the "correct" utilisation of channel A to individual level, by verifying these properties on all types of robots independently. By allowing the logic to relate to the set of targeted robots, verifying all targeted robots separately entails the correct "group usage" of channel A.

To model-check LTOL formulas we first need to translate them to Büchi automata. The following theorem states that the set of computations satisfying a given formula are exactly the ones accepted by some finite automaton on infinite words. Before we proceed with the theorem, we fix the following notations. Given a tuple $T = (t_1, \ldots, t_n)$, we use $T[i]$ to denote the element t_i of T and $T[x/i]$ to denote the tuple where t_i is replaced with x.

THEOREM 6.2. *For every LTOL formula φ, there is an ABW $A_\varphi = \langle Q, \Sigma, M, \delta_\varphi, q_0, F \subseteq Q \rangle$ such that $L_\omega(A_\varphi)$ is exactly the set of computations satisfying the formula φ.*

PROOF. The set of states Q is the set of all sub formulas of φ with φ being the initial state q_0. The automaton has two alphabets, namely a *state-alphabet* $\Sigma = 2^{\mathcal{V}}$ and a *message-alphabet* $M = \text{CH} \times 2^{\text{D}} \times K \times 2^{2^{\text{CV}}}$. The set F of accepting states consists of all sub formulas of the form $\varphi_1 \mathcal{R} \varphi_2$. The transition relation $\delta_\varphi : Q \times \Sigma \times M \rightarrow \mathcal{B}^+(Q)$ is defined inductively on the structure of φ. It also relies on an auxiliary function $f : O \times M \rightarrow \mathbb{B}$ to evaluate observations and is defined recursively on O as follows:

- $f(cv, m) = \bigwedge_{c \in m[4]} c(cv)$ and
  $f(\neg cv, m) = \bigvee_{c \in m[4]} \neg c(cv)$;
- $f(ch, m) = \text{true if } m[1] = ch$ and false otherwise;
- $f(\neg ch, m) = \text{true if } m[1] \neq ch$ and false otherwise;
- $f(d, m) = \text{true if } m[2](d)$ and false otherwise;
- $f(\neg d, m) = \text{true if } \neg m[2](d)$ and false otherwise;
- $f(k, m) = \text{true if } m[3] = k$ and false otherwise;
- $f(\neg k, m) = \text{true if } m[3] \neq k$ and false otherwise;
- $f(O_1 \vee O_2, m) = f(O_1, m) \vee f(O_2, m)$ and
  $f(O_1 \wedge O_2, m) = f(O_1, m) \wedge f(O_2, m)$;
- $f(\bullet^{\exists} O, m) = \bigvee_{c \in m[4]} f(O, m[c/4])$;
- $f(\bullet^{\forall} O, m) = \bigwedge_{c \in m[4]} f(O, m[c/4])$.

The transition relation δ_φ is defined as follows:
- $\delta_\varphi(v, \sigma, m) = \text{true if } v \in \sigma$ and false otherwise;
- $\delta_\varphi(\neg v, \sigma, m) = \text{true if } v \notin \sigma$ and false otherwise;
- $\delta_\varphi(\varphi_1 \vee \varphi_2, \sigma, m) = \delta_\varphi(\varphi_1, \sigma, m) \vee \delta_\varphi(\varphi_2, \sigma, m)$;
- $\delta_\varphi(\varphi_1 \wedge \varphi_2, \sigma, m) = \delta_\varphi(\varphi_1, \sigma, m) \wedge \delta_\varphi(\varphi_2, \sigma, m)$;
- $\delta_\varphi(\varphi_1 \mathcal{U} \varphi_2, \sigma, m) = \delta_\varphi(\varphi_1, \sigma, m) \wedge \varphi_1 \mathcal{U} \varphi_2 \vee \delta_\varphi(\varphi_2, \sigma, m)$;
- $\delta_\varphi(\varphi_1 \mathcal{R} \varphi_2, \sigma, m) = (\delta_\varphi(\varphi_1, \sigma, m) \vee \varphi_1 \mathcal{R} \varphi_2) \wedge \delta_\varphi(\varphi_2, \sigma, m)$;
- $\delta_\varphi(\langle O \rangle \varphi, \sigma, m) = \varphi \wedge f(O, m)$;
- $\delta_\varphi([O]\varphi, \sigma, m) = \varphi \vee f(\overline{O}, m)$.

The proof of correctness of this construction proceeds by induction on the structure of φ. □

Note that, from Theorem 6.2, the number of states in A_φ is linear in the size of φ, i.e., $|Q|$ is in $\mathcal{O}(|\varphi|)$. The size of the transition relation $|\delta_\varphi|$ is in $\mathcal{O}(|Q|^2.|\Sigma|.|M|)$, i.e., it is in $|\varphi|^2.|\text{CH}|.|K|.2^{\mathcal{O}(|\mathcal{V}|+|\text{D}|+2^{|\text{CV}|})}$. Furthermore the function f can be computed in $\mathcal{O}(|O|.|\text{cv}|)$ time and in $\mathcal{O}(\log|O| + \log|\text{cv}|)$ space. Finally, the size of the alternating automaton $|A_\varphi|$ is in

$\mathcal{O}(|Q|.|\delta_\varphi|)$, i.e., $|A_\varphi|$ is in $(|\varphi|)^3.|\text{CH}|.|K|.2^{\mathcal{O}(|\mathcal{V}|+|\text{D}|+2^{|\text{CV}|})}$

PROPOSITION 6.3 ([36]). *For every alternating Büchi automaton A there is a nondeterministic Büchi automaton A' such that $L_\omega(A') = L_\omega(A)$ and $|Q'|$ is in $2^{\mathcal{O}(|Q|)}$.*

By Theorem 6.2 and Proposition 6.3, we have that:

COROLLARY 6.4. *For every formula φ there is a Büchi automaton A with a state-alphabet $\Sigma = 2^{\mathcal{V}}$ and a message-alphabet $M = \text{CH} \times 2^{\text{D}} \times K \times 2^{2^{\text{CV}}}$ where $A = \langle Q, \Sigma, M, S^0, \delta, F \rangle$ and $L_\omega(A)$ is exactly the set of computations satisfying the formula φ such that:*
- *$|Q|$ is in $2^{\mathcal{O}(|\varphi|)}$ and $|\delta|$ is in $\mathcal{O}(|Q|^2.|\Sigma|.|M|)$, i.e., $|\delta|$ is in $|\text{CH}|.|K|.2^{\mathcal{O}(|\varphi|+|\mathcal{V}|+|\text{D}|+2^{|\text{CV}|})}$*
- *The required space for building the automaton is NLOG($|Q|.|\delta|$), i.e., it is in $\mathcal{O}(\log|\text{CH}| + \log|K| + |\varphi| + |\mathcal{V}| + |\text{D}| + 2^{|\text{CV}|})$*
- *The size of the Büchi automaton is $|Q|.|\delta|$, i.e., $|A|$ is in $|\text{CH}|.|K|.2^{\mathcal{O}(|\mathcal{V}|+|\text{D}|+2^{|\text{CV}|})}$*

We do not see the double exponential in the automaton size with respect to common variables CV as a major restriction. Note that $|\text{cv}|$ does not grow with respect to the size of the formula or to the number of the agents. Thus, if we limit the number of common variables (which should be small), efficient verification can still be attainable.

THEOREM 6.5. *The satisfiability problem of LTOL is PSPACE-complete with respect to $|\varphi|$, $|\mathcal{V}|$, $|\text{D}|$, $\log|\text{CH}|$, $\log|K|$ and EXPSPACE with respect to $|\text{CV}|$.*

THEOREM 6.6. *The model-checking problem of LTOL is PSPACE-complete with respect to $|\text{Sys}|$, $|\varphi|$, $|\mathcal{V}|$, $|\text{D}|$, $\log|\text{CH}|$, $\log|K|$ and EXPSPACE with respect to $|\text{CV}|$.*

7 CONCLUDING REMARKS

We introduced a formalism that combines message-passing and shared-memory to facilitate realistic modelling of distributed MAS. A system is defined as a set of distributed agents that execute concurrently and only interact by message-passing. Each agent controls its local behaviour as in Reactive Modules [5, 19] while interacting externally by message passing as in π-calculus-like formalisms [1, 35]. Thus, we decouple the individual behaviour of an agent from its external interactions to facilitate reasoning about either one separately. We also make it easy to model interaction features of MAS, that may only be tediously hard-coded in existing formalisms.

We introduced an extension to LTL, named LTOL, that characterises messages and their targets. This way we may not only be able to reason about the intentions of agents in communication, but also we may explicitly specify their interaction protocols. Finally, we showed that the model-checking problem for LTOL is EXPSPACE only with respect to the number of common variables and PSPACE-complete with respect to the rest of the input.

Related works. As mentioned before, formal modelling is highly influenced by traditional formalisms used for verification, see [5, 17]. These formalisms are, however, very abstract in that their models representations are very close to their mathematical interpretations (i.e., the underlying transition systems). Although this may make it easy to conduct some logical analysis [6, 14, 37] on models, it does imply that most of the high-level MAS features may only be hard-coded, and thus leading to very detailed models that may not be tractable or efficiently implementable. This concern has been already recognised and thus more formalisms have been proposed, e.g., *Interpreted Systems Programming Language* (ISPL) [30] and MOCHA [7] are proposed as implementation languages of *Interpreted Systems* (IS) [17] and *Reactive Modules* (RM) [5] respectively. They are still either fully synchronous or shared-memory based and thus do not support flexible coordination and/or interaction interfaces. A recent attempt to add dynamicity in this sense has been adopted by *visibly CGS* (vCGS) [10]: an extension of *Concurrent-Game Structures* (CGS) [6] to enable agents to dynamically hide/reveal their internal states to selected agents. However, vCGS relies on an assumption of [8] which requires that agents know the identities of each other. This, however, only works for closed systems with a fixed number of agents.

Other attempts to add dynamicity and reconfiguration include dynamic I/O automata [13], Dynamic reactive modules of Alur and Grosu [4], Dynamic reactive modules of Fisher et al. [19], and open MAS [28]. However, their main interest was in supporting dynamic creation of agents. Thus, the reconfiguration of communication was not their main interest. While RECIPE may be easily extended to support dynamic creation of agents, none of these formalisms may easily be used to control the targets of communication and dissemination of information.

As for logics we differ from traditional languages like LTL and CTL in that our formula may refer to messages and their constraints. This is, however, different from the atomic labels of PDL [34] and modal μ-calculus [29] in that LTOL mounts complex and structured observations on which designers may predicate on. Thus the interpretation of a formula includes information about the causes of variable assignments and the interaction protocols among agents. Such extra information may prove useful in developing compositional verification techniques.

Future works. We plan to provide tool support for RECIPE and LTOL, but with a more user-friendly syntax.

We want to exploit the interaction mechanisms in RECIPE and the extra information in LTOL formulas to conduct verification compositionally. As mentioned, we believe that relating to sender intentions will facilitate that.

We intend to study the relation with respect to temporal epistemic logic [24]. Although we do not provide explicit knowledge operators, the combination of data exchange, receivers selection, and enabling/disabling of synchronisation allow agents to dynamically deduce information about each other. Furthermore we want to extend RECIPE to enable dynamic creation of agents while reconfiguring communication.

Finally, we want to target the distributed synthesis problem [18]. Several fragments of the problem have been proven to be decidable, e.g., when the information of agents is arranged hierarchically [12], the number of agents is limited [23], or the actions are made public [11]. We conjecture that the ability to disseminate information and reason about it might prove useful in this setting.

REFERENCES

[1] Yehia Abd Alrahman, Rocco De Nicola, and Michele Loreti. 2019. A calculus for collective-adaptive systems and its behavioural theory. *Inf. Comput.* 268 (2019). https://doi.org/10.1016/j.ic.2019.104457
[2] Marcos K. Aguilera, Naama Ben-David, Irina Calciu, Rachid Guerraoui, Erez Petrank, and Sam Toueg. 2018. Passing Messages While Sharing Memory. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing (PODC '18)*. ACM, New York, NY, USA, 51–60. https://doi.org/10.1145/3212734.3212741
[3] Marcos K. Aguilera, Naama Ben-David, Rachid Guerraoui, Virendra Marathe, and Igor Zablotchi. 2019. The Impact of RDMA on Agreement. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC '19)*. ACM, New York, NY, USA, 409–418. https://doi.org/10.1145/3293611.3331601
[4] Rajeev Alur and Radu Grosu. 2004. *Dynamic Reactive Modules*. Technical Report 2004/6. Stony Brook.
[5] Rajeev Alur and Thomas Henzinger. 1999. Reactive Modules. *Formal Methods in System Design* 15, 1 (1999), 7–48.
[6] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. 2002. Alternating-Time Temporal Logic. *J. ACM* 49, 5 (2002), 672–713. https://doi.org/10.1145/585265.585270
[7] Rajeev Alur, Thomas A. Henzinger, Freddy Y. C. Mang, Shaz Qadeer, Sriram K. Rajamani, and Serdar Tasiran. 1998. MOCHA: Modularity in Model Checking.. In *CAV'98*. 521–525.
[8] Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. 1995. Sharing Memory Robustly in Message-passing Systems. *J. ACM* 42, 1 (Jan. 1995), 124–142. https://doi.org/10.1145/200836.200869
[9] Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schüpbach, and Akhilesh Singhania. 2009. The Multikernel: A New OS Architecture for Scalable Multicore Systems. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles (SOSP '09)*. ACM, New York, NY, USA, 29–44. https://doi.org/10.1145/1629575.1629579
[10] Francesco Belardinelli, Ioana Boureanu, Catalin Dima, and Vadim Malvone. 2019. Verifying Strategic Abilities in Multi-agent Systems with Private Data-Sharing. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019*. 1820–1822.
[11] Francesco Belardinelli, Alessio Lomuscio, Aniello Murano, and Sasha Rubin. 2017. Verification of Multi-agent Systems with Imperfect Information and Public Actions. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017*. 1268–1276.
[12] Raphaël Berthon, Bastien Maubert, Aniello Murano, Sasha Rubin, and Moshe Y. Vardi. 2017. Strategy logic with imperfect information. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*. 1–12. https://doi.org/10.1109/LICS.2017.8005136
[13] Paul C. Attie and Nancy A. Lynch. 2016. Dynamic input/output automata: A formal and compositional model for dynamic systems. *Inf. Comput.* 249 (2016), 28–75. https://doi.org/10.1016/j.ic.2016.03.008
[14] Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. 2010. Strategy logic. *Inf. Comput.* 208, 6 (jun 2010), 677–693. https://doi.org/10.1016/j.ic.2009.07.004
[15] Marco Dorigo. 2014. The Swarm-bots and swarmanoid experiments in swarm robotics. In *2014 IEEE International Conference on Autonomous Robot Systems and Competitions, ICARSC 2014, Espinho, Portugal, May 14-15, 2014*. 1. https://doi.org/10.1109/ICARSC.2014.6849753
[16] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. 1999. Patterns in Property Specifications for Finite-State Verification. In *Proceedings of the 1999 International Conference on Software Engineering, ICSE' 99, Los Angeles, CA, USA, May 16-22, 1999*. ACM, 411–420.
[17] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. 1995. *Reasoning about Knowledge*. MIT Press.
[18] Bernd Finkbeiner and Sven Schewe. 2005. Uniform Distributed Synthesis. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*. IEEE, 321–330. https://doi.org/10.1109/LICS.2005.53
[19] Jasmin Fisher, Thomas A. Henzinger, Dejan Nickovic, Nir Piterman, Anmol V. Singh, and Moshe Y. Vardi. 2011. Dynamic Reactive Modules. In *CONCUR 2011 - Concurrency Theory - 22nd International Conference, CONCUR 2011, Aachen, Germany, September 6-9, 2011. Proceedings*. 404–418. https://doi.org/10.1007/978-3-642-23217-6_27
[20] Paul Gochet and E. Pascal Gribomont. 2006. Epistemic logic. In *Logic and the Modalities in the Twentieth Century*. 99–195. https://doi.org/10.1016/S1874-5857(06)80028-2

[21] Andreas Griesmayer and Alessio Lomuscio. 2013. Model Checking Distributed Systems against Temporal-Epistemic Specifications. In *Formal Techniques for Distributed Systems - Joint IFIP WG 6.1 International Conference, FMOODS/FORTE 2013, Held as Part of the 8th International Federated Conference on Distributed Computing Techniques, DisCoTec 2013, Florence, Italy, 2013. Proceedings.* 130–145. https://doi.org/10.1007/978-3-642-38592-6_10

[22] Julian Gutierrez, Paul Harrenstein, and Michael Wooldridge. 2017. From Model Checking to Equilibrium Checking: Reactive Modules for Rational Verification. *Artif. Intell.* 248 (2017), 123–157. https://doi.org/10.1016/j.artint.2017.04.003

[23] Julian Gutierrez, Giuseppe Perelli, and Michael Wooldridge. 2018. Imperfect information in Reactive Modules games. *Inf. Comput.* 261, Part (2018), 650–675.

[24] Joseph Y. Halpern and Moshe Y. Vardi. 1989. The Complexity of Reasoning about Knowledge and Time. I. Lower Bounds. *J. Comput. Syst. Sci.* 38, 1 (1989), 195–237. https://doi.org/10.1016/0022-0000(89)90039-1

[25] Heiko Hamann. 2018. *Swarm Robotics - A Formal Approach.* Springer. https://doi.org/10.1007/978-3-319-74528-2

[26] Aaron Helsinger, Michael Thome, and Todd Wright. 2004. Cougaar: a scalable, distributed multi-agent architecture. In *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)*, Vol. 2. 1910–1917 vol.2. https://doi.org/10.1109/ICSMC.2004.1399959

[27] Panagiotis Karagiannis, Stereos Alexandros Matthaiakis, Dionisis Andronas, Konstantinos Filis, Christos Giannoulis, George Michalos, and Sotiris Makris. 2019. Reconfigurable Assembly Station: A Consumer Goods Industry Paradigm. *Procedia CIRP* 81 (2019), 1406–1411. https://doi.org/10.1016/j.procir.2019.04.070 52nd CIRP Conference on Manufacturing Systems (CMS), Ljubljana, Slovenia, June 12-14, 2019.

[28] Panagiotis Kouvaros, Alessio Lomuscio, Edoardo Pirovano, and Hashan Punchihewa. 2019. Formal Verification of Open Multi-Agent Systems. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019.* 179–187.

[29] Dexter Kozen. 1983. Results on the Propositional mu-Calculus. *Theor. Comput. Sci.* 27 (1983), 333–354. https://doi.org/10.1016/0304-3975(82)90125-6

[30] Alessio Lomuscio, Hongyang Qu, and Franco Raimondi. 2017. MCMAS: an opensource model checker for the verification of multi-agent systems. *STTT* 19, 1 (2017), 9–30.

[31] Isabela Maganha, Cristovao Silva, and Luis Miguel D. F. Ferreira. 2019. The Layout Design in Reconfigurable Manufacturing Systems: a Literature Review. *The International Journal of Advanced Manufacturing Technology* (2019). https://doi.org/10.1007/s00170-019-04190-3

[32] Shahar Maoz and Jan Oliver Ringert. 2015. GR(1) synthesis for LTL specification patterns. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015.* ACM, 96–106.

[33] Nithin Mathews, Anders Lyhne Christensen, Eliseo Ferrante, Rehan O'Grady, and Marco Dorigo. 2010. Establishing Spatially Targeted Communication in a Heterogeneous Robot Swarm. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 (AAMAS '10).* International Foundation for Autonomous Agents and Multiagent Systems, 939–946.

[34] Michael J. Fischer and Richard E. Ladner. 1979. Propositional Dynamic Logic of Regular Programs. *J. Comput. Syst. Sci.* 18, 2 (1979), 194–211. https://doi.org/10.1016/0022-0000(79)90046-1

[35] Robin Milner, Joachim Parrow, and David Walker. 1992. A Calculus of Mobile Processes, I. *Inf. Comput.* 100, 1 (1992), 1–40. https://doi.org/10.1016/0890-5401(92)90008-4

[36] Satoru Miyano and Takeshi Hayashi. 1984. Alternating Finite Automata on omega-Words. *Theor. Comput. Sci.* 32 (1984), 321–330. https://doi.org/10.1016/0304-3975(84)90049-5

[37] Fabio Mogavero, Aniello Murano, Giuseppe Perelli, and Moshe Y. Vardi. 2014. Reasoning About Strategies: On the Model-Checking Problem. *TOCL* 15, 4. doi:10.1145/2631917.

[38] Michael Rubenstein, Christian Ahler, Nick Hoff, Adrian Cabrera, and Radhika Nagpal. 2014. Kilobot: A low cost robot with scalable operations designed for collective behaviors. *Robotics and Autonomous Systems* 62, 7 (2014), 966–975. https://doi.org/10.1016/j.robot.2013.08.006

[39] Cheng Wang, Jianyu Jiang, Xusheng Chen, Ning Yi, and Heming Cui. 2017. APUS: fast and scalable paxos on RDMA. In *Proceedings of the 2017 Symposium on Cloud Computing, SoCC 2017, Santa Clara, CA, USA, September 24-27, 2017.* 94–107. https://doi.org/10.1145/3127479.3128609

[40] Kai Weng Wong and Hadas Kress-Gazit. 2016. Need-based coordination for decentralized high-level robot control. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2016, Daejeon, South Korea, October 9-14, 2016.* 2209–2216. https://doi.org/10.1109/IROS.2016.7759346

[41] Michael Wooldridge. 2002. *Introduction to Multiagent Systems.* Wiley.