Autonomous Agents and Multi-Agent Systems (2021) 35:47
https://doi.org/10.1007/s10458-021-09521-x

Modelling and verification of reconfigurable multi-agent systems

Yehia Abd Alrahman[1] · **Nir Piterman**[1]

Accepted: 28 June 2021 / Published online: 26 August 2021

Abstract
We propose a formalism to model and reason about reconfigurable multi-agent systems. In our formalism, agents 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. Inspired by existing multi-robot systems, we represent a system as a set of agents (each with local state), executing independently and only influence each other by means of message exchange. Agents are able to sense their local states and partially their surroundings. We extend LTL to be able to reason explicitly about the intentions of agents in the interaction and their communication 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 [42,61]. Several mathematical formalisms have been suggested to represent the behaviour of such systems and to reason about the strategies that agents exhibit [9,42]. For instance, modelling languages, such as RM [8,33] and ISPL [42], 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 and epistemic operators [27,31]) specifically to support multi-agent modelling [32]. Similarly,

This work is funded by the ERC consolidator Grant D-SynMA (No. 772459) and the Swedish research council Grants: SynTM (No. 2020-03401) and VR Project (No. 2020-04963). The authors would like to thank Giuseppe Perelli who was a co-author on an earlier version of this article.

✉ Yehia Abd Alrahman
yehia.abd.alrahman@gu.se

✉ Nir Piterman
nir.piterman@gu.se

1 University of Gothenburg, Gothenburg, Sweden

logics that support reasoning about the intentions and strategic abilities of such agents have been used and extended [22,50].

These works are heavily influenced by the formalisms used for verification (e.g., Reactive Modules [6,8], concurrent game structures [9], and interpreted systems [42]). 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 [13,37]. 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.

Furthermore, the compositionality of shared memory approaches is limited and the supported interaction interfaces are in general not very flexible [14]. Alternatively, message passing formalisms [48] 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. The reason is that they model agents as mathematical expressions over interaction operators. Thus the state of an agent is implicit in the structure of the expression.

Based on an early result, where a translation from shared memory to message passing was provided [11], 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. As discussed in [3], the translation was not provided in a standard way where every feature of one formalism is efficiently translated to a corresponding one in the other formalism, but rather based on emulation. That is, it computationally shows if a problem has a solution in one formalism it also has one in the other formalism. However, this is not surprising as most computational formalisms are Turing powerful. A good translation (See [2]) should also preserve the observable behaviour of the translated process and its divergence tendencies. This is important in distributed settings because the observable behaviour of a process defines its communication capabilities, which can be influenced in open world settings. Thus the above mentioned translation only works under closed world assumption and does not capture divergence. Namely, a timely process in one formalism may diverge indefinitely while awaiting for other processes. Furthermore, the translation cannot deal with failure or anonymous interaction, and thus requires that there is a majority of correct processes and a pre-defined knowledge of each other's identities and the number of processes in the systems. Thus, there is no way to model the birth/death of processes in the system.

To combine the benefits of both approaches recent developments [3,59] suggest adopting hybrids, that accurately represent actual distributed systems, e.g., [4,46]. 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 [52] and Swarmanoid [23], agents are additionally able to sense their local states and partially their surroundings. Interaction is driven by message passing following the interleaving semantics of [48]; 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 [60], 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 [46], messages destined for particular agents [2], 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., [36]. 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.

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.

This article is an extended and revised version of the conference paper presented in [1]. The major extensions in this article consist of: (i) a compositional and enumerative semantic definition of the proposed formalism, that coincides with the early symbolic one. The new definition facilitates reasoning about the individual behaviour of agents and their compositions with others. For this purpose, we defined a parallel composition operator with reconfigurable broadcast and multicast semantics. Thus, the definition is not only intuitive, but can also be used to reason about models under *open-world assumption*; (ii) a major improvement on our early results [1] regarding satisfiability and model checking, that were computed in an EXPSPACE upper bound. Here, we provide a novel automata construction that permits PSPACE analysis, matching the lower bound. Thus, this part is majorly rewritten and improved. Moreover, we enhance the presentation of the different parts of the article and provide the proofs of all results.

The structure of this article is as follows: In Sect. 2, we informally present our formalism and motivate our design choices. In Sect. 3, we give the necessary background and in Sect. 4 we present the compositional semantic definition. In Sect. 5 we introduce the formalism both in terms of enumerative and symbolic semantics, and we prove that they coincide. In Sect. 6, we present a non-trivial case study to show the distinctive features of our formalism. In Sect. 7 we discuss our extension to LTL and provide efficient decision procedures to check both satisfiability and model checking in polynomial space. Finally, in Sect. 8 we report closely related works and in Sect. 9 we discuss our concluding remarks.

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 Sect. 6. The scenario is based on Reconfigurable Manufacturing Systems (RMS) [38,44], 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 $\star$. 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 (i.e., 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 $\star$. 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 variable names* CV (*common variables*, for short). Each agent has local variables that are identified by these names. Agents assign their own individual information to these local variables (e.g., the type of agent, its location, its readiness, etc.). Send guards are expressed in terms of conditions on these names 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 the 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 and finite automata

We unify notations and give the necessary background. We introduce doubly-labeled transition systems and discrete systems and show how to translate the former to the latter. We further introduce nondeterministic and alternating Büchi word automata.

3.1 Transition systems and discrete systems

A *Doubly-Labeled Transition System* (TS) 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 labeling 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 $j \geq 0$ 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 transition system* and if $|\Sigma| = 1$ then $\mathscr{T}$ is a *transition-labeled transition 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:

- $\mathscr{V} = \{v_1, \ldots, v_n\}$: 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 $\prod_{v_i \in \mathscr{V}} D_{v_i}$.
  We assume some underlying first-order language over $\mathscr{V}$ that includes (i) *expressions* constructed from the variables in $\mathscr{V}$, (ii) *atomic formulas* that are either Boolean variables or the application of different predicates to expressions, and (iii) *assertions* that are first-order formulas constructed from atomic formulas using Boolean connectives or quantification of variables. Assertions, also sometimes called *state formulas*, characterize states through restriction of possible variable values in them.

- 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., $(s, s') \models \rho$, 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 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 $(s, s') \models \rho$. 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, υ, s') such that $(s, \upsilon, s') \models \rho$, where s supplies the interpretation to $\mathscr{V}$, υ to $\mathscr{V}_\Upsilon$ and s' to $\mathscr{V}'$.

3.2 Finite automata on infinite words

We use the automata-theoretic approach to linear temporal logic [58]. Thus, we translate temporal logic formulas to automata. We give here the necessary background.

For an alphabet Σ, the set Σ^ω is the set of infinite sequences of elements from Σ. Given an alphabet Σ and a set D of directions, a *Σ-labeled D-tree* is a pair (T, τ), where $T \subseteq D^*$ is a tree over D and $\tau : T \to \Sigma$ maps each node of T to a letter in Σ. A *path* π of a tree T is a set $\pi \subseteq T$ such that $\epsilon \in \pi$ and for every $x \in \pi$ either x is a leaf in T or there exists a unique $\gamma \in D$ such that $x \cdot \gamma \in \pi$. For $\pi = \gamma_1 \cdot \gamma_2 \cdots$, we write $\tau(\pi)$ for $\tau(\epsilon) \cdot \tau(\gamma_1) \cdot \tau(\gamma_1\gamma_2) \cdots$.

For a finite set X, let $\mathcal{B}^+(X)$ be the set of positive Boolean formulas over X (i.e., Boolean formulas built from elements in X using $\wedge$ and $\vee$), where we also allow the formulas $true$ and $false$. For a set $Y \subseteq X$ and a formula $\theta \in \mathcal{B}^+(X)$, we say that Y *satisfies* θ iff assigning $true$ to elements in Y and assigning $false$ to elements in $X \backslash Y$ makes θ true.

Definition 1 *(Alternating Büchi Word Automata (ABW))* An alternating Büchi word automaton is of the form $A = \langle \Sigma, Q, q_{in}, \delta, F\rangle$, where Σ is the input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \to \mathcal{B}^+(Q)$ is a transition function, $q_{in} \in Q$ is an initial state, and $F \subseteq Q$ specifies a Büchi acceptance condition.

A run of an ABW A on $w = \sigma_0\sigma_1 \ldots$ is a Q-labeled D-tree, (T, τ), where $\tau(\epsilon) = q_{in}$ and, for every $x \in T$, we have $\{\tau(x \cdot \gamma_1), \ldots, \tau(x \cdot \gamma_k)\} \models \delta(\tau(x), \sigma_{|x|})$ where $\{x \cdot \gamma_1, \ldots, x \cdot \gamma_k\}$ is the set of children of x. A run of A is accepting if all its infinite paths satisfy the acceptance condition. For a path π, let $inf(\pi) = \{q \mid q$ appears infinitely often in $\tau(\pi)\}$. A path π is accepting if $inf(\pi) \cap F \neq \emptyset$. Thus, every infinite path in the run tree must visit the acceptance set F infinitely often. The ABW A accepts w if there exists an accepting run on w. We denote by $L_\omega(A)$ the set of words accepted by A.

Definition 2 *(Nondeterministic Büchi Word Automata (NBW))* A NBW is $N = \langle \Sigma, Q, Q_{in}, \delta, F\rangle$, where Σ is an input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, $Q_{in} \subseteq Q$ is a set of initial state, and $F \subseteq Q$ specifies a Büchi acceptance condition.

A run of a NBW N on $w = \sigma_0\sigma_1 \ldots \in \Sigma^\omega$ is a sequence $r = q_0q_1 \ldots \in Q^\omega$ such that $q_0 \in Q_{in}$, and for all $i \geq 0$ we have $q_{i+1} \in \delta(q_i, \sigma_{i+1})$. A run is *accepting* if $inf(r) \cap F \neq \emptyset$. The NBW N accepts w if there exists an accepting run of N on w. We denote by $L_\omega(N)$ the set of words accepted by N.

We state the following well known results about Linear Temporal Logic (LTL), NBW, and ABW (omitting the definition of LTL).

Theorem 1 ([57,58]) *For every LTL formula φ of length n there exist an ABW A_φ with $O(n)$ states such that $L(A_\varphi) = L(\varphi)$.*

Theorem 2 ([49]) *For every ABW A with n states there is an NBW N such that $L_\omega(N) = L_\omega(A)$. The number of states of N is in $2^{\mathcal{O}(n)}$.*

4 Channelled transition systems

In this section, we propose *Channelled Transition System (CTS)* to facilitate compositional modelling of interactive systems. Namely, we extend the format of transition labels of Doubly-Labelled Transition Systems to also specify the role of the transition (i.e., send- or receive-message) and the used communication channels. We define a parallel composition operator that considers both broadcast and multicast semantics and we study its properties. The techniques to prove these results are rather standard. However, we are not familiar with a setup that conveniently allows the existence of transitions to depend on subscription to channels as we suggest below.

4.1 Channelled transition systems (CTS)

A *Channelled Transition System* (CTS) is $\mathcal{T} = \langle C, \Sigma, \Upsilon, S, S_0, R, L, \textsc{ls}\rangle$, where C is a set of channels, including the broadcast channel ($\star$), 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, $L : S \to \Sigma$ is a labelling function, and $\textsc{ls} : S \to 2^C$ is a channel-listening function such that for every $s \in S$ we have $\star \in \textsc{ls}(s)$. We assume that $\Upsilon = \Upsilon^+ \times \{!, ?\} \times C$, for some set Υ^+. That is, every transition labelled with some $\upsilon \in \Upsilon^+$ is either a message send (!) or a message receive (?) on some channel $c \in C$.

A *path* of a CTS $\mathcal{T}$ 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 $(s_i, a_i, s_{i+1}) \in R$. As before, we assume that for every state $s \in S$ there exist $a \in \Upsilon$ and $s' \in S$ such that $(s, a, s') \in R$. Thus, a sequence σ is maximal if it is infinite.

Remark 1 Note that the transition labels a_i of a CTS's path $\sigma = s_0, a_0, s_1, a_1, \ldots$ range over both send (!) and receive (?) transitions. Depending on the underlying semantics of the CTS, send transitions may happen independently regardless of the existence of receivers, e.g., in case of broadcast semantics. However, receive transitions may only happen jointly with some send transition. By allowing CTS's paths to also range over receive transitions, we can model every system as a collection of (open) systems that interact through message exchange. That is, a receive transition in a system is a hole that is closed/filled when composed with a send transition from another system. A complete system (i.e., with filled holes) is called a *closed system*.

The analysis in this article considers closed systems where a system path ranges over send transitions only. In other words, we only consider the messages exchanged within the system under consideration.

The parallel composition of systems is defined below.

Definition 3 *(Parallel Composition)* Given two CTS $\mathscr{T}_i = \langle C_i, \Sigma_i, \Upsilon_i, S_i, S_0^i, R_i, L_i, \textsc{ls}^i\rangle$, where $i \in \{1,2\}$ their composition $\mathscr{T}_1 \parallel \mathscr{T}_2$ is the following CTS $\mathscr{T} = \langle C, \Sigma, \Upsilon, S, S_0, R, L, \textsc{ls}\rangle$, where the components of $\mathscr{T}$ are:

- $C = C_1 \cup C_2$
- $\Sigma = \Sigma_1 \times \Sigma_2$
- $\Upsilon = \Upsilon^1 \cup \Upsilon^2$
- $S = S_1 \times S_2$
- $S_0 = S_0^1 \times S_0^2$
- $R =$

$$\left\{ ((s_1,s_2),(v,!,c),(s_1',s_2')) \;\middle|\; \begin{array}{ll} (s_1,(v,!,c),s_1') \in R_1, c \in \textsc{ls}^2(s_2) \text{ and} & \\ \quad (s_2,(v,?,c),s_2') \in R_2 & \text{or} \\ (s_1,(v,?,c),s_1') \in R_1, c \in \textsc{ls}^1(s_1), \text{ and} & \\ \quad (s_2,(v,!,c),s_2') \in R_2 & \text{or} \\ (s_1,(v,!,c),s_1') \in R_1, c \notin \textsc{ls}^2(s_2), \text{ and} & \\ \quad s_2 = s_2' & \text{or} \\ c \notin \textsc{ls}^1(s_1), s_1 = s_1', \text{ and } (s_2,(v,!,c),s_2') \in R_2 & \end{array} \right\} \cup$$

$$\left\{ ((s_1,s_2),(v,?,c),(s_1',s_2')) \;\middle|\; \begin{array}{ll} c \in \textsc{ls}^1(s_1), (s_1,(v,?,c),s_1') \in R_1, c \in \textsc{ls}^2(s_2) & \\ \text{and } (s_2,(v,?,c),s_2') \in R_2 & \text{or} \\ (s_1,(v,?,c),s_1') \in R_1, c \notin \textsc{ls}^2(s_2), \text{ and } s_2 = s_2' & \text{or} \\ c \notin \textsc{ls}^1(s_1), s_1 = s_1', \text{ and } (s_2,(v,?,c),s_2') \in R_2 & \end{array} \right\} \cup$$

$$\left\{ ((s_1,s_2),(v,\gamma,\star),(s_1',s_2')) \;\middle|\; \begin{array}{ll} \gamma \in \{!,?\}, (s_1,(v,\gamma,\star),s_1') \in R_1, s_2 = s_2' \text{ and} & \\ \forall s_2'' . (s_2,(v,?,\star),s_2'') \notin R_2 & \text{or} \\ \gamma \in \{!,?\}, s_1 = s_1', \forall s_1'' . (s_1,(v,?,\star),s_1'') \notin R_1 \text{ and} & \\ (s_2,(v,\gamma,\star),s_2') \in R_2, & \end{array} \right\}$$

- $L(s_1,s_2) = (L_1(s_1), L_2(s_2))$
- $\textsc{ls}(s_1,s_2) = \textsc{ls}^1(s_1) \cup \textsc{ls}^2(s_2)$

The transition relation R of the composition defines two modes of interactions, namely multicast and broadcast. In both interaction modes, the composition $\mathscr{T}$ sends a message $(v,!,c)$ on channel c (i.e., $((s_1,s_2),(v,!,c),(s_1',s_2')) \in R$) if either $\mathscr{T}_1$ or $\mathscr{T}_2$ is able to generate this message, i.e, $(s_1,(v,!,c),s_1') \in R_1$ or $(s_2,(v,!,c),s_2') \in R_2$.

Consider the case of a multicast channel. A multicast is blocking. Thus, a multicast message is sent if either it is received or the channel it is sent on is not listened to. Suppose that a message originates from $\mathscr{T}_1$, i.e., $(s_1,(v,!,c),s_1') \in R_1$. Then, $\mathscr{T}_2$ must be able to either receive the message or, in the case that $\mathscr{T}_2$ does not listen to the channel, discard it. CTS $\mathscr{T}_2$ receives if $(s_2,(v,?,c),s_2') \in R_2$. It discards if $c \notin \textsc{ls}^2(s_2)$ and $s_2 = s_2'$. The case of $\mathscr{T}_2$ sending is dual. Note that $\mathscr{T}_2$ might be a composition of other CTS(s), say $\mathscr{T}_2 = \mathscr{T}_3 \| \mathscr{T}_4$. In this case, $\mathscr{T}_2$ listens to channel c if at least one of $\mathscr{T}_3$ or $\mathscr{T}_4$ is listening. That is, it could be that either $c \in (\textsc{ls}(s_3) \cap \textsc{ls}(s_4))$, $c \in (\textsc{ls}(s_2) \backslash \textsc{ls}(s_3))$, or $c \in (\textsc{ls}(s_2) \backslash \textsc{ls}(s_4))$. In the first case, both must receive the message. In the latter cases, the listener receives and the non-listener discards. Accordingly, when a message is sent by one system, it is propagated to all other connected systems in a joint transition. A multicast is indeed blocking because a connected system cannot discard an incoming message on a channel it is listening to. More precisely, a joint transition $((s_1,s_2),(v,!,c),(s_1',s_2'))$ where $c \in \textsc{ls}(s_2)$ requires that $(s_2,(v,?,c),s_2')$ is supplied. In other words, message sending is blocked until all connected receivers are ready to participate in the interaction.

Consider now a broadcast. A broadcast is non-blocking. Thus, a broadcast message is either received or discarded. Suppose that a message originates from $\mathscr{T}_1$, i.e., $(s_1, (\upsilon, !, \star), s_1') \in R_1$. If $\mathscr{T}_2$ is receiving, i.e., $(s_2, (\upsilon, ?, \star), s_2') \in R_2$ the message is sent. However, by definition, we have that $\star \in \text{LS}(s)$ for every s in a CTS. Namely, a system may not disconnect the broadcast channel $\star$. For this reason, the last part of the transition relation R defines a special case for handling (non-blocking) broadcast. Accordingly, a joint transition $((s_1, s_2), (\upsilon, \gamma, \star), (s_1', s_2')) \in R$ where $\gamma \in \{!, ?\}$ is always possible and may not be blocked by any receiver. In fact, if $(\gamma = !)$ and $(s_1, (\upsilon, !, \star), s_1') \in R_1$ then the joint transition is possible whether $(s_2, (\upsilon, ?, \star), s_2') \in R_2$ or not. In other words, a broadcast can happen even if there are no receivers. Furthermore, if $(\gamma = ?)$ and $(s_1, (\upsilon, ?, \star), s_1') \in R_1$ then also the joint transition is possible regardless of the other participants. In other words, a broadcast is received only by interested participants.

4.2 Properties of parallel composition

Our parallel composition is commutative and associative. Furthermore, it supports non-blocking broadcast and blocking multicast semantics as stated in the following lemmas:

Lemma 1 (Commutativity and Associativity) *Given two CTS $\mathscr{T}_1$ and $\mathscr{T}_2$ we have that:*

- $\|$ *is commutative:* $\mathscr{T}_1 \| \mathscr{T}_2 = \mathscr{T}_2 \| \mathscr{T}_1$;
- $\|$ *is associative:* $(\mathscr{T}_1 \| \mathscr{T}_2) \| \mathscr{T}_3 = \mathscr{T}_1 \| (\mathscr{T}_2 \| \mathscr{T}_3)$.

Note that Lemma 1 is crucial to ensure that our parallel compostion operator is a commutative monoid, as otherwise it would not represent the right behaviour of interacting programs.

Lemma 2 (Non-blocking Broadcast) *Given a CTS $\mathscr{T}_1$ and for every other CTS $\mathscr{T}$, we have that for every reachable state (s_1, s) of $\mathscr{T}_1 \| \mathscr{T}$ the following holds.*

$$(s_1, (\upsilon, !, \star), s_1') \in R_1 \textit{ implies } ((s_1, s), (\upsilon, !, \star), (s_1', s')) \in R_{\mathscr{T}_1 \| \mathscr{T}}$$

Lemma 3 (Blocking Multicast) *Given a CTS $\mathscr{T}_1$ and a multicast channel $c \in C \backslash \{\star\}$ such that $(s_1, (\upsilon, !, c), s_1') \in R_1$, then for every other CTS $\mathscr{T}$ we have that in every reachable state (s_1, s) of $\mathscr{T}_1 \| \mathscr{T}$ the following holds.*

$$((s_1, s), (\upsilon, !, c), (s_1', s')) \in R_{\mathscr{T}_1 \| \mathscr{T}} \textit{ iff } \begin{pmatrix} c \in \text{LS}(s) \textit{ and } (s, (\upsilon, ?, c), s') \in R \\ \textit{or} \qquad\qquad\qquad\qquad c \notin \text{LS}(s) \end{pmatrix}$$

The proofs of these lemmas are omitted here and included in the Appendix.

5 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. We give semantics to individual agents in terms of channelled transition systems (CTS). Therefore, we use the parallel composition operator in Definition 3 to compose the individual behaviour of the different agents to generate a global (or a system) one.

While the CTS semantics makes it clear what are the capabilities of individual agents and their interaction, it may not be the most convenient in order to mechanically analyse large systems comprised of multiple agents. Thus, we provide a symbolic semantics at system level using discrete systems. This second semantics enables efficient analysis by representing *closed* systems through the usage of BDDs or representing computations through Boolean formulas. We show that the two semantics (when restricted to *closed* systems) coincide.

The efficient analysis of *open* ReCiPe systems is left as future work.

We assume that a set of K agents agree on a set of common variables cv, a set of data variables d, and a set of channels ch containing the broadcast channel $\star$. As explained, common variables are variables that are owned (separately) by all agents. The values of these variables may be different in different agents. The common variables are used in order to have a common language to express properties that are interpretable on all agents (as either true or false).

Definition 4 *(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 $\mathsf{Dom}(v)$ is the domain of v, then s^i is an element in $\prod_{v\in V_i} \mathsf{Dom}(v)$. 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 renaming 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 $g_i^r(V_i, \star) = true$, 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.
- $\mathcal{T}_i^r(V_i, V_i', \text{d}, \text{ch})$ is an assertion describing the receive transition relation. We assume that agents are broadcast input-enabled, i.e., $\forall v, \mathbf{d}\ \exists v'\ \textbf{s.t.}\ \mathcal{T}_i^r(v, v', \mathbf{d}, \star)$.
  In examples, we use $\text{keep}(X)$ to denote that the variables X are not changed by a transition (either send or receive). More precisely, $\text{keep}(X)$ is equivalent to the following assertion $\bigwedge_{x\in X} x = x'$.
- 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 (that we shall call *an observation*) is defined by the channel it is sent on (ch), the data it carries ($\mathbf{d}$), the sender identity (i), and the assertion describing the possible local assignments to common variables of receivers (π). Formally:

Definition 5 *(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 5 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 $s^i \in \prod_{v\in V_i} \mathsf{Dom}(v)$ 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 $c \models \pi$. We also use $\pi(f_i^{-1}(s_i))$ to denote the assignment of $v \in \text{cv}$ by $s_i(f_i(v))$ in π.

The semantics of an agent A_i is the CTS $\mathscr{T}(A_i)$ defined as follows.

Definition 6 *(Agent Semantics)* Given an agent A_i we define $\mathscr{T}(A_i) = \langle C, \Sigma, \Upsilon, S, S_0, R, L, \text{ls}\rangle$, where the components of $\mathscr{T}(A_i)$ are as follows.

– $C = \text{ch}$

- $\Sigma = \prod_{v \in V_i} \mathsf{Dom}(v)$, i.e., the set of states of A_i
- $\Upsilon = \Upsilon^+ \times \{!, ?\} \times \text{CH}$ and $\Upsilon^+ = 2^{\text{D}} \times K \times 2^{2^{\text{CV}}}$, where the set K ranges over the identities of the senders.
- $S = \Sigma$
- $S_0 = \{s \in S \mid \theta_i(s)\}$
- $R =$

$$\begin{array}{ll}\{(s, (\mathbf{d}, i, \pi, !, c), s') \mid \mathcal{T}_i^s(s, s', \mathbf{d}, c) \text{ and } \pi = g_i^s(s, c, \mathbf{d})\} & \cup \\ \{(s, (\mathbf{d}, i', \pi, ?, c), s') \mid \mathcal{T}_i^r(s, s', \mathbf{d}, c), i' \neq i, c \in \text{LS}(s), \text{ and } \pi(f_i^{-1}(s_i))\} & \end{array}$$

- $L(s) = s$
- $\text{LS}(s) = \{c \in C \mid g_i^r(s, c)\}$

Generally, the semantics of an agent is defined as an *open* CTS $\mathscr{T}(A_i)$. The transition alphabet Υ of $\mathscr{T}(A_i)$ is the set of observations (as in Definition 5) that are additionally labelled with either send (!) or receive (?) symbols, corresponding to send and receive transitions. Furthermore, in every state s, an agent is listening to the set of channels in $\text{LS}(s)$. Namely, all channels that satisfy the agent's receive guard g_i^r in state s. We give further intuition for the definition of the transition relation R.

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

- Case ($\gamma =!$): Agent i is a sender and 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 $true$.
- Case ($\gamma =?$): Agent i is a receiver (potentially) accepting a message from another agent i' on channel c and data $\mathbf{d}$ with a send guard π such that $c \in \text{LS}(s)$, $\pi(f_i^{-1}(s_i))$, and $\mathcal{T}_i^r(s_i, s_i', \mathbf{d}, ch)$. Note that the condition $i' \neq i$ is required to ensure that the message is sent by another agent.

Intuitively, if the agent i is the sender, it determines the predicate π (by assigning s_i, $\mathbf{d}$, and ch in g_i^s) and i's send transition $\mathcal{T}_i^s$ is satisfied by assigning s_i, s_i', $\mathbf{d}$, and ch to it. That is, upon sending the message with $\mathbf{d}$ on channel ch the sender changes the state from s_i to s_i'. If the agent i is the receiver, it must satisfy the condition on receivers π (when translated to its local copies of the common variables), it must be connected to ch (according to g_i^r), and it must have a valid receive transition $\mathcal{T}_i^r$ when reading the data sent in $\mathbf{d}$ on channel ch.

Note that the semantics of an individual agent is totally decoupled from the semantics of how agents interact. Thus, different interaction modes (or parallel composition operators) can be adopted without affecting the semantics of individual agents. In our case, we have chosen to implement broadcast as a non-blocking send and non-blocking receive and a multicast as a blocking send and receive. However, if one chooses to do so, other composition operators could be defined. For example, a point-to-point composition would allow only two agents to communicate over a channel and would not allow send without receive.

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

Let $S_i = \Pi_{v \in V_i} \mathsf{Dom}(v)$ be the set of states of agent i and $S = \Pi_i S_i$ be the set of states of the whole system. Given an assignment $s \in S$ we denote by s_i the projection of s on S_i.

Definition 7 *(ReCiPe System as a CTS)* Given a set $\{A_i\}_i$ of agents, a system is defined as the parallel composition of the CTS representations of all A_i, i.e., a system is a CTS of the form $\mathscr{T} = \|_{i \in I} \mathscr{T}(A_i)$.

A triplet (s, υ, s'), where $\upsilon = (\mathbf{d}, i, \pi, !, c)$ is in the transition relation of the composed system $\mathcal{T}$ (according to Definition 3), if the following conditions hold:

- There exists a sender i such that $(s_i, (\mathbf{d}, i, \pi, !, c), s_i') \in R_i$. By Definition 6, we know that $(s_i, (\mathbf{d}, i, \pi, !, c), s_i') \in R_i$ iff $\pi = g_i^s(s_i, ch, \mathbf{d})$, 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:
  1. $c \in \text{LS}^{i'}(s_{i'})$ and $(s_{i'}, (\mathbf{d}, i, \pi, ?, c), s_{i'}') \in R_{i'}$. By Definition 6, we know that $c \in \text{LS}^{i'}(s_{i'})$ and $(s_{i'}, (\mathbf{d}, i, \pi, ?, c), s_{i'}') \in R_{i'}$ iff $g_{i'}^r(s_{i'}, c)$, $\pi(f_{i'}^{-1}(s_{i'}))$, and $\mathcal{T}_{i'}^r(s_{i'}, s_{i'}', \mathbf{d}, ch)$, all evaluate to $true$;
  2. $c \notin \text{LS}^{i'}(s_{i'})$ and $s_{i'} = s_{i'}'$. By Definition 6 this is equivalent to $\neg g_{i'}^r(s_{i'}, ch)$; or
  3. $ch = \star$ and $s_{i'} = s_{i'}'$. By Definition 6 this is equivalent to $\neg\pi(f_{i'}^{-1}(s_{i'}))$.

Intuitively, a message $(\mathbf{d}, i, \pi, !, c)$ 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', $\mathbf{d}$, and ch 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 $\mathbf{d}$ on ch, (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.

In order to facilitate symbolic analysis, we now define a symbolic version of $\|_{k\in K}\mathcal{T}(A_k)$, under closed world assumption. That is, we only focus on messages that originate from the system under consideration. In fact, from an external observer point of view, only message sending is observable while reception cannot be observed. This notion of observability is the norm in existing theories on group communication [26,51]. Thus, we consider the paths of $\|_{k\in K}\mathcal{T}(A_k)$ that are of the form $\sigma = s_0, a_0, s_1, a_1, \ldots$ such that a_j is of the form $(\mathbf{d}, i, \pi, !, c)$, $s_0 \in S_0$ and for every $j \geq 0$ we have $(s_j, a_j, s_{j+1}) \in R$. Note that $(\mathbf{d}, i, \pi, !, c)$ coincides with our definition of an observation m.

Thus, let Υ be the set of possible observations in $\|_{k\in K}\mathcal{T}(A_k)$. That is, let CH be the set of channels, $\mathcal{D}$ the product of the domains of variables in D, K the set of agent identities, and $\Pi(\text{CV})$ the set of predicates over CV then $\Upsilon \subseteq \text{CH} \times \mathcal{D} \times K \times \Pi(\text{CV})$. 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 V_i (a state of the agent with identity i), CH, and D.

Furthermore, we extend the format of the allowed transitions in the classical definition of a discrete system from assertions over an extended set of variables to assertions that allow quantification.

Definition 8 *(Discrete System)* Given a set $\{A_i\}_i$ of agents, a system is defined as follows: $S = \langle \mathcal{V}, \rho, \theta \rangle$, where $\mathcal{V} = \biguplus_i V_i$, a state of the system is in $\prod_i \prod_{v\in V_i} \mathsf{Dom}(v)$ and the initial assertion $\theta = \bigwedge_i \theta_i$. The transition relation of the system is characterised as follows:

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

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 $\exists \mathrm{CV}.f_j$) 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 9 *(System trace)* A system trace is an infinite sequence $\rho = s_0 m_0, s_1 m_1, \ldots$ 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:

$$(s_t,\ s_{t+1}) \models \mathcal{T}_k^s(s_t^k, s_{t+1}^k, \mathbf{d}_t, ch_t) \wedge \bigwedge_{j \neq k} \left(\exists \mathrm{CV}.f_j \wedge \begin{pmatrix} g_j^r(s_t^j, ch_t) \wedge \mathcal{T}_j^r(s_t^j, s_{t+1}^j, \mathbf{d}_t, ch_t) \wedge \pi_t \\ \vee \hfill \neg g_j^r(s_t^j, ch_t) \wedge s_t^j = s_{t+1}^j \\ \vee \hfill ch_t = \star \wedge \neg \pi_t \wedge s_t^j = s_{t+1}^j \end{pmatrix} \right)$$

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 theorem states a full abstraction property [47], namely that the CTS semantics of systems and their discrete counterpart define the same transition relation, under closed world assumption. That is, by considering the messages originating from the system under consideration as the only observations.

Theorem 3 (Full abstraction) *Given a set of* ReCiPe *agents $\{A_i\}_i$, their discrete system representation, defined as $S = \langle \mathcal{V},\ \rho,\ \theta \rangle$, is semantically equivalent to the parallel composition of their CTS representation, defined as $\mathcal{T} = \|_i \mathcal{T}(A_i)$, under closed world assumption. More precisely,*

– *for every assignment s to system variables $\mathcal{V}$, it follows that: $\theta(s)$ iff $s \in S_0$;*

– *for all assignments s and s' to variables in $\mathscr{V}$ and respectively in $\mathscr{V}'$ it follows that: $\rho(s, s')$ iff there exist assignment to data variables* $\mathbf{d}$*, a communication channel ch, and an agent i such that $(s, (\mathbf{d}, i, \pi, !, ch), s') \in R_{\mathscr{T}}$.*

Proof We prove each statement separately.

– For k agents in the symbolic representation, θ characterises the set of system states $S' \subseteq \Pi_i S_i$ that satisfy the initial conditions of all agents, i.e., $\{s \mid s = (s_0, s_1, \ldots, s_k)$ and$s \models \bigwedge_i \theta_i\}$. Note that $(s_0, s_1, \ldots, s_k) \models \bigwedge_i \theta_i$ iff $s_0 \models \theta_0 \wedge s_1 \models \theta_1 \wedge \cdots \wedge s_k \models \theta_k$. By Definitions 3 and 6 this is exactly the set of initial states S_0 in $\mathscr{T} = \|_i \mathscr{T}(A_i)$;
– By Definition 8, we have that $\rho(s, s')$ evaluates to true if there exists a valuation $\mathbf{d}$ to D and a channel ch in CH such that both of the following hold:
  – There exists an agent i such that the send transition $\mathcal{T}_i^s$ is satisfied by assigning to current *local* state s_i, next local state s_i' (i.e., the projection of the system states s and s' on agent i), the valuation $\mathbf{d}$, and the communication channel ch. According to the enumerative semantics in Definition 6, agent i has an individual send transition given the current *local* state s_i, next local state s_i', valuation $\mathbf{d}$ to data variables, and channel ch. Namely, agent i has a send transition $(s_i, (\mathbf{d}, i, \pi, !, ch), s_i') \in R_i$ such 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:
    1. it is connected to channel ch (i.e., $g_{i'}^r(s_{i'}, ch)$ holds), satisfies the send predicate (i.e., $\pi(f_{i'}^{-1}(s_{i'}))$ holds), and participates in the interaction (i.e., $\mathcal{T}_{i'}^r(s_{i'}, s_{i'}', \mathbf{d}, ch)$ holds). By Definition 6, we know that agent i' has an individual receive transition $(s_{i'}, (\mathbf{d}, i, \pi, ?, ch), s_{i'}') \in R_{i'}$ where $ch \in \text{LS}^{i'}(s_{i'})$;
    2. it is not connected to channel ch (i.e., $\neg g_{i'}^r(s_{i'}, ch)$) and $s_{i'} = s_{i'}'$. By Definition 6, agent i' does not have a receive transition for this message. In other words, since $ch \notin \text{LS}^{i'}(s_{i'})$ then agent i' cannot observe this transmission ;
    3. or the message is sent on a broadcast channel ($ch = \star$), where agent i' does not satisfy the sender predicate (i.e., $\neg\pi(f_{i'}^{-1}(s_{i'}))$) and $s_{i'} = s_{i'}'$. By Definition 6 this is equivalent to ignoring this message by not implementing a receive transition.

  So far, we have shown that every individual (send/receive transition) in the symbolic model has a corresponding one in the enumerative semantics of individual agents. We need to show that the composition of these individual transitions according to ρ in the symbolic model has exactly the same semantics of the parallel composition in Definition 3. That is, $\rho(s, s')$ iff for the identified $\mathbf{d}$, i, ch and π we have $(s, (\mathbf{d}, i, \pi, !, ch), s') \in R_{\mathscr{T}}$, given the assignments s and s' to variables in $\mathscr{V}$ and respectively in $\mathscr{V}'$.

  The existential quantification on sender transitions in ρ (i.e., $\bigvee_k \mathcal{T}_k^s(V_k, V_k', \text{D}, ch)$) implies that the order of the composition is immaterial, namely any two systems states $(s_0, s_1, \ldots, s_k)$ and $(s_1, s_0, \ldots, s_k)$ that only differ in the order of individual agent' states are semantically equivalent. By Lemma 1, we have that parallel composition is commutative, and thus the order is immaterial under the enumerative system semantics as well. If $\rho(s, s')$ is due to a message exchange on the broadcast channel $\star$ then the non-blocking semantics of the broadcast is preserved by the transition relation of the CTS composition as stated in Lemma 2. Moreover, if $\rho(s, s')$ is due to a message exchange on a multicast channel c then the blocking semantics of the multicast is preserved by the transition relation of the CTS composition as stated in Lemma 3. Lastly, the universal quantification on all possible receivers in ρ (i.e., $\bigwedge_{j \neq k}$) follows by the CTS semantics of parallel composition in Definition 3, where a receive transition can be received jointly by different agents, and by the commutativity and associativity of parallel composition (Lemma 1), where the scope of a send transition can be extended to cover all possible receivers.

The other direction of the proof follows in similar manners.

□

The following is a corollary of Theorem 3 to relate the traces arising from Definition 9 to that of Definition 7.

Corollary 1 (Trace equivalence) *The traces of a symbolic system composed of a set of agents $\{A_i\}_i$ are the paths of the induced CTS.*

6 Reconfigurable manufacturing scenario

We complete the details of the RMS example, informally described in Sect. 2. Many aspects of the example are kept simple on purpose to aid the presentation.

The system, in our scenario, consists of an assembly product line agent (`line`) and several types of task-driven robots. We describe the behaviour of the product line and only robots of type-1 (`t1`) as these are sufficient for exposing all features of ReCiPe.

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 (e.g., `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 $\{\mathsf{A}, \ldots\}$. For simplicity, we only use the multicast channel A and fix it to the line agent. The set of data variables includes MSG, NO, and LNK, indicating the type of the message, a number (of robots per type), and a name of a channel respectively.

We note that when a data variable is not important for some message it is omitted from the description of the message.

We start with the description of the line agent `line`. We give a high-level overview of the protocol applied by the line agent using the state machine in Fig. 1. The states capture a partial evaluation of the state variables of the agent. In this case, the value of the state variable st. Transitions labels represent guarded commands. We use the format "$\langle \boldsymbol{\Phi} \rangle$ **d** !/? $\mathsf{ch}[\mathsf{v}_1' = \mathsf{a}_1; \ldots \mathsf{v}_n' = \mathsf{a}_n]$" to denote a guarded command cmd. Namely, the predicate $\boldsymbol{\Phi}$ is a condition on the current assignment to local variables of an agent (and for receive transitions also on data variables that appear in the message). We freely use **d** to refer to an assignment to data variables. Usually, we write directly only the value of the MSG variable to avoid cluttering. Sometimes, we add the values of additional data variables. Each guarded command is labelled with a role (! for send and ? for receive transitions); also with a channel name ch and a new assignment to local variables $[\mathsf{v}_1' = \mathsf{a}_1; \ldots \mathsf{v}_n' = \mathsf{a}_{n]}$ to represent the side effects of the interaction. For the line agent, the protocol consists of starting from the pending

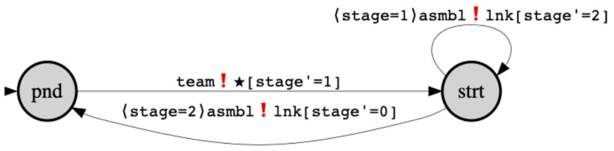
(stage=1)asmbl ! lnk[stage'=2]
pnd
team ! ★[stage'=1]
(stage=2)asmbl ! lnk[stage'=0]
strt

Fig. 1 Product line agent

state and sending a team formation broadcast. This is followed by sending of an assembly multicast on the channel stored in local variable lnk and updating the stage to 2. Finally, an additional assembly multicast on the same channel resets the process. We include below the full description with the guards and predicates. Each transition in the state machine corresponds to a disjunct in either the send or the receive transition predicate below. Variables that are not assigned in a transition are kept unchanged in the predicate. The send and receive guards of the agent are only partially captured in the state machine.

We now turn to the formal description of the line agent, starting with its set of variables. In addition to copies of common variables (e.g., $f_l(\text{@type}) = \mathsf{ltype}$), the line agent has the following state variables: st is a state ranging over {pnd, 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 range over the different stages of production.

The initial condition θ_l of a line agent is defined as follows:

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

Thus, starting from the pending state, the line agent has a task of assembling one of two products, and uses a multicast channel A to coordinate the assembly team. If there are multiple product lines, then each is initialised with a dedicated channel.

The send guard of the LINE agent is of the following form:

$$\begin{aligned} g_l^s : ch{=}\star \wedge \neg\text{@asgn} \wedge (\mathsf{prd}{=}\texttt{1}{\rightarrow}(\text{@type}{=}\texttt{t1} \vee \text{@type}{=}\texttt{t2}))\wedge \\ (\mathsf{prd}{=}\texttt{2}{\rightarrow}(\text{@type}{=}\texttt{t1} \vee \text{@type}{=}\texttt{t3})) \vee ch{=}\mathsf{lnk} \wedge \text{@rdy} = \mathsf{stage} \end{aligned}$$

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 of types t1 and t2 and if the identity of the product is 2, then the required agents are of types 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., $\text{@rdy} = \mathsf{stage}$. The receive guard of LINE is $ch = \star$, i.e., it is only connected to channel $\star$.

We may now proceed by explaining ReCiPe's send and receive transition relations of the line agent in light of the state machine in Fig. 1. The send transition relation of LINE is of the following form:

$$\mathcal{T}_l^s : \text{KEEP}(\mathsf{lnk}, \mathsf{prd}, \mathsf{ltype}, \mathsf{lasgn}, \mathsf{lrdy})\wedge \\ \left(\begin{array}{r} \mathsf{st} = \texttt{pnd} \wedge \mathbf{d}(\text{MSG} \mapsto \texttt{team}; \text{NO} \mapsto \texttt{2}; \text{LNK} \mapsto \mathsf{lnk}) \\ \wedge\, \mathsf{stage}' = \texttt{1} \wedge \mathsf{st}' = \texttt{strt} \wedge ch = \star \\ \vee\, \mathsf{st} = \texttt{strt} \wedge \mathbf{d}(\text{MSG} \mapsto \texttt{asmbl}) \wedge \mathsf{stage} = \texttt{1}\wedge \\ \wedge\, \mathsf{st}' = \texttt{strt} \wedge \mathsf{stage}' = \texttt{2} \wedge ch = \mathsf{lnk} \\ \\ \vee\, \mathsf{st} = \texttt{strt} \wedge \mathbf{d}(\text{MSG} \mapsto \texttt{asmbl}) \wedge \mathsf{st}' = \texttt{pnd} \\ \wedge\, \mathsf{stage} = \texttt{2} \wedge \mathsf{stage}' = \texttt{0} \wedge ch = \mathsf{lnk} \end{array}\right)$$

The LINE agent starts in the pending state (see θ_l). It broadcasts a request ($\mathbf{d}(\text{MSG} \mapsto \texttt{team})$) for two robots ($\mathbf{d}(\text{NO} \mapsto \texttt{2})$) per required type asking them to join the team on the multicast channel stored in its lnk variable ($\mathbf{d}(\text{LNK} \mapsto \mathsf{lnk})$). According to the send guard, described before, if the identity of the product to assemble is 1 ($\mathsf{prd} = \texttt{1}$) 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., $\mathbf{d}(\text{LNK}){=}\mathsf{lnk}$) a team on a dedicated link can be formed incrementally at run-time. As a side effects of broadcasting the $\texttt{team}$ message, the line agent moves to the start state ($\mathsf{st}' = \texttt{strt}$) where the first stage of production begins ($\mathsf{stage}' = \texttt{1}$). In the start state, the line agent 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 $\mathsf{@rdy}$ agrees with stage as specified in the send guard). 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 receive transition relation. That is, LINE is not influenced by incoming messages.

We now specify the behaviour of $\texttt{T1}$-robots and show how an autonomous and incremental one-by-one team formation is done anonymously at run-time. As before, we give a high-level overview of the protocol using the state machine in Fig. 2. The team formation starts when unassigned robots are in pending states ($\texttt{pnd}$). From this state they may only receive a team message from a line agent. 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., $\mathsf{lnk}' = \mathbf{d}(\text{LNK})$ etc.) and move to the start state ($\texttt{strt}$). From the start state there are three possible transitions:

- Join - move to state $\texttt{end}$ - a robot joins the team by *broadcasting* a $\texttt{form}$ message to $\texttt{T1}$-robots forwarding the number of still required robots ($\mathbf{d}(\text{NO}) = (\mathsf{no} - \texttt{1})$) and the team link ($\mathbf{d}(\text{LNK}){=}\mathsf{lnk}$). This message is sent only if $\mathsf{no} \geq \texttt{1}$, i.e, at least one robot is needed. From state ($\texttt{end}$) the robot starts its mission.
- Wait - stay in state $\texttt{strt}$ - a robot *receives* a $\texttt{form}$ message from a robot, updating the number of still required robots (i.e., if $\mathbf{d}(\text{NO}) > \texttt{0}$).
- Step back - return to state $\texttt{pnd}$ - a robot *receives* a $\texttt{form}$ message from a robot, informing that no more robots are needed, i.e., $\mathbf{d}(\text{NO}) = \texttt{0}$. The robot disconnects from the team link, i.e., $\mathsf{lnk}' = \bot$. Thus it may not block interaction on the team link.

After joining the team, a robot in state $\texttt{end}$ (i.e., with $\mathsf{step} = \texttt{1}$) starts its mission independently until it finishes ($\mathsf{step}' = \texttt{n} \wedge \mathsf{brdy}' = 1$). We have used $(\ldots)$ to abstract the individual behaviour of the robot in state ($\texttt{end}$). In fact, each local step corresponds to a broadcast message (local) that is hidden from other agents. This will be clarified later in the send guard of the robot which evaluates to false when (local) is enabled.

When all team robots finish their individual tasks (i.e., circled in the self-loop on state $\texttt{end}$ while $\mathsf{brdy} = 1$ until $\mathsf{step} = n$), they become ready to receive an $\texttt{asmbl}$ message on A, to start the next stage of production (i.e, $\mathsf{brdy}' = 2$) while still staying in $\texttt{end}$ state.

From this final stage (i.e., $\mathsf{brdy} = \texttt{2}$) the robots are ready to receive the final $\texttt{asmbl}$ message to finalise the product and subsequently they reset to their initial conditions.

As before, each transition corresponds to a disjunct in the send and receive transition relations, which are fully specified later in this section.

We now turn to the formal description of the robot, starting with its set of variables. In addition to copies of common variables a $\texttt{T1}$-robot has the following variables: st ranges over $\{\texttt{pnd}, \texttt{strt}, \texttt{end}\}$, 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: $f_b(\mathsf{@type}) = \mathsf{btype}$, $f_b(\mathsf{@asgn}) = \mathsf{basgn}$ and $f_b(\mathsf{@rdy}) = \mathsf{brdy}$.

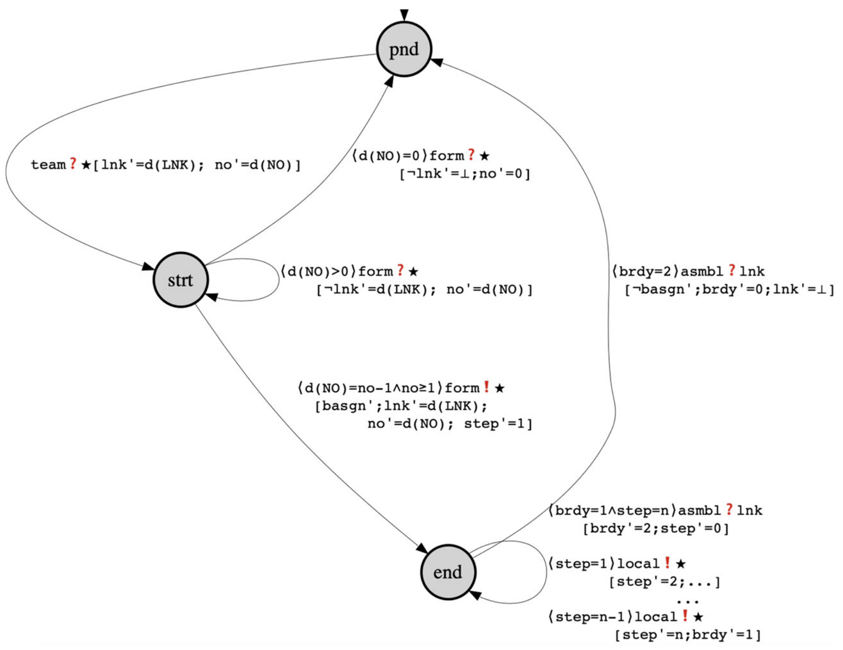
pnd
strt
end
team?★[lnk'=d(LNK); no'=d(NO)]
⟨d(NO)=0⟩form?★ [¬lnk'=⊥;no'=0]
⟨d(NO)>0⟩form?★ [¬lnk'=d(LNK); no'=d(NO)]
⟨brdy=2⟩asmbl?lnk [¬basgn';brdy'=0;lnk'=⊥]
⟨d(NO)=no-1∧no≥1⟩form!★ [basgn';lnk'=d(LNK); no'=d(NO); step'=1]
⟨brdy=1∧step=n⟩asmbl?lnk [brdy'=2;step'=0]
⟨step=1⟩local!★ [step'=2;...]
...
⟨step=n-1⟩local!★ [step'=n;brdy'=1]

Fig. 2 The agent of T1-robot

Initially, a robot is in the pending state and is available for recruitment:

$$\theta_b : (\mathsf{st} = \texttt{pnd}) \wedge (\mathsf{btype} = \texttt{t1}) \wedge \neg\mathsf{basgn} \wedge (\mathsf{lnk} = \bot) \wedge \\ (\mathsf{step} = \mathsf{brdy} = \mathsf{no} = \texttt{0})$$

The send guard of the robot is of the following form:

$$g_b^s : (ch = \star) \wedge \mathbf{d}(\text{MSG} \neq \mathsf{local}) \wedge (@\mathsf{type} = \mathsf{btype}) \wedge \neg @\mathsf{asgn} \vee \\ (ch = \star) \wedge \mathbf{d}(\text{MSG} = \mathsf{local}) \wedge (@\mathsf{asgn} \wedge \neg @\mathsf{asgn})$$

Interestingly, the send guard delimits the scope of the broadcast, depending on the assignment to data variables. Namely, it specifies that a robot either broadcasts to unassigned robots of the same type if the message is not a local one ($\mathbf{d}(\text{MSG} \neq \mathsf{local})$) or otherwise hides the message from all other agents by broadcasting on a false predicate (i.e., the predicate $@\mathsf{asgn} \wedge \neg @\mathsf{asgn}$). Note that such message cannot be received by any agent, and it can be regarded as a local computation. Thus, it becomes very easy to distinguish the individual behaviour of an agent from its interactions with the rest of the system.

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:

$$g_b^r : ch = \star \vee ch = \mathsf{lnk}.$$

Finally, we report the send $\mathcal{T}_b^s$ and receive $\mathcal{T}_b^r$ transition predicates below.

$$\mathcal{T}_b^s : \textsc{keep}(\mathsf{lnk}, \mathsf{btype}) \wedge \left(\begin{array}{r} \mathsf{st} = \texttt{strt} \wedge \mathbf{d}(\text{MSG} \mapsto \texttt{form}; \text{LNK} \mapsto \mathsf{lnk}; \text{NO} \mapsto \mathsf{no} - \texttt{1}) \\ \wedge\ (\mathsf{no} \geq \texttt{1}) \wedge \mathsf{step} = \texttt{0} \wedge \mathsf{step}' = \texttt{1} \wedge \mathsf{st}' = \texttt{end} \\ \wedge \textsc{keep}(\mathsf{brdy}) \wedge \mathsf{basgn}' \wedge (\mathsf{no}' = \texttt{0}) \wedge ch = \star \\ \vee\ \mathsf{st} = \mathsf{st}' = \texttt{end} \wedge \mathbf{d}(\text{MSG} \mapsto \texttt{local}) \wedge ch = \star \wedge \\ \mathsf{step} = \texttt{1} \wedge \mathsf{step}' = \texttt{2} \wedge \textsc{keep}(\mathsf{basgn}, \mathsf{no}, \mathsf{brdy}) \\ \vdots \quad [\textsc{individual behavior}] \\ \vee\ \mathsf{st} = \mathsf{st}' = \texttt{end} \wedge \mathbf{d}(\text{MSG} \mapsto \texttt{local}) \wedge ch = \star \wedge \mathsf{step} = \texttt{n-1} \\ \wedge \mathsf{step}' = \texttt{n} \wedge \mathsf{brdy}' = \texttt{1} \wedge \textsc{keep}(\mathsf{basgn}, \mathsf{no}) \end{array}\right)$$

$$\mathcal{T}_b^r : \textsc{keep}(\mathsf{btype}) \wedge \left(\begin{array}{r} \mathsf{st} = \texttt{pnd} \wedge \mathbf{d}(\text{MSG} \mapsto \texttt{team}) \wedge \mathsf{st}' = \texttt{strt} \wedge ch = \star \wedge \\ \wedge\ \mathsf{lnk}' = \mathbf{d}(\text{LNK}) \wedge \mathsf{no}' = \mathbf{d}(\text{NO}) \wedge \textsc{keep}(\mathsf{basgn}, \mathsf{brdy}, \mathsf{step}) \\ \vee\ \mathsf{st} = \mathsf{st}' = \texttt{strt} \wedge \mathbf{d}(\text{MSG} \mapsto \texttt{form}) \wedge \mathbf{d}(\text{NO}) > \texttt{0} \wedge ch = \star \wedge \\ \textsc{keep}(\mathsf{basgn}, \mathsf{brdy}, \mathsf{step}) \wedge \mathsf{lnk}' = \mathbf{d}(\text{LNK}) \wedge \mathsf{no}' = \mathbf{d}(\text{NO}) \\ \vee\ \mathsf{st} = \texttt{strt} \wedge \mathbf{d}(\text{MSG} \mapsto \texttt{form}; \text{NO} \mapsto \texttt{0}) \wedge ch = \star \wedge \mathsf{st}' = \texttt{pnd} \wedge \\ \wedge \textsc{keep}(\mathsf{basgn}, \mathsf{brdy}, \mathsf{step}) \wedge \mathsf{lnk}' = \bot \wedge \mathsf{no}' = \texttt{0} \\ \vee\ \mathsf{st} = \texttt{end} \wedge \mathbf{d}(\text{MSG} \mapsto \texttt{asmbl}) \wedge \mathsf{brdy} = \texttt{1} \wedge ch = \mathsf{lnk} \wedge \mathsf{step} = \texttt{n} \wedge \\ \wedge\ \textsc{keep}(\mathsf{basgn}, \mathsf{lnk}) \wedge \mathsf{st}' = \texttt{end} \wedge \mathsf{brdy}' = \texttt{2} \wedge \mathsf{step}' = \texttt{0} \\ \vee\ \mathsf{st} = \texttt{end} \wedge \mathbf{d}(\text{MSG} \mapsto \texttt{asmbl}) \wedge \mathsf{brdy} = \texttt{2} \wedge ch = \mathsf{lnk} \\ \wedge\ \mathsf{st}' = \texttt{pnd} \wedge \mathsf{brdy}' = \texttt{0} \wedge \mathsf{lnk}' = \bot \wedge \neg \mathsf{basgn}' \end{array}\right)$$

7 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:

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

We use the classic abbreviations $\rightarrow$, $\leftrightarrow$ and the usual definitions for $true$ and $false$. We also introduce the temporal abbreviations $\mathsf{F}\phi \equiv true\, \mathcal{U}\, \phi$ (*eventually*), $\mathsf{G}\phi \equiv \neg\mathsf{F}\neg\phi$ (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. That is, we push the negation down to atomic propositions. 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., $\overline{\phi_1\, \mathcal{U}\, \phi_2} = \overline{\phi_1}\, \mathcal{R}\, \overline{\phi_2}$, $\overline{\phi_1 \wedge \phi_2} = \overline{\phi_1} \vee \overline{\phi_2}$, $\overline{cv} = \neg cv$, 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:

$$
\begin{array}{ll|ll}
m \models ch' & \textbf{iff } ch = ch' & m \models \neg ch' & \textbf{iff } ch \neq ch' \\
m \models d' & \textbf{iff } \mathbf{d}(d') & m \models \neg d' & \textbf{iff } \neg\mathbf{d}(d') \\
m \models k' & \textbf{iff } k = k' & m \models \neg k' & \textbf{iff } k \neq k'
\end{array}
$$

$$
\begin{array}{lll}
m \models cv & \textbf{iff} & \text{for every assignment } c \models \pi \text{ we have } c \models cv \\
m \models \neg cv & \textbf{iff} & \text{there is an assignment } c \models \pi \text{ such that } c \not\models cv \\
m \models \bullet^{\exists} O & \textbf{iff} & \text{there is an assignment } c \models \pi \text{ such that } (ch, d, k, \{c\}) \models O \\
m \models \bullet^{\forall} O & \textbf{iff} & \text{for every assignment } c \models \pi \text{ it holds that } (ch, d, k, \{c\}) \models O \\
m \models O_1 \vee O_2 & \textbf{iff} & \text{either } m \models O_1 \text{ or } m \models O_2 \\
m \models O_1 \wedge O_2 & \textbf{iff} & m \models O_1 \text{ and } m \models O_2
\end{array}
$$

We only comment on the semantics of the descriptors $\bullet^{\exists} O$ and $\bullet^{\forall} O$ as 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 $\bullet^{\forall} false \wedge ch = \star$ 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 Sect. 6, the descriptor $\bullet^{\exists}(\text{@type} = \texttt{t1}) \wedge \bullet^{\forall}(\text{@type} = \texttt{t1})$ 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 $\bullet^{\exists}(O_1 \vee O_2)$. Furthermore, when cv and respectively $\neg cv$ appear outside the scope of a quantifier ($\bullet^{\forall}$ or $\bullet^{\exists}$), they are semantically equivalent to the descriptors $\bullet^{\forall} cv$ and respectively $\bullet^{\exists} \neg cv$. Thus, we assume that they are written in the latter normal form.

We interpret LTOL formulas over system computations:

Definition 10 *(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 $M = \text{CH} \times 2^{\text{D}} \times K \times 2^{2^{\text{CV}}}$ 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 ith time point of the system computation. Moreover, we denote the suffix of ρ starting with the ith 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:

$$\begin{array}{l}
\rho_{\geq i} \models v \ \textbf{iff}\ s_i \models v \ \text{ and } \ \rho_{\geq i} \models \neg v \ \textbf{iff}\ s_i \not\models v;\\
\rho_{\geq i} \models \phi_2 \vee \phi_2 \ \textbf{iff}\ \rho_{\geq i} \models \phi_1 \ \text{ or } \ \rho_{\geq i} \models \phi_2;\\
\rho_{\geq i} \models \phi_2 \wedge \phi_2 \ \textbf{iff}\ \rho_{\geq i} \models \phi_1 \ \text{ and } \ \rho_{\geq i} \models \phi_2;\\
\rho_{\geq i} \models \phi_1 \,\mathcal{U}\, \phi_2 \ \textbf{iff} \text{ there exists } j \geq i \text{ s.t. } \rho_{\geq j} \models \phi_2 \text{ and,}\\
\qquad\qquad\qquad\qquad\qquad \text{for every } i \leq k < j,\ \rho_{\geq k} \models \phi_1;\\
\rho_{\geq i} \models \phi_1 \,\mathcal{R}\, \phi_2 \ \textbf{iff} \text{ for every } j \geq i \text{ either } \rho_{\geq j} \models \phi_2 \text{ or,}\\
\qquad\qquad\qquad\qquad\qquad \text{there exists } i \leq k < j,\ \rho_{\geq k} \models \phi_1;\\
\rho_{\geq i} \models \langle O \rangle \phi \ \textbf{iff}\ m_i \models O \ \text{ and } \ \rho_{\geq i+1} \models \phi;\\
\rho_{\geq i} \models [O] \phi \ \textbf{iff}\ m_i \models O \ \text{ implies } \ \rho_{\geq i+1} \models \phi.
\end{array}$$

Intuitively, the temporal formula $\langle O \rangle \phi$ 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 $[O]\phi$ 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 LTL.

With observation descriptors we can refer to the intention of agents in the interaction. For example, the following descriptor

$$O := \bullet^{\exists}(@\mathsf{type} = \mathtt{t1}) \wedge \bullet^{\exists}(@\mathsf{type} = \mathtt{t2}) \wedge \bullet^{\forall}(@\mathsf{type} = \mathtt{t1} \vee @\mathsf{type} = \mathtt{t2})$$

specifies that the target of the message is “exactly and only" type-1 and type-2 robots. This descriptor can be used later to specify that whenever the line agent “l" recruits for a product with identity 1, it notifies both type-1 and type-2 robots as follows:

$$\mathsf{G}((\mathsf{prd} = \mathtt{1} \wedge \mathsf{st} = \mathtt{pnd} \wedge \langle l \wedge ch = \star \rangle true) \rightarrow \langle O \rangle true)$$

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” [24,45] can be easily expressed in LTL and can be used to check the formation protocol. Consider the following formulas:

$$\varphi_1 := \langle \mathrm{MSG} = \mathtt{team} \wedge \mathrm{NO} = \mathtt{2} \wedge \bullet^{\exists}(@\mathsf{type} = \mathtt{t1}) \rangle true$$

specifying that a team message is sent to type-1 robots and requires two robots,

$$\varphi_2 := \langle \mathrm{MSG} = \mathtt{form} \wedge \bullet^{\exists}(@\mathsf{type} = \mathtt{t1}) \rangle true$$

specifying that a formation message is sent to type-1 robots, and

$$\varphi_3 := \langle ch = \mathsf{A} \rangle true$$

specifying that a message is sent on channel A.

Now, the template “After φ_1 have exactly two φ_2 until φ_3” specifies 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,

$$O_1(t) := \mathrm{MSG}{=}\mathtt{team} \wedge \bullet^{\exists}(@\mathsf{type}{=}\mathtt{t})$$

specifies that a team message is sent to robots of type t;

$$O_2(k,t) := \text{MSG}{=}\texttt{form} \wedge \neg k \wedge \text{NO}{=}\texttt{0} \wedge \bullet^{\exists}(\neg@\mathsf{asgn} \wedge @\mathsf{type}{=}\texttt{t})$$

specifies that a robot different from k sends a form message specifying that no more robots are needed and this message is sent to unassigned type t robots;

$$O_3(t) := \text{MSG}{=}\texttt{asmbl} \wedge ch = \mathsf{A} \wedge @\mathsf{rdy}{=}\texttt{2} \wedge \bullet^{\exists}(@\mathsf{type}{=}\texttt{t})$$

specifies that 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

$$\begin{array}{ll}\text{(i)} & \varphi_1(t) := (\mathsf{lnk}{\neq}\mathsf{A})\ \mathcal{W}\ \langle O_1(t)\rangle true \\ \text{(ii)} & \varphi_2(k,t) := \mathsf{G}([O_2(k,t) \vee O_3(t)]\varphi_1(t))\end{array} \tag{1}$$

state that: (i) robots are not connected to channel A until they get a team message, inviting them to join a team; (ii) 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.

7.1 The satisfiability and the model checking problems of LTOL

In this section, we improve our early results on satisfiability and model checking of LTOL, presented in the AAMAS version [1] of this article. In that version, we computed an EXPSPACE upper bound for both problems with respect to the set of common variables CV that appear in the observation descriptors and PSPACE upper bound with respect to the rest of the input. This result was not surprising as the semantics of observations requires quantification on the assignments to common variables CV appearing in O. Indeed, the number of assignments to CV is doubly exponential in the size of CV, i.e, the number of assignments is $2^{2^{|\text{CV}|}}$. Both problems require translation to Nondeterministic Büchi Automata (NBW), and a direct translation would incur a double exponential blowup in the size of the automaton with respect to |CV|. Thus, a membership in EXPSPACE with respect to |CV| follows from the membership in NLOGSPACE of the nonemptiness problem for NBW.

In this article, we improved the latter results to PSPACE, matching the lower bound. This is achieved by a novel automaton construction. Namely, we introduce a further dependency between the formula and the alphabet that is read by the automaton. Thus, the automaton does not read concrete messages but it rather partitions messages into sets, according to their effects on the truth values of subformulas of the formula.

Before we proceed with the automaton construction, we fix the sets of system variables $\mathcal{V}$, the communication channels CH, the data variables D, the identities of agents K, and the common variables CV.

Our direct construction in [1] considers a *state-alphabet* $\Sigma = 2^{\mathcal{V}}$ and a *message-alphabet* $M = \text{CH} \times \text{D} \times K \times 2^{2^{\text{CV}}}$. Clearly, the message-alphabet is doubly-exponential in CV and implies that the decision procedures based on M would be in EXPSPACE (with respect to CV). However, M is "too large" for the automaton (c.f., [62]). Thus, we consider a smaller alphabet that is derived from the observation descriptors appearing in the formula. This alphabet is at most exponential in the size of the formula (allowing for PSPACE analysis). To achieve PSPACE analysis, we have to extend the decision procedures to further consider *observation-alphabet satisfiability* and *observation-alphabet model-checking*, as we will see below.

Recall the alphabets Σ and M above and fix an LTOL formula φ. Let $\mathsf{obs}(\varphi)$ be the set of observations appearing "top-level" in the operators $\langle\cdot\rangle$ and $[\cdot]$ in φ. More precisely, $\mathsf{obs}(\varphi)$ is closed under the subformula relation of φ, but is not closed under the subformula relation of O. Consider $\varphi_2(k,t)$ in Eq. 1:

$$\mathsf{obs}(\varphi_2(k,t)) = \{O_2(k,t) \vee O_3(t), O_1(t)\}$$

We denote by $|\mathsf{obs}(\varphi)|$ the size of the set $\mathsf{obs}(\varphi)$. We denote by $|O|$ the length of the observation O and by $|\mathsf{obs}(\varphi)|$ the sum of lengths of observations in φ. Note that $|\mathsf{obs}(\varphi)|$ is bounded by the size of φ. Thus, we may now define an *observation-alphabet* $\mathcal{O} = 2^{\mathsf{obs}(\varphi)}$, that is at most exponential in the size of φ. We will use this alphabet to enable PSPACE analysis.

In our construction, the automaton reads words from the alphabet $(\Sigma \times \mathcal{O})^\omega$ while system computations are derived from the alphabet $(\Sigma \times M)^\omega$.

Intuitively, an automaton word $w \in (\Sigma \times \mathcal{O})^\omega$ and a system computation $\rho \in (\Sigma \times M)^\omega$ agree on the state-alphabet Σ and only differ in their treatment to messages. Formally, given a word $w = (\sigma_0, o_0), (\sigma_1, o_1), \ldots$, and a system computation $\rho = (\sigma_0', m_0), (\sigma_1', m_1), \ldots$. We say that ρ satisfies w if for every $i \geq 0$ we have that $\sigma_i' = \sigma_i$ and for every $O \in \mathsf{obs}(\varphi)$ we have $m_i \models O$ iff $O \in o_i$. Note that $m_i \models O$ follows the semantics of observation descriptors. Thus, a word w defines a language over system computations.

More precisely, for a word $w \in (\Sigma \times \mathcal{O})^\omega$ we denote by $L_\omega(w)$ the set of system computations satisfying w. We say that w is *non empty* if there is some system computation satisfying it, i.e., if $L_\omega(w) \neq \emptyset$. Furthermore, for a letter $o \in \mathcal{O}$, we denote by $\mathscr{M}(o) = \{m \in M \mid \forall O \in \mathsf{obs}(\varphi) \,.\, O \in o \iff m \models O\}$ the set of models of o. That is, all the messages that satisfy all the observations in o and do not satisfy all the observations that are not in o. We say that o is *non empty* if $\mathscr{M}(o) \neq \emptyset$.

Clearly, a word $w = (\sigma_0, o_0), (\sigma_1, o_1), \ldots$ is non empty if and only if for every $i \geq 0$ we have that o_i is non empty.

We show that satisfiability of LTOL can be reduced to finding a word w such that the set of system computations satisfying w is not empty. Similarly, model checking is reduced to building an automaton for $\neg\varphi$ and identifying a word w satisfying $\neg\varphi$ and a computation ρ of the system under study such that ρ satisfies w.

The following theorem states that the set of computations satisfying a given formula are exactly the ones satisfying words accepted by some finite automaton on infinite words.

Theorem 4 *For every* LTOL *formula φ, there is an Alternating Büchi Automaton (ABW) $A_\varphi = \langle Q, \Sigma, \mathcal{O}, \delta_\phi, q_0, F \subseteq Q\rangle$ such that $\bigcup_{w \in L_\omega(A_\varphi)} L_\omega(w)$ is exactly the set of computations satisfying the formula φ.*

Notice that for a given word w, either *all* the computations that satisfy w satisfy φ or *all* the computations that satisfy w do not satisfy φ (i.e., satisfy $\overline{\varphi}$). In the first case w is accepted by A_φ and in the second it is not accepted by A_φ. Thus, the definition of $\mathcal{O}$ is such that words partition the computations to equivalence sets that are uniform with respect to the satisfaction of 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 the state-alphabet $\Sigma = 2^{\mathcal{V}}$ and the observation alphabet $\mathcal{O} = 2^{\mathsf{obs}(\varphi)}$. The set F of accepting states consists of all sub formulas of the form $\phi_1 \mathcal{R} \phi_2$. The transition relation $\delta_\phi : Q \times \Sigma \times \mathcal{O} \to \mathcal{B}^+(Q)$ is defined inductively on the structure of φ, as follows:

- $\delta_\phi(v, \sigma, o) = true$ if $v \in \sigma$ and $false$ otherwise;
- $\delta_\phi(\neg v, \sigma, o) = true$ if $v \notin \sigma$ and $false$ otherwise;
- $\delta_\phi(\phi_1 \vee \phi_2, \sigma, o) = \delta_\phi(\phi_1, \sigma, o) \vee \delta_\phi(\phi_2, \sigma, o)$;
- $\delta_\phi(\phi_1 \wedge \phi_2, \sigma, o) = \delta_\phi(\phi_1, \sigma, o) \wedge \delta_\phi(\phi_2, \sigma, o)$;
- $\delta_\phi(\phi_1 \mathcal{U} \phi_2, \sigma, o) = \delta_\phi(\phi_1, \sigma, o) \wedge \phi_1 \mathcal{U} \phi_2 \vee \delta_\phi(\phi_2, \sigma, o)$;
- $\delta_\phi(\phi_1 \mathcal{R} \phi_2, \sigma, o) = (\delta_\phi(\phi_1, \sigma, o) \vee \phi_1 \mathcal{R} \phi_2) \wedge \delta_\phi(\phi_2, \sigma, o)$;
- $\delta_\phi(\langle O\rangle\phi_1, \sigma, o) = \begin{cases} \phi_1 & O \in o \\ false & O \notin o \end{cases}$;
- $\delta_\phi([O]\phi_1, \sigma, o) = \begin{cases} \phi_1 & O \in o \\ true & O \notin o \end{cases}$.

The proof of correctness of this construction proceeds by induction on the structure of φ. We prove that when A_φ is in state ϕ_1, it accepts exactly all computations that satisfy ϕ_1. The base cases (i.e., state variable propositions) follow from the definition of δ_ϕ while other cases follow from the semantics of φ and the induction hypothesis. The construction ensures that a computation can only satisfy $\phi_1 \mathcal{U} \phi_2$, if it has a suffix satisfying ϕ_2; otherwise A_ϕ will have an infinite path stuck in $\phi_1 \mathcal{U} \phi_2$ which is not accepting. □

Note that, from Theorem 4, 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 $|\delta_\phi|$ is in $\mathcal{O}(|Q|^2.|\Sigma|.|\mathcal{O}|)$, i.e., it is in $|\varphi|^2.2^{\mathcal{O}(|\varphi|)}$. Finally, the size of the alternating automaton $|A_\varphi|$ is in $\mathcal{O}(|Q|.|\delta_\phi|)$, i.e., $|A_\varphi|$ is in $|\varphi|^3.2^{\mathcal{O}(|\varphi|)}$.

By Theorem 4 and Proposition 2, we have that:

Corollary 2 *For every formula φ there is an NBW N_φ with a state-alphabet $\Sigma = 2^{\mathcal{V}}$ and an observation-alphabet $\mathcal{O} = 2^{obs(\varphi)}$ where $N_\varphi = \langle Q, \Sigma, \mathcal{O}, S^0, \delta, F\rangle$ and $\bigcup_{w \in L_\omega(N_\varphi)} L_\omega(w)$ is exactly the set of computations satisfying φ such that:*

- *$|Q|$ is in $2^{\mathcal{O}(|\varphi|)}$ and $|\delta|$ is in $\mathcal{O}(|Q|^2.|\Sigma|.|\mathcal{O}|)$, i.e., $|\delta|$ is in $2^{\mathcal{O}(|\varphi|)}$.*
- *The required space for building the automaton is* NLOG$(|Q|.|\delta|)$*, i.e., it is in $\mathcal{O}(|\varphi|)$*
- *The size of the Büchi automaton is $|Q|.|\delta|$, i.e., $|N|$ is in $2^{\mathcal{O}(|\varphi|)}$.*

Theorem 5 *The satisfiability problem of* LTOL *is* PSPACE-COMPLETE *with respect to $|\varphi|$.*

Proof By Corollary 2, given a formula φ, we can construct an NBW N_φ of size $|Q_n|.|\delta_n|$ that accepts precisely the computations that satisfy φ. Thus, φ is satisfiable iff N_φ is nonempty. In order to prove that the formula is satisfiable we have to show that N_φ accepts a word w such that some computation ρ satisfies w. However, a word w is non empty iff every letter $o \in \mathcal{O}$ appearing in w is non empty. It follows that while testing the non emptiness of N_φ we have to follow only transitions using non empty letters in $\mathcal{O}$. The nonemptiness of an NBW is tested in nondeterministic logarithmic space. However, as N_φ is exponential in $|\varphi|$ we get an algorithm working in space polynomial in $|\varphi|$. The algorithm constructs N_φ on-the-fly. We have to show that the emptiness of letters in $\mathcal{O}$ can be tested in space polynomial in $|\varphi|$. This follows from Proposition 1 below.

The hardness argument can be proved by a reduction from LTL satisfiability [53]. □

Theorem 6 *The model-checking problem of* LTOL *is* PSPACE-COMPLETE *with respect to $|Sys|$ and $|\varphi|$.*

Note that the stated bounds in terms of $|Sys|$ refer to the symbolic representation of the system. The complexity is LOGSPACE in the size of the corresponding CTS $\mathcal{T}(Sys)$, which is anyway exponentially larger.

Proof Given a finite state system $Sys = \langle \mathscr{V},\ \rho,\ \theta \rangle$ and a set of assertions on state variables $\mathscr{V}$, on $\textsc{ch}$, $\textsc{d}$, K, and on $cv_1, \ldots cv_n$. We assume ρ to be total and then we can construct a CTS representation of Sys as follows: $\mathscr{T}(Sys) = \langle \textsc{ch}, \Sigma, M, S, S_0, R, L, \textsc{ls} \rangle$, where the components of $\mathscr{T}(Sys)$ are as follows. $S = \Sigma$ (L is the identity function), and thus S is the set of possible interpretations of the variables in $\mathscr{V}$, i.e., $S = 2^{\mathscr{V}}$. The set of initial states S_0 is the set of states s such that $s \models \theta$, i.e., $S_0 = \{s \models \theta\}$, and $M = \textsc{ch} \times 2^{\textsc{d}} \times K \times 2^{2^{\textsc{cv}}}$. We have that $R(s, m) = \{s' : (s, m, s') \models \rho\}$ and $\emptyset$ otherwise. Furthermore, we consider all states in $\mathscr{T}(Sys)$ to be accepting. The number of states in the transition system $\mathscr{T}(Sys)$ may be exponentially larger than the description of Sys. Notice that although M is doubly exponential in $\textsc{cv}$ the labels of transitions of $\mathscr{T}(Sys)$ are those obtained from $\mathcal{T}_k^s$ for some k. Thus, the number of distinct labels appearing on transitions of $\mathscr{T}(Sys)$ is bounded by $|S| \cdot |\textsc{ch}| \cdot 2^{|\textsc{d}|} \cdot |K|$.

The system Sys satisfies φ iff all the computations of Sys satisfy φ, thus for every computation $\rho \in L_\omega(\mathscr{T}(Sys))$ there exists a word $w \in L_\omega(N_\varphi)$ such that $\rho \models w$. Dually, Sys does not satisfy φ iff for some computation $\rho \in L_\omega(\mathscr{T}(Sys))$ and for some word $w \in L_\omega(N_{\neg\varphi})$ we have $\rho \models w$. This is equivalent to check $L_\omega(\mathscr{T}(Sys)) \cap \bigcup_{w \in L_\omega(N_{\neg\varphi})} L_\omega(w) = \emptyset$. Since our formulas are in positive normal form, $\neg\varphi$ can be obtained from φ by $\overline{\varphi}$. By Corollary 2, we have that $N_{\neg\varphi}$ has $2^{\mathscr{O}(|\varphi|)}$ states and $|N_{\neg\varphi}|$ is in $2^{\mathscr{O}(|\varphi|)}$. Note, however, that the words of $N_{\neg\varphi}$ are in $(\Sigma \times \mathcal{O})^\omega$ while the computations of $\mathscr{T}(Sys)$ are in $(\Sigma \times M)^\omega$. The model-checking problem can be reduced to finding a word w accepted by $A_{\neg\varphi}$ and a computation ρ of $\mathscr{T}(Sys)$ such that $\rho \models w$. Recall that $\rho \models w$ if for every $i \geq 0$ we have that $\sigma_i^\rho = \sigma_i^w$ and for every $O \in \mathsf{obs}(\varphi)$ we have $m_i^\rho \models O$ iff $O \in o_i^w$. This amounts to check the nonemptiness problem of a (modified) intersection of $\mathscr{T}(Sys)$ and $N_{\neg\varphi}$, where the transition (s, m, s') of $\mathscr{T}(Sys)$ can match transitions of $N_{\neg\varphi}$ that read letters (s, o) for $m \models o$. Note that for every $m \in M$ there is a unique $o \in \mathcal{O}$ such that $m \models o$. Thus, we check letter by letter that the word w accepted by $N_{\neg\varphi}$ and the computation ρ produced by $\mathscr{T}(Sys)$ are such that $\rho \models w$. Thus, we only need to show that checking $m \models o$ can be tested in space polynomial in $|\varphi|$. Indeed, This follows from Proposition 2. Since all states in $\mathscr{T}(Sys)$ are accepting, the construction of $N_{\mathscr{T}(Sys),\neg\varphi}$ is the product of $\mathscr{T}(Sys)$ with $N_{\neg\varphi}$ with transitions composed as explained. We have that $N_{\mathscr{T}(Sys),\neg\varphi}$ has $2^{\mathscr{O}(|Sys|+|\varphi|)}$ states. Hence, $|N_{\mathscr{T}(Sys),\neg\varphi}|$ is in $2^{\mathscr{O}(|Sys|+|\varphi|)}$. We have that $N_{\mathscr{T}(Sys),\neg\varphi}$ can be constructed on-the-fly and a membership in $\textsc{pspace}$ with respect to $|Sys|$ and $|\varphi|$, follows from the membership in $\textsc{nlogspace}$ of the nonemptiness problem for NBW. Checking that $Sys \models \varphi$ is in $\mathscr{O}(|\varphi| + |Sys|)$.

The hardness follows from the same hardness results for discrete systems and $\textsc{ltl}$ [53]. □

The following proposition states that given a letter $o \in \mathcal{O}$ we can check whether there exists a message m that satisfies o in $\textsc{np}$ with respect to $|\mathsf{obs}(\varphi)|$. Notice that, in particular, $|\mathsf{obs}(\varphi)|$ should be larger than the number of variables in $\textsc{cv}$ and $\textsc{d}$ that appear in φ, the number of agents in K that are mentioned in φ and the number of channels in $\textsc{ch}$ appearing in φ. Those that do not appear in φ can be removed from the message alphabet M.

Proposition 1 (Observation satisfiability) *Consider a letter $o \in \mathcal{O}$. Emptiness of o is* $\textsc{np-}$ $\textsc{complete}$ *in $|\mathsf{obs}(\varphi)|$.*

Proof Given a letter $o \in \mathcal{O}$ let $o^{\Uparrow}$ be the set of observations in o and the negations of the observations *not* appearing in o. That is, $o^{\Uparrow} = o \cup \{\overline{O} \mid O \in \mathsf{obs}(\varphi) \backslash o\}$. Let $o_\wedge^{\Uparrow} = \bigwedge_{O \in o^{\Uparrow}} O$ be the conjunction of all observations in $o^{\Uparrow}$. Clearly, the Emptiness of a letter $o \in \mathcal{O}$ is equivalent to the satisfiability of $o_\wedge^{\Uparrow}$. Thus, we can restrict our attention to the satisfaction of an observation. Given an observation O let $atom(O)$ denote the set of subformulas of O of the form $\bullet^{\exists} O'$ and $\forall O'$.

We show that satisfaction of O can be solved in NP as follows:

- select a subset S of $atom(O)$;
- select an assignment to D, a channel ch and an agent k;
- for each $\bullet^{\exists} O' \in S$ guess one assignment to CV.

Verify that the choice of S, the assignment to D, the channel ch and the agent k satisfy O. Notice, that the elements of $atom(O)$ are treated as Boolean values in this check: $O'' \in S$ is evaluted as true and $O'' \notin S$ is evaluated as false. For each $\bullet^{\exists} O' \in S$ check that the assignment to CV guessed for $\bullet^{\exists} O'$ fulfills two conditions:

- the assignment to CV together with the assignment to D, the channel ch and the agent k satisfy O'.
- For every $\bullet^{\forall} O'' \in S$ check that the assignment to CV together with the assignment to D, the channel ch and the agent k satisfy O''.

Notice that the sum of sizes of the guessed elements is polynomal in the size of O and the verification can be completed in polynomial time.

Hardness in NP follows from the hardness of Boolean satisfiability. □

The following proposition states that checking if a message $m \in M$ satisfies an observation letter $o \in \mathcal{O}$ can be tested in P^{NP} in $|\mathsf{obs}(\varphi)|$. We consider the case that π is represented as a Boolean formula over CV. This is reasonable as when considering a transition (s, m, s'), where $m = (ch, \mathbf{d}, i, \pi)$, then π can be obtained as such a formula from g_i^s by using the values in s, ch, and $\mathbf{d}$.

Proposition 2 (Observation model-checking) *Consider a letter $o \in \mathcal{O}$ and an observation $m \in M$. Whether $m \models o$ can be tested in P^{NP} in $|\mathsf{obs}(\varphi)|$.*

Proof As in the case of Proposition 1 give a letter $o \in \mathcal{O}$ we consider $o^{\Uparrow}_{\wedge}$. Thus, we restrict our attention to the case of whether m satisfies an observation O.

Let $m = (ch, \mathbf{d}, i, \pi)$. We simplify O by converting every reference to ch, D or i to the constants appearing in m. It follows that we are left with a Boolean combination of $\bullet^{\exists}\cdot$ and $\bullet^{\forall}\cdot$ subformulas, where only variables from CV appear.

For a subformula $\bullet^{\exists} O'$ we can check whether $m \models \bullet^{\exists} O'$ by checking whether $\pi \wedge O'$ is satisfiable. For a subformula $\bullet^{\forall} O'$ we can check whether $m \models \bullet^{\forall} O'$ by checking whether $\pi \rightarrow O'$ is valid. Both checks can be accomplished by an NP oracle.

The problem is NP-hard in $|\text{CV}|$ as $m \models \bullet^{\exists} true$ holds iff π is satisfiable. The problem is co-NP-hard in $|\text{CV}|$ as $m \models \bullet^{\forall} false$ iff π is unsatisfiable. We do not know whether the problem is P^{NP}-complete. □

We note that in the case that m is represented as a set of assignments to CV, we can modify the Boolean value problem [43] to show that $m \models o$ can be evaluated in LOGSPACE.

8 Related works

In this section, we present closely related works with respect to (i) traditional formal modelling of Multi-Agent Systems; (ii) distributed and concurrent computation models; (iii) knowledge and strategic formalisms; (iv) business process modelling; and (v) logics for temporal reasoning.

Traditional formal modelling of MAS As mentioned before, formal modelling is highly influenced by traditional formalisms used for verification, see [8,27]. 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 [9,22,50] 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) [42] and MOCHA [7] are proposed as implementation languages of *Interpreted Systems* (IS) [27] and *Reactive Modules* (RM) [8] 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) [14]: an extension of *Concurrent-Game Structures* (CGS) [9] to enable agents to dynamically hide/reveal their internal states to selected agents. However, vCGS relies on an assumption of [11] 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 [10], Dynamic reactive modules of Alur and Grosu [6], Dynamic reactive modules of Fisher et al. [30], and open MAS [39]. 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.

Distributed and concurrent computation models. There are a plenty of formalisms that were specifically designed to model concurrent computations and distributed systems, (cf. π-calculus [48], Psi-calculus [18,21], *AbC* calculus [2,5], *Channel Systems* [12], etc). These formalisms rely heavily on message-passing and synchronisation, and except for *AbC* and the broadcast version of Psi-calculus [21] they mostly rely on point-to-point communication mechanisms to establish interaction. Clearly, point-to-point communication is not appropriate to model interaction in MAS settings, and a group-based communication is more appropriate (See [2,5] for a detailed comparison).

Furthermore, these formalisms also differ in their degree of support to reconfiguration. On one hand, *Channel Systems* cannot deal with reconfiguration and only support fixed communication structures. On the other hand, while π and Psi-calculi can support 1-by-1 expansion of the scope of interaction, *AbC* supports a general group communication with sophisticated scoping mechanisms. In ReCiPe, we extend these ideas to support awareness capabilities, interaction beyond broadcast, and dynamic construction of groups and private group coalitions. It is worth mentioning that although there is a separation result [25] stating that point-to-point and broadcast communication are incomparable, we can still mimic point-to-point communication in ReCiPe under closed world assumption. For instance, the team formation protocol in Sect. 6 is such example where robots are recruited one-by-one with a non-deterministic selection. This, of course, works because we assume a closed world settings where no other agents may intervene and disturb such protocol. We conjecture that if ReCiPe is extended with mechanisms to allow communicating secret messages, we would be able to encode point-to-point in general.

Knowledge and Strategic formalisms. There is a rich literature on modelling and reasoning about knowledge and strategic behaviour in multi-agent systems. They employ techniques to study knowledge dissemination in distributed settings [54,56] and respectively analysing strategic behaviour [17,20] to study concepts like Nash equilibria, Pareto optimality and evolutionary strategies.

Reasoning in these settings is known to be hard, and restrictions on the structure of the systems and their communication mechanisms are imposed to mitigate such difficulty. For instance, hierarchical environments [56] , broadcast environments [41], public actions [17], public announcements [54] or gossip spread mechanisms [55]. Most of these approaches rely on an assumption of perfect recall to guarantee the decidability of the verification problem, i.e., all agents of the system are aware of all events that have happened so far. In fact, they keep a complete record of all events they have observed so far and they base their judgement on such complete record.

This assumption can be appropriate in specific distributed applications like Blockchains, supply chains where there is a powerful infrastructure that mediates the interaction and makes global information available to everyone easily. However, this is a strong assumption to be made in general distributed settings. For instance, in applications where a group of robots with limited resources communicate in ad-hoc wireless networks. Also, in applications where security is of concern and only specific agents are allowed to have access to a specific information.

To the best of our knowledge, all proposed broadcast settings in the context of knowledge and strategic reasoning imply more than a broadcast in actual communication. That is, in broadcast environments [41] and public actions [17] the communication is totally deterministic and all agents have access to all events that have happened so far and their respective order. This, of course, simplifies the verification problem because it removes all possible sources of information forks (the source of undecidability for some verification tasks). That is, an agent who is not targeted by the communication still knows the events that have happened. In normal broadcast like in ReCiPe, non-targeted agents are unaware of the communication and cannot have access to communications that they did not participate in. In fact, using a perfect recall gives the power to agents to even count how many communication steps have happened so far. This is, clearly, unreasonable in distributed settings because one agent might participate for a finite time in the communication protocol and stays idle most of the time. Allowing agents to have access to that much of information does not come without a cost, it requires memory and, although decidable, it is not surprising that the complexity of verification in such settings is very high.

Other than the fact that all of the mentioned approaches rely on flat or static communication structure, the main difference with respect to our way of dissemination of information in ReCiPe is due to the fact that we are handling a different problem, i.e., a coordination problem. In fact, a coordination problem is a subproblem of general knowledge dissemination. The classic examples of the latter assume a very powerful communication infrastructure and usually ask whether a group of uninformed agents can become totally informed about the state of the whole system after a number of communication rounds. In that settings, an event is made publicly accessible to all agents after being executed, and thus each agent is able to keep track of the state of the system during execution. Although that agent may not know the exact exchange of information, it can still make deductions based on the history of communications. The logic of gossiping [55] gives an excellent characterisation of the different cases of communications in point-to-point settings. From our perspective, we consider the asynchronous case to be the more representative case to coordination in distributed settings as agents only observe the communication they participate in. In a coordination problem, the objective is different as agents can be arbitrarily informed about the state of the system. Some agents might be totally uninformed and stay so because they do not interact with the rest of the systems while other agents might need to know some more information at some point during execution and then forget about it later. Thus, we believe that in a dynamic and reconfigurable communication structure a perfect recall is not required, given that communication can be introduced on need-basis to resolve information forks.

Business process modelling Some approaches from the realm of busniess process modelling such as Artifact-centric systems [16] have some similarities with respect to ReCiPe. An artifact system consists of two main components: *data* which models the local state alongside the view of the agent on the environment; and *lifecycle* which models the behaviour of agents working on the data. Artifact-centric systems define agents' local states as evolving database instances and can be considered as a specialisation of interpreted systems were local states are structured in form of databases. Agents interact among themselves and with an environment comprising all artifacts in the system. That is, all local states of agents are considered as partial views of a holistic environment database. Thus, the environment can be represented as a special agent holding the combined knowledge of all agents in the system. There are two main differences with respect to ReCiPe: (1) there is no notion of reconfiguration of interaction interfaces in artifact systems and all agents have to participate in every interaction even to just stay idle; (2) considering the environment as a special agent composed with the rest of the system restricts its capabilities. That is, the environment can no longer be able to simultaneously trigger multiple events with respect to different agents local views. In other words, this restriction reduces the power of the environment and forces it to interleave and alternate behaviour with respect to other agents. In ReCiPe, the environment can be more powerful because some local variables of agents may abstract incomparable local views of the environment. Changes in the assignments to these variables cannot be restricted, and thus the environment can change them simultaneously resulting in triggering multiple events for the different agents which have to coordinate their executions.

Logics for temporal reasoning 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 [29] and modal μ-calculus [40] 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.

9 Concluding remarks

We introduced a formalism that combines message-passing and shared-memory to facilitate realistic modelling of distributed multi agent systems. 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 [8,30] while interacting externally by message passing as in π-calculus-like formalisms [2,5,48]. 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 provided a novel automata construction that permits satisfiability and model-checking in space polynomial with respect to the size of the formula and the size of the system. This is a major improvement on the early results in [1] that were in EXPSPACE with respect to the number of common variables and PSPACE-complete with respect to the rest of the input.

Future works We plan to provide tool support for ReCiPe, but with a more user-friendly syntax. We would like to provide a light-weight programming-language-like syntax to further simplify modelling.

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 [35]. Although we do not provide explicit knowledge operators, the combination of data exchange, receivers selection, and enabling/disabling of synchronisation based on the evolving states of the different agents, allow them to dynamically deduce information about each other. Furthermore we want to study ReCiPe under dynamic creation of agents while reconfiguring communication. Thanks to the new compositional semantics in terms of CTS, the dynamic creation of agents can now be easily linked to the execution of some blocking transitions. To give an intuition about a (possible) extension consider the following semantic rule:

$$A_1 \xrightarrow{(\upsilon,?,c)} A_1' \| A_1 \tag{2}$$

Here, we use $A \xrightarrow{m} A'$ to denote that in state s agent A may receive a message m and evolves to A' with state s', i.e., $(s, m, s') \in R_A$. Clearly, the semantic rule 2 indicates that the agent replicates itself once a multicast message on c is received as a side-effect of interaction. Thus, if we compose A_1 with some other agent, say A_2, such that $A_2 \xrightarrow{(\upsilon,!,c)} A_2'$, the following transition is derivable by the semantics of parallel composition in Definition 3:

$$\frac{\text{if} \quad A_1 \xrightarrow{(\upsilon,?,c)} A_1' \| A_1 \qquad \text{and} \qquad A_2 \xrightarrow{(\upsilon,!,c)} A_2'}{A_1 \| A_2 \xrightarrow{(\upsilon,!,c)} A_1' \| A_1 \| A_2} \tag{3}$$

Namely, a new replica of A_1 is dynamically created when agents exchanged a specific message. Agent A_1 can be thought of as a server that spawn a new thread to handle concurrent requests from clients.

Finally, we want to target the distributed synthesis problem [28]. Several fragments of the problem have been proven to be decidable, e.g., when the information of agents is arranged hierarchically [19], the number of agents is limited [34], or the actions are made public [15]. We conjecture that the ability to disseminate information and reason about it might prove useful in this setting.

Funding Open access funding provided by University of Gothenburg.

A Proofs for Section 4 (Channelled Transition Systems)

Lemma 1 (Commutativity and Associativity) *Given two CTS $\mathcal{T}_1$ and $\mathcal{T}_2$ we have that:*

– *$\|$ is commutative: $\mathcal{T}_1 \| \mathcal{T}_2 = \mathcal{T}_2 \| \mathcal{T}_1$;*
– *$\|$ is associative: $(\mathcal{T}_1 \| \mathcal{T}_2) \| \mathcal{T}_3 = \mathcal{T}_1 \| (\mathcal{T}_2 \| \mathcal{T}_3)$.*

Note that Lemma 1 is crucial to ensure that our parallel compostion operator is a commutative monoid, as otherwise it would not represent the right behaviour of interacting programs.

Proof We prove each statement separately. In both statement, the proof proceeds by case analysis on the joint transition.

(**$\|$ is commutative**): we consider all possible joint transitions from $\mathcal{T}_1 \| \mathcal{T}_2$ and we show that they have corresponding transitions in $\mathcal{T}_2 \| \mathcal{T}_1$ and vice versa. We only show one direction and the other direction follows in a similar way.

– **Case** $(\upsilon, !, c)$: By Definition 3, we have that $((s_1, s_2), (\upsilon, !, c), (s_1', s_2')) \in R_{12}$ (R_{12} is the transition relation of $\mathcal{T}_1 \| \mathcal{T}_2$) in the following cases:
  1. $(s_1, (\upsilon, !, c), s_1') \in R_1$ and $(s_2, (\upsilon, ?, c), s_2') \in R_2$: It follows that $((s_2, s_1), (\upsilon, !, c), (s_2', s_1')) \in R_{21}$ (R_{21} is the relation of $\mathcal{T}_2 \| \mathcal{T}_1$) is derivable because $(s_2, (\upsilon, ?, c), s_2') \in R_2$ and $(s_1, (\upsilon, !, c), s_1') \in R_1$;
  2. $(s_1, (\upsilon, !, c), s_1') \in R_1$ and $c \notin \mathrm{LS}^2(s_2)$ and $(s_2 = s_2')$: It follows that $((s_2, s_1), (\upsilon, !, c), (s_2', s_1')) \in R_{21}$ is derivable because $c \notin \mathrm{LS}^2(s_2)$ and $(s_2 = s_2')$ and $(s_1, (\upsilon, !, c), s_1') \in R_1$;
  3. $(s_1, (\upsilon, !, \star), s_1') \in R_1$ and $(s_2 = s_2')$ and $\forall s_2''.(s_2, (\upsilon, ?, c), s_2'') \notin R_2$: It follows that $((s_2, s_1), (\upsilon, !, \star), (s_2', s_1')) \in R_{21}$ is derivable because $(s_2 = s_2')$ and $\forall s_2''.(s_2, (\upsilon, ?, c), s_2') \notin R_2$ and $(s_1, (\upsilon, !, c), s_1') \in R_1$;
  4. $(s_2, (\upsilon, !, c), s_2') \in R_2$ and $(s_1, (\upsilon, ?, c), s_1') \in R_1$: This is the symmetric case of case (1);
  5. $(s_2, (\upsilon, !, c), s_2') \in R_2$ and $c \notin \mathrm{LS}^1(s_1)$ and $(s_1 = s_1')$: This is the symmetric case of case (2);
  6. $(s_2, (\upsilon, !, \star), s_2') \in R_2$ and $(s_1 = s_1')$ and $\forall s_1''.(s_1, (\upsilon, ?, c), s_1'') \notin R_1$: This is the symmetric case of case (3).

– **Case** $(\upsilon, ?, c)$**:** By Definition 3, we have that $((s_1, s_2), (\upsilon, ?, c), (s_1', s_2')) \in R_{12}$ in the following cases:
  1. $(s_1, (\upsilon, ?, c), s_1') \in R_1$ and $(s_2, (\upsilon, ?, c), s_2') \in R_2$: It follows that $((s_2, s_1), (\upsilon, ?, c), (s_2', s_1')) \in R_{21}$ is derivable for the same reason;
  2. $(s_1, (\upsilon, ?, c), s_1') \in R_1$ and $c \notin \mathrm{LS}^2(s_2)$ and $(s_2 = s_2')$: It follows that $((s_2, s_1), (\upsilon, ?, c), (s_2', s_1')) \in R_{21}$ is derivable because $c \notin \mathrm{LS}^2(s_2)$ and $(s_2 = s_2')$ and $(s_1, (\upsilon, ?, c), s_1') \in R_1$;
  3. $(s_2, (\upsilon, ?, c), s_2') \in R_2$ and $c \notin \mathrm{LS}^1(s_1)$ and $(s_1 = s_1')$: This is the symmetric case of case (2);
  4. $(s_1, (\upsilon, ?, \star), s_1') \in R_1$ and $(s_2 = s_2')$ and $\forall s_2''.(s_2, (\upsilon, ?, c), s_2'') \notin R_2$: It follows that $((s_2, s_1), (\upsilon, ?, \star), (s_2', s_1')) \in R_{21}$ is derivable because $(s_2 = s_2')$ and $\forall s_2''.(s_2, (\upsilon, ?, c), s_2'') \notin R_2$ and $(s_1, (\upsilon, ?, c), s_1') \in R_1$;
  5. $(s_2, (\upsilon, ?, \star), s_2') \in R_2$ and $(s_1 = s_1')$ and $\forall s_1''.(s_1, (\upsilon, ?, c), s_1'') \notin R_1$: This is the symmetric case of case (3).

(**$\|$ is associative):** we consider all possible joint transitions from $(\mathcal{T}_1 \| \mathcal{T}_2) \| \mathcal{T}_3$ and we show that they have corresponding transitions in $\mathcal{T}_1 \| (\mathcal{T}_2 \| \mathcal{T}_3)$ and vice versa. We only show one direction and the other direction follows in a similar way.

- **Case** $(\upsilon, !, c)$**:** By Definition 3, we have $(((s_1, s_2), s_3), (\upsilon, !, c), ((s_1', s_2'), s_3')) \in R_{(12)3}$ in the following cases:
  1. $((s_1, s_2), (\upsilon, !, c), (s_1', s_2')) \in R_{12}$ and $(s_3, (\upsilon, ?, c), s_3') \in R_3$: As before, there are six cases for $((s_1, s_2), (\upsilon, !, c), (s_1', s_2')) \in R_{12}$, we only consider the case $((s_1, (\upsilon, !, c), s_1') \in R_1$ and $(s_2, (\upsilon, ?, c), s_2') \in R_2)$; and other cases follow similarly. It follows that $((s_1, (s_2, s_3)), (\upsilon, !, c), (s_1', (s_2', s_3')) \in R_{1(23)}$ where $(s_1, (\upsilon, !, c), s_1') \in R_1$ and
  $((s_2, s_3), (\upsilon, ?, c), (s_2', s_3')) \in R_{23}$ such that $(s_2, (\upsilon, ?, c), s_2') \in R_2$ and $(s_3, (\upsilon, ?, c), s_3') \in R_3$ as required;
  2. $((s_1, s_2), (\upsilon, !, c), (s_1', s_2')) \in R_{12}$ and $c \notin \text{LS}^3(s_3)$ and $(s_3 = s_3')$: It follows that $((s_1, (s_2, s_3)), (\upsilon, !, c), (s_1', (s_2', s_3')) \in R_{1(23)}$ where $(s_1, (\upsilon, !, c), s_1') \in R_1$ and $((s_2, s_3), (\upsilon, ?, c), (s_2', s_3')) \in R_{23}$ such that $(s_2, (\upsilon, ?, c), s_2') \in R_2$ and $c \notin \text{LS}^3(s_3)$ and $(s_3 = s_3')$;
  3. $((s_1, s_2), (\upsilon, !, \star), (s_1', s_2')) \in R_{12}$ and $(s_3 = s_3')$ and $\forall s_3''.(s_3, (\upsilon, ?, \star), s_3') \notin R_3$: It follows that $((s_1, (s_2, s_3)), (\upsilon, !, \star), (s_1', (s_2', s_3')) \in R_{1(23)}$ where $(s_1, (\upsilon, !, \star), s_1') \in R_1$ and $((s_2, s_3), (\upsilon, ?, \star), (s_2', s_3')) \in R_{23}$ such that $(s_2, (\upsilon, ?, \star), s_2') \in R_2$ and $(s_3 = s_3')$ and $\forall s_3''.(s_3, (\upsilon, ?, \star), s_3') \notin R_3$;
  4. $((s_1, s_2), (\upsilon, ?, c), (s_1', s_2')) \in R_{12}$ and $(s_3, (\upsilon, !, c), s_3') \in R_3$: This is the symmetric case of case (1);
  5. $(s_3, (\upsilon, !, c), s_3') \in R_3$ and $c \notin \text{LS}^{12}(s_1, s_2)$ and $((s_1, s_2) = (s_1', s_2'))$: This is the symmetric case of case (2);
  6. $(s_3, (\upsilon, !, \star), s_3') \in R_3$ and $(s_1, s_2) = (s_1', s_2')$ and $\forall (s_1'', s_2'')$.
  $((s_1, s_2), (\upsilon, ?, c), (s_1'', s_2'')) \notin R_1$: This is symmetric to case (3).
- **Case** $(\upsilon, ?, c)$**:** it follows similarly by case analysis on Definition 3.

□

Lemma 2 (Non-blocking Broadcast) *Given a CTS $\mathscr{T}_1$ and for every other CTS $\mathscr{T}$, we have that for every reachable state (s_1, s) of $\mathscr{T}_1 \| \mathscr{T}$ the following holds.*

$$(s_1, (\upsilon, !, \star), s_1') \in R_1 \textit{ implies } ((s_1, s), (\upsilon, !, \star), (s_1', s')) \in R_{\mathscr{T}_1 \| \mathscr{T}}$$

Proof By Definition 3, we have only two cases to derive $((s_1, s), (\upsilon, !, \star), (s_1', s')) \in R_{\mathscr{T}_1 \| \mathscr{T}}$ given that $(s_1, (\upsilon, !, \star), s_1') \in R_1$. Note that, by definition, the condition $\star \in \text{LS}^k(s)$ always holds for any agent k and in any state s. We show that when the channel is a broadcast $\star$, the receiver does not play any role in enabling the transmission on the channel. In other words, it is only sufficient to have a sender to enable a broadcast at system level. More precisely, if $(s_1, (\upsilon, !, \star), s_1') \in R_1$ then we have the following:

- $((s_1, s), (\upsilon, !, \star), (s_1', s')) \in R_{\mathscr{T}_1 \| \mathscr{T}}$ because $(s, (\upsilon, ?, \star), s') \in R$; or
- $((s_1, s), (\upsilon, !, \star), (s_1', s')) \in R_{\mathscr{T}_1 \| \mathscr{T}}$ because $(s = s')$ and $\forall s''.(s, (\upsilon, ?, \star), s') \notin R_3$.

Namely, whether there exists a receiver or not, a broadcast can always happen (cannot be blocked). □

Lemma 3 (Blocking Multicast) *Given a CTS $\mathscr{T}_1$ and a multicast channel $c \in C\backslash\{\star\}$ such that $(s_1, (\upsilon, !, c), s_1') \in R_1$, then for every other CTS $\mathscr{T}$ we have that in every reachable state (s_1, s) of $\mathscr{T}_1 \| \mathscr{T}$ the following holds.*

$$((s_1, s), (\upsilon, !, c), (s_1', s')) \in R_{\mathscr{T}_1 \| \mathscr{T}} \textit{ iff } \begin{pmatrix} c \in \text{LS}(s) \textit{ and } (s, (\upsilon, ?, c), s') \in R \\ \textit{or} \qquad\qquad\qquad c \notin \text{LS}(s) \end{pmatrix}$$

Proof We show that it is not sufficient to only have a sender on a multicast channel c to enable a send transition at system level. We require that all listening/connected agents to that channel being able to jointly receive the transmitted message. This also implies that if no one is listening then the transition can happen. By Definition 3, there exist only two cases where $((s_1, s), (\upsilon, !, c), (s_1', s')) \in R_{\mathscr{T}_1 \| \mathscr{T}}$ given that $(s_1, (\upsilon, !, c), s_1') \in R_1$. More precisely, if $(s_1, (\upsilon, !, c), s_1') \in R_1$ then we have the following:

- $((s_1, s), (\upsilon, !, c), (s_1', s')) \in R_{\mathscr{T}_1 \| \mathscr{T}}$ because $(s, (\upsilon, ?, c), s') \in R$. This implies that $c \in \text{LS}(s)$; or
- $((s_1, s), (\upsilon, !, c), (s_1', s')) \in R_{\mathscr{T}_1 \| \mathscr{T}}$ because $(s = s')$ and $c \notin \text{LS}(s)$.

Namely, either all receivers jointly participate or no one is listening, as otherwise the multicast on c is blocked. The other direction of the proof follows similarly. □

References

1. Abd Alrahman, Y., Perelli, G., & Piterman, N. (2020). Reconfigurable interaction for MAS modelling. In A.E.F. Seghrouchni, G. Sukthankar, B. An, N. Yorke-Smith (Eds.) *Proceedings of the 19th international conference on autonomous agents and multiagent systems, AAMAS '20*, Auckland, New Zealand, May 9-13, 2020, (pp. 7–15). International foundation for autonomous agents and multiagent systems.
2. Abd Alrahman, Y., De Nicola, R., & Loreti, M. (2019). A calculus for collective-adaptive systems and its behavioural theory. *Information and Computation*,. https://doi.org/10.1016/j.ic.2019.104457.
3. Aguilera, M.K., Ben-David, N., Calciu, I., Guerraoui, R., Petrank, E., & Toueg, S. (2018). Passing messages while sharing memory. In *Proceedings of the 2018 ACM symposium on principles of distributed computing*, PODC '18, pp. 51–60. ACM, New York, NY, USA. https://doi.org/10.1145/3212734.3212741.
4. Aguilera, M.K., Ben-David, N., Guerraoui, R., Marathe, V., & Zablotchi, I. (2019). The impact of rdma on agreement. In: *Proceedings of the 2019 ACM symposium on principles of distributed computing*, PODC '19, pp. 409–418. ACM, New York, NY, USA. https://doi.org/10.1145/3293611.3331601.
5. Alrahman, Y. A., Nicola, R. D., & Loreti, M. (2020). Programming interactions in collective adaptive systems by relying on attribute-based communication. *Science of Computer Programming*, *192*, 102428. https://doi.org/10.1016/j.scico.2020.102428.
6. Alur, R., & Grosu, R. (2004). *Dynamic reactive modules*. Tech. Rep. 2004/6, Stony Brook.
7. Alur, R., Henzinger, T., Mang, F., Qadeer, S., Rajamani, S., & Tasiran, S. (1998). MOCHA: modularity in model checking. In: *CAV'98*, pp. 521–525.
8. Alur, R., & Henzinger, T. (1999). Reactive Modules. *Formal Methods in System Design*, *15*(1), 7–48.
9. Alur, R., Henzinger, T., & Kupferman, O. (2002). Alternating-time temporal logic. *Journal of the ACM*, *49*(5), 672–713. https://doi.org/10.1145/585265.585270.
10. Attie, C. P., & Lynch, N. A. (2016). Dynamic input/output automata: A formal and compositional model for dynamic systems. *Information and Computation*, *249*, 28–75. https://doi.org/10.1016/j.ic.2016.03.008.
11. Attiya, H., Bar-Noy, A., & Dolev, D. (1995). Sharing memory robustly in message-passing systems. *Journal of the ACM*, *42*(1), 124–142. https://doi.org/10.1145/200836.200869.
12. Baier, C., & Katoen, J. (2008). *Principles of model checking*. : MIT Press.
13. Baumann, A., Barham, P., Dagand, P.E., Harris, T., Isaacs, R., Peter, S., Roscoe, T., Schüpbach, A., & Singhania, A. (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, (pp. 29–44). ACM, New York, NY, USA. https://doi.org/10.1145/1629575.1629579.
14. Belardinelli, F., Boureanu, I., Dima, C., & Malvone, V. (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, pp. 1820–1822.
15. Belardinelli, F., Lomuscio, A., Murano, A., & Rubin, S. (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, (pp. 1268–1276).
16. Belardinelli, F., Lomuscio, A., & Patrizi, F. (2014). Verification of agent-based artifact systems. *Journal of Artificial Intelligence Research*, *51*, 333–376. https://doi.org/10.1613/jair.4424.

17. Belardinelli, F., Lomuscio, A., Murano, A., & Rubin, S. (2020). Verification of multi-agent systems with public actions against strategy logic. *Artificial Intelligence*, *285*, 103302. https://doi.org/10.1016/j.artint.2020.103302.
18. Bengtson, J., Johansson, M., Parrow, J., & Victor, B. (2011). Psi-calculi: a framework for mobile processes with nominal data and logic. *Logical Methods in Computer Science*,. https://doi.org/10.2168/LMCS-7(1:11)2011.
19. Berthon, R., Maubert, B., Murano, A., Rubin, S., & Vardi, M. (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, (pp. 1–12). https://doi.org/10.1109/LICS.2017.8005136.
20. Berthon, R., Maubert, B., Murano, A., Rubin, S., & Vardi, M. Y. (2021). Strategy logic with imperfect information. *ACM Transactions on Computational Logic (TOCL)*, *22*(1), 5:1–5:51. https://doi.org/10.1145/3427955.
21. Borgström, J., Huang, S., Johansson, M., Raabjerg, P., Victor, B., Pohjola, J. Å., et al. (2015). Broadcast psi-calculi with an application to wireless protocols. *Software and System Modeling*, *14*(1), 201–216. https://doi.org/10.1007/s10270-013-0375-z.
22. Chatterjee, K., Henzinger, T., & Piterman, N. (2010). Strategy logic. *Information and Computation*, *208*(6), 677–693. https://doi.org/10.1016/j.ic.2009.07.004.
23. Dorigo, M. (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, p. 1. https://doi.org/10.1109/ICARSC.2014.6849753.
24. Dwyer, M., Avrunin, G., & Corbett, J. (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., (pp. 411–420). ACM.
25. Ene, C., & Muntean, T. (1999). Expressiveness of point-to-point versus broadcast communications. In *Fundamentals of computation theory*, (pp. 258–268). Springer.
26. Ene, C., & Muntean, T. (2001). A broadcast-based calculus for communicating systems. In *Parallel and distributed processing symposium, international*, vol. 3, (pp. 30149b–30149b). IEEE Computer Society.
27. Fagin, R., Halpern, J., Moses, Y., & Vardi, M. Y. (1995). *Reasoning about knowledge*. : MIT Press.
28. Finkbeiner, B., & Schewe, S. (2005). Uniform distributed synthesis. In *20th IEEE symposium on logic in computer science (LICS 2005)*, 26-29 June 2005, Chicago, IL, USA, Proceedings, (pp. 321–330). IEEE. https://doi.org/10.1109/LICS.2005.53.
29. Fischer, M. J., & Ladner, R. E. (1979). Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, *18*(2), 194–211. https://doi.org/10.1016/0022-0000(79)90046-1.
30. Fisher, J., Henzinger, T., Nickovic, D., Piterman, N., Singh, A., & Vardi, M. (2011). Dynamic reactive modules. In *CONCUR 2011 - concurrency theory - 22nd international conference, CONCUR 2011*, Aachen, Germany, September 6-9, 2011. Proceedings, (pp. 404–418). https://doi.org/10.1007/978-3-642-23217-6_27.
31. Gochet, P., & Gribomont, E.P. (2006). Epistemic logic. In *Logic and the modalities in the twentieth century*, (pp. 99–195). https://doi.org/10.1016/S1874-5857(06)80028-2.
32. Griesmayer, A., & Lomuscio, A. (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*, (pp. 130–145). https://doi.org/10.1007/978-3-642-38592-6_10.
33. Gutierrez, J., Harrenstein, P., & Wooldridge, M. (2017). From model checking to equilibrium checking: reactive modules for rational verification. *Artificial Intelligence*, *248*, 123–157. https://doi.org/10.1016/j.artint.2017.04.003.
34. Gutierrez, J., Perelli, G., & Wooldridge, M. (2018). Imperfect information in reactive modules games. *Information and Computation*, *261*(Part), 650–675.
35. Halpern, J., & Vardi, M. (1989). The complexity of reasoning about knowledge and time. i. lower bounds. *Journal of Computer and System Sciences*, *38*(1), 195–237. https://doi.org/10.1016/0022-0000(89)90039-1.
36. Hamann, H. (2018). *Swarm robotics - a formal approach*. : Springer. https://doi.org/10.1007/978-3-319-74528-2.
37. Helsinger, A., Thome, M., & Wright, T. (2004). Cougaar: a scalable, distributed multi-agent architecture. In *2004 IEEE international conference on systems, man and cybernetics* (IEEE Cat. No.04CH37583), vol. 2, (pp. 1910–1917) vol.2. https://doi.org/10.1109/ICSMC.2004.1399959.
38. Karagiannis, P., Matthaiakis, S. A., Andronas, D., Filis, K., Giannoulis, C., Michalos, G., & Makris, S. (2019). Reconfigurable assembly station: a consumer goods industry paradigm. *Procedia CIRP,81,* 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.
39. Kouvaros, P., Lomuscio, A., Pirovano, E., & Punchihewa, H. (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, (pp. 179–187).
40. Kozen, D. (1983). Results on the propositional mu-Calculus. *Theoretical Computer Science*, *27*, 333–354. https://doi.org/10.1016/0304-3975(82)90125-6.
41. Lomuscio, A., van der Meyden, R., & Ryan, M. (2000). Knowledge in multiagent systems: initial configurations and broadcast. *ACM Transactions on Computational Logic (TOCL)*, *1*(2), 247–284. https://doi.org/10.1145/359496.359527.
42. Lomuscio, A., Qu, H., & Raimondi, F. (2017). MCMAS: an open-source model checker for the verification of multi-agent systems. *STTT*, *19*(1), 9–30.
43. Lynch, N. A. (1977). Log space recognition and translation of parenthesis languages. *Journal of the ACM*, *24*(4), 583–590. https://doi.org/10.1145/322033.322037.
44. Maganha, I., Silva, C., & Ferreira, L. (2019). The Layout Design in Reconfigurable Manufacturing Systems: a Literature Review. In *The International Journal of Advanced Manufacturing Technology*,. https://doi.org/10.1007/s00170-019-04190-3.
45. Maoz, S., & Ringert, J. (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, (pp. 96–106). ACM.
46. Mathews, N., Christensen, A.L., Ferrante, E., O'Grady, R., & Dorigo, M. (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, (pp. 939–946). International foundation for autonomous agents and multiagent systems*.
47. Milner, R. (1975). Processes: A mathematical model of computing agents. In H. Rose, J. Shepherdson (Eds.) Logic Colloquium '73, *Studies in logic and the foundations of mathematics*, vol. 80, pp. 157 – 173. Elsevier. https://doi.org/10.1016/S0049-237X(08)71948-7. http://www.sciencedirect.com/science/article/pii/S0049237X08719487.
48. Milner, R., Parrow, J., & Walker, D. (1992). A calculus of mobile processes. *I. Information and Computation*, *100*(1), 1–40. https://doi.org/10.1016/0890-5401(92)90008-4.
49. Miyano, S., & Hayashi, T. (1984). Alternating finite automata on omega-words. *Theoretical Computer Science*, *32*, 321–330. https://doi.org/10.1016/0304-3975(84)90049-5.
50. Mogavero, F., Murano, A., Perelli, G., & Vardi, M. (2014). *Reasoning about strategies: on the model-checking problem*,. https://doi.org/10.1145/2631917.
51. Prasad, K. V. (1995). A calculus of broadcasting systems. *Science of Computer Programming*, *25*(2), 285–327.
52. Rubenstein, M., Ahler, C., Hoff, N., Cabrera, A., & Nagpal, R. (2014). Kilobot: A low cost robot with scalable operations designed for collective behaviors. *Robotics and Autonomous Systems*, *62*(7), 966–975. https://doi.org/10.1016/j.robot.2013.08.006.
53. Sistla, A., & Clarke, E. (1985). The complexity of propositional linear temporal logics. *Journal of the ACM*, *32*(3), 733–749. https://doi.org/10.1145/3828.3837.
54. Van Ditmarsch, H., & Kooi, B. (2006). The secret of my success. *Synthese*, *151*(2), 201–232.
55. van Ditmarsch, H., van der Hoek, W., & Kuijer, L. B. (2020). The logic of gossiping. *Artificial Intelligence*, *286*, 103306. https://doi.org/10.1016/j.artint.2020.103306.
56. van der Meyden, R., & Wilke, T. (2005). Synthesis of distributed systems from knowledge-based specifications. In M. Abadi, L. de Alfaro (Eds.) *CONCUR 2005 - concurrency theory, 16th international conference, CONCUR 2005*, San Francisco, CA, USA, August 23-26, 2005, Proceedings, *Lecture Notes in Computer Science*, vol. 3653, (pp. 562–576). Springer. https://doi.org/10.1007/11539452_42.
57. Vardi, M.Y. (1994). Nontraditional applications of automata theory. In *International conference on theoretical aspects of computer software, Lecture notes in computer science*, vol. 789, (pp. 575–597). Springer.
58. Vardi, M. Y., & Wolper, P. (1994). Reasoning about infinite computations. *Information and Computation*, *115*(1), 1–37.
59. Wang, C., Jiang, J., Chen, X., Yi, N., & Cui, H. (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, pp. 94–107. https://doi.org/10.1145/3127479.3128609.
60. Wong, K.W., & Kress-Gazit, H. (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, pp. 2209–2216. https://doi.org/10.1109/IROS.2016.7759346.
61. Wooldridge, M. (2002). *Introduction to Multiagent Systems*. : Wiley.

62. Yan, Q. (2006). Lower bounds for complementation of *omega*-automata via the full automata technique. In *33rd International colloquium on automata, languages and programming*, *Lecture notes in computer science*, vol. 4052, pp. 589–600. Springer.