A compositional semantics for reconfigurable multi-mode interaction in R-CHECK

International Journal on Software Tools for Technology Transfer (2026) 28:199–218
https://doi.org/10.1007/s10009-026-00861-1

FOUNDATION FOR MASTERING CHANGE

ReoCAS 2024

A compositional semantics for reconfigurable multi-mode interaction in R-CHECK

Yehia Abd Alrahman[1,2] · Shaun Azzopardi[3] · Luca Di Stefano[4] · Nir Piterman[1,2]

Accepted: 5 March 2026 / Published online: 26 March 2026

Abstract

Autonomous multi-agent systems use different modes of communication to support their autonomy and ease of interaction. In order to enable modelling and reasoning about such systems, we need frameworks that combine many forms of communication. R-CHECK is a modelling, simulation, and verification environment supporting the development of multi-agent systems, providing attributed channelled broadcast and multicast communication. Another common communication mode is point-to-point, wherein agents communicate with each other directly. Capturing point-to-point through R-CHECK's multicast and broadcast is possible, but cumbersome and prone to interference. Here, we extend R-CHECK (and its underlying formal calculus RECIPE) with bidirectional attributed point-to-point communication, which can be established based on identity or properties of participants. Moreover, we provide a compositional semantics that clearly describes how different modes of interaction co-exist without interference. We also support model-checking of point-to-point interactions by extending linear temporal logic with observation descriptors related to the participants in this communication mode. We argue that these extensions simplify the design, and demonstrate their benefits by means of an illustrative case study.

Keywords Model checking · Agent theories and models · Verification of multi-agent systems

1 Introduction

Multi-agent Systems (MAS) are some of the most interesting and challenging systems to design. This is particularly the case when tasks of the system require interaction between agents based on mutual interest and changing tasks. Machines operating in this way need to create opportunistic interactions. This is possible if agents can reconfigure their interaction interfaces and dynamically form groups at runtime based on changes in their context. We call such systems *Reconfigurable MAS* [23, 24]. We are interested in designing such systems and, due to the challenge involved, supporting reasoning about the behaviour of designed systems to improve their reliability and security.

MAS are often programmed using high-level languages that support domain-specific features of MAS. For example, emergent behaviour [7, 8, 31], interactions [9], intentions [18], knowledge [22], and so forth. These notions are too involved to be directly encoded in plain transition systems. Thus, we often want programming abstractions that focus on the domain concepts, abstract away from low-level details, and consequently reduce the size of the model under consideration. The rationale is that designing a system requires having the right level of abstraction to represent its behaviour. Furthermore, one would like to reason about the design to check that it indeed fulfils its requirements. Model checking is a prominent technique for such reasoning. Thus, model checking tools that support high-level features of Reconfigurable MAS are required to enable reasoning about high-level features of designs. We need to support an intuitive

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).

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

S. Azzopardi
shaun.a@dedaub.com

L. Di Stefano
luca.di.stefano@tuwien.ac.at

N. Piterman
nir.piterman@gu.se

[1] University of Gothenburg, Gothenburg, Sweden

[2] Chalmers University of Technology, Gothenburg, Sweden

[3] Dedaub, San Gwann, Malta

[4] Institute of Computer Engineering, TU Wien, Treitlstraße 3, 1040 Vienna, Austria

description of programs, actions, protocols, reconfiguration, self-organisation, etc.

We have previously presented RECIPE [2, 9] and R-CHECK [12], a framework and a toolkit for designing, simulating, and verifying reconfigurable multi-agent systems. RECIPE supported multiple modes of communication through predicated communication on broadcast and multicast channels. Agents could use a predicated broadcast to target only agents satisfying specific conditions. They could use a predicated multicast to ensure that all participants satisfy certain conditions. A unique feature of this framework is its active support of reconfiguration. RECIPE allows agents to connect and disconnect from multi-cast channels during runtime. Thus, the discovery of interested agents (through broadcast) and the formation of ad-hoc groups with them (through multicast) becomes simple and intuitive. While RECIPE presented a theoretical model based on transition systems and their symbolic versions, R-CHECK extended it with a high-level modelling language. R-CHECK enables reasoning about systems through simulation and model checking. In order to reason about intentions of senders, we extended LTL to LTOL, which allows next operators that are conditioned upon contents, predicates, and senders of messages. This allows further insights into the interactions that happen in the system to be included in logical specifications. LTOL model checking was supported through a translation to NUXMV [17].

One of the challenges of modelling with R-CHECK is to capture (anonymously) the existence of recipients. Indeed, both broadcast and multicast channels allow messages through in the case that there are no recipients. In order to model situations in which knowledge of the existence of others is needed, we made assumptions about sufficiently many participants being available. Based on this assumption, we were able to emulate point-to-point communication through a combination of broadcast and multicast messages. However, this was cumbersome and prone to interference, which could easily lead to deadlock. In addition, encoding point-to-point communication through a protocol of coordination that requires multiple messages, created complicated models that were hard to understand and reason about.

In the conference paper [13], we extended R-CHECK by supporting unidirectional attributed point-to-point communication, where data flows only in one direction, i.e., from the supplier to the getter. This article is an extended and an enhanced version of [13]. There are three new contributions with respect to [13].

Particularly, our new contributions are:

(i) we extend the theoretical model RECIPE and its implementation in R-CHECK by bidirectional point-to-point communication. That is, we allow both the getter and the supplier to exchange data during communication. We later show in the case study that this has a positive impact on modelling. It majorly reduces the size of R-CHECK models that rely on point-to-point communication. We show a major reduction and simplification of the case study with respect to its early implementation in [13].

(ii) we provide a formal compositional semantics for R-CHECK and show that it behaves as expected. That is, the semantics describes systems both at local and distributed level, and the parallel composition is a commutative monoid.

(iii) we develop a new open-source implementation of R-CHECK as a Visual Studio Code (VSCode) extension, featuring improved parser and providing better error reporting, syntax highlighting, and keyword autocompletion. Moreover, it supports basic type-checking.

This article is structured as follows: in Sect. 2, we give a background on RECIPE [2, 9], the underlying theory of R-CHECK. In Sect. 3, we augment the language of R-CHECK with point-to-point communication and its symbolic semantics. In Sect. 4 and Sect. 5, we formally present the syntax and the compositional semantics of R-CHECK. In Sect. 6 we extend the LTOL logic to allow specification of point-to-point communication. In Sect. 7, we provide a case study to model autonomous resource allocation, and in Sect. 8 we introduce the new implementation of R-CHECK. Finally, we discuss related work in Sect. 9 and report our concluding remarks in Sect. 10.

2 Background materials: the RECIPE formalism

We present background materials necessary to introduce our language extension and its semantics.

RECIPE [2, 9] is a symbolic concurrent formalism that serves as the underlying semantics of R-CHECK. RECIPE relies on (attributed-) channel communication. Agents agree on a set of channel names CH to exchange messages on. These messages carry data (in variables D) specified by senders. Agents can constrain the targets of communication by attributing the messages through predicates, similar to *AbC* [7, 8]. As opposed to the latter, RECIPE supports dynamic reconfiguration by letting agents disconnect from channels. Moreover, RECIPE supports two kinds of communication, *channelled-broadcast* and *channelled-multicast*. In channelled-broadcast the communication is non-blocking, that is, the communication can still go through if a targeted receiver is not ready to engage. Contrarily, in multicast, the communication is blocking until all targeted receivers are willing to accept the message and engage in the communication. Thus, the set of channels CH includes a channel used

exclusively for broadcast, $\star$, which agents cannot disconnect from.

Usually, broadcast is used for service discovery: for instance, when agents are unaware of the existence of each other, and want to be discovered or to establish links for further interaction. On the other hand, multicast can capture a more structured interaction where agents have dedicated links to interact on. The reconfiguration of interaction interfaces in ReCiPe makes it possible to integrate the two ways of communication in a meaningful way. That is, agents may start with a flat communication structure and use broadcast to discover others. With ReCiPe's channel passing, agents can dynamically build dedicated communication structures based on channel references they exchange.

In order to target a subset of agents in an interaction, sending agents rely on a set of *property identifiers* PV, i.e., identifiers that senders use to specify properties required of targeted receivers. For instance, agent k may specify that it wants to communicate on channel a with all agents that listen to a and satisfy property $\mathsf{BatteryLevel} \geq 30\%$. In other words, property identifiers are used by agents to indirectly specify constraints on the targeted receivers in a similar manner to the attribute-based paradigm [7, 8].

Each agent has a way to relate property identifiers to its local state through a re-labelling function f. We have generalised this function in R-CHECK to deal with more sophisticated expressions. Thus, agents specify properties anonymously using these identifiers, which are later translated to the corresponding receiver's local state. Messages are then only delivered to receivers that satisfy the property after relabelling.

Formally, an agent can be defined symbolically in terms of a Discrete System (DS) [29]. A DS can be thought of as an encoding of a transition system through Boolean predicates over a set of system variables. To encode the current state and the next state of the system, the two copies of system variables are used. The assignments to the original copy of variables, say V, are used to denote the current state of the system. Moreover, a primed copy V' is used where its assignments denote the next state of the system after command execution. In this way, the satisfaction of a Boolean predicate over the assignments of V and V' denotes the execution of system event.

More precisely, an agent is defined as follows:

Definition 1 (Agent)
An agent is a tuple $A = \langle V,\ f,\ g^s,\ g^r, \mathcal{T}^s,\ \mathcal{T}^r, \theta\rangle$,

- V is a finite set of typed local variables.
- $f : \text{PV} \rightarrow V$ is a function, associating property identifiers to local variables.
- $g^s \subseteq V \times \text{CH} \times \text{D} \times \text{PV}$ is a send guard specifying the property of the targeted receivers. Based on the current assignments of V, CH, and D, g^s simplifies to a predicate over PV and it is evaluated over the state of every receiver j by applying f_j.
- $g^r \subseteq V \times \text{CH}$ is a receive guard describing the connectedness of an agent to a channel ch. We let $g^r(v, \star) = \mathsf{true}$ for every v, i.e., every agent is always connected to the broadcast channel.
- $\mathcal{T}^s \subseteq V \times V' \times \text{D} \times \text{CH}$ and $\mathcal{T}^r \subseteq V \times V' \times \text{D} \times \text{CH}$ are assertions describing, respectively, the send and receive transition relations. We assume that an agent is broadcast input-enabled, i.e., $\forall v, \mathbf{d}\ \exists v'$ s.t. $\mathcal{T}^r(v, v', \mathbf{d}, \star)$ holds.
- θ is an assertion on V describing the initialization of the agent.

In this definition, a state of an agent s is an assignment to the agent's local variables V, i.e., for $v \in V$ if $\mathsf{Dom}(v)$ is the domain of v, then s is an element in $\prod_{v \in V} \mathsf{Dom}(v)$. In case that all variables range over a finite domain then the number of states is finite. A state is initial if its assignment to V satisfies θ. Note that A is a discrete system, and thus we use the set V' to denote the primed copy of V. That is, V' stores the next assignment to V. Moreover, we use Id to denote the assertion $\bigwedge_{v \in V} v = v'$. That is, V is kept unchanged. We use $\mathbf{d}$ to denote an assignment to the data variables D. We also abuse the notation and use f for the assertion $\bigwedge_{pv \in \text{PV}} pv = f(pv)$.

Agents exchange messages of the form $m = (ch, \mathbf{d}, i, \pi)$, where ch is the channel m is sent on, $\mathbf{d}$ the data it carries, i the sender identity (we assume a unique identifier for each agent), and π the assertion specifying the property of targeted receivers. The predicate π is obtained by grounding the sender's send guard on the sender's current state, used channel ch, and exchanged data $\mathbf{d}$.

Send transition relations $\mathcal{T}^s$ characterise what messages may be sent, with one message sent at each point in time, whereas receive transition relations $\mathcal{T}^r$ characterise the reaction of a receiving agent to a message.

We use $\text{KEEP}(X)$ to denote that a set of variables X is not changed by a transition (either send or receive). That is, $\text{KEEP}(X)$ is equivalent to the assertion $\bigwedge_{x \in X} x = x'$. Note that $\mathsf{Id} = \text{KEEP}(V)$.

A set of agents agreeing on property identifiers PV, data variables D, and channels CH defines a *system*. We give the semantics of systems in terms of predicates to facilitate efficient symbolic analysis (through BDD or SMT). We use $\uplus$ for disjoint union.

Formally, a ReCiPe system is also a DS, defined as follows:

Definition 2 (System)
Given a set $\{A_i\}_i$ of agents, a system is $S = \langle \mathcal{V},\ \rho,\ \theta\rangle$, where $\mathcal{V} = \biguplus_i V_i$, a state of the system "s" is in $\prod_i \prod_{v \in V_i} \mathsf{Dom}(v)$

and the initial assertion $\theta = \bigwedge_i \theta_i$. The transition relation ρ is as follows:

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

The transition relation ρ describes two modes of interactions: blocking multicast and non-blocking broadcast. Formally, ρ relates a system state s to its successors s' given a message $m = (ch, \mathbf{d}, k, \pi)$. Namely, there exists an agent k that sends a message with data $\mathbf{d}$ (an assignment to D), on channel ch, with assertion π (obtained as $g_k^s(v_k, ch, \mathbf{d}, \cdot)$) on channel ch and all other agents are either (a) connected to channel ch, satisfy the send predicate π, and participate in the interaction (i.e., have a corresponding receive transition for the message), (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 $\exists \mathrm{PV}.f_j$) 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 $\exists \mathrm{PV}.f_j \wedge \pi$. The relation ensures that, when sending on a channel different from $\star$, the set of receivers is the full set of *connected* agents. On the broadcast channel agents not satisfying the send predicate do not block the sender.

Example 1
Consider a RECIPE system that is composed of two agents A_1 and A_2, agreeing on the set of channels $\mathrm{CH} = \{\star, \mathsf{c}\}$, the data variables $\mathrm{D} = \{\mathrm{MSG}, \mathrm{LNK}\}$, and the property variables $\mathrm{PV} = \{pv\}$. Let us also assume they also agree on the existence of an enumerated type `enum` that contains at least an element named client. Here, we use non-Boolean variables to simplify the presentation.

A_1 is defined as follows:
- $V_1 = \{\mathsf{cLink} : \texttt{channel},\ \mathsf{role} : \texttt{enum}\}$
- $f_1 = \{pv \mapsto \mathsf{role}\}$
- g_1^s is $(ch = \star \wedge pv = \mathsf{client})$
- g_1^r is true
- $\mathcal{T}_1^s$ is $(\mathrm{KEEP}(\mathrm{V}_1) \wedge \mathbf{d}(\mathrm{MSG} \mapsto \mathsf{join}, \mathrm{LNK} \mapsto \mathsf{c}) \wedge \mathsf{ch} = \star)$
- $\mathcal{T}_1^r$ is $\mathrm{KEEP}(V_1)$
- θ_1 is $(\mathsf{cLink} = \mathsf{c} \wedge \mathsf{role} = \mathsf{client})$

That is, A_1 has two local variables cLink of `channel` type and role of `enum` type. Moreover, A_1 relabels the property identifier pv locally as the value of its local variable role. The send predicate g_1^s indicates that A_1 intends to interact on the broadcast channel $\star$ with agents that satisfy the property $pv = \mathsf{client}$ according to their local relabelling. The receive predicate g_1^r indicates that A_1 is always enabled to receive.

Behaviour-wise, A_1 can send a message join with a link c on the broadcast channel $\star$. Moreover, A_1 is not willing to receive any messages. Initially, the local variables of A_1 are set such that cLink is assigned link c and role is a client.

A_2 is defined as follows:
- $V_2 = \{\mathsf{cLink} : \texttt{channel},\ \mathsf{role} : \texttt{enum}\}$
- $f_2 = \{pv \mapsto \mathsf{role}\}$
- g_2^s is false
- g_2^r is true
- $\mathcal{T}_2^s$ is false
- $\mathcal{T}_2^r$ is $(\mathsf{cLink} = \bot \wedge \mathsf{cLink}' = \mathbf{d}(\mathrm{LNK}) \wedge \mathrm{KEEP}(\mathsf{role}) \wedge \mathbf{d}(\mathrm{MSG} \mapsto \mathsf{join}) \wedge \mathsf{ch} = \star)$
- θ_2 is $(\mathsf{cLink} = \bot \wedge \mathsf{role} = \mathsf{client})$

Clearly, A_2 only differs from A_1 with respect to the send guard, the send transition relation (which are set to false), the receive transition relation (which indicates that A_2 is willing to receive a message named join and stores the value of LNK of the message in cLink) and the initial condition where cLink is set to $\bot$. By applying Def. 2, we have that the composition of A_1 and A_2 indeed forms a RECIPE system (where local variables of A_1 and A_2 are joined with disjoint union to account for similar local naming).

Now, starting from the initial conditions of both agents, we apply the system transition relation ρ. Clearly, there exist only one message that satisfies ρ, namely the message on channel $\star$ and data variables assigned as follows $\{\mathrm{MSG} \mapsto \mathsf{join}, \mathrm{LNK} \mapsto \mathsf{c}\}$, where A_1 is the sender (i.e., its send transition relation $\mathcal{T}_1^s$ is satisfied). Moreover, there is only one receiver A_2 which is connected to $\star$ (i.e., g_2^r is satisfied), its receive transition relation $\mathcal{T}_2^r$ is satisfied with respect to the same message, and the send guard g_1^s is $(ch = \star \wedge pv = \mathsf{client})$ in conjunction to local relabelling of A_2 (i.e., $pv = \mathsf{role}$) is satisfiable. Thus, ρ holds and as a result A_2 sets its local cLink variable to c that is communicated in the message. In the next cycle, ρ is checked again based on the new updated states.

3 Extending RECIPE with attributed point-to-point communication

We propose a Point-to-Point communication extension to R-CHECK. However, to be able to support this, we first need

to extend the semantic framework, i.e., RECIPE. Notice that we rely on RECIPE as the underlying semantic framework for R-CHECK.

3.1 RECIPE with point-to-point communication

There are several ways to support Point-to-Point Communication in the literature. For instance, we can use the complementary send/receive communication as in π-calculus [26] or the tuple-space approach as in Klaim [19]. In our case, we decided to use a specialised attributed Point-to-Point Communication that takes inspiration from the tuple-space approach while keeping models amenable to formal verification. Note that a tuple-space approach, where agents are allowed to put/get tuples to/from a shared/private tuple-space, can imply higher-order communication. A tuple can be simply the code of an agent. Moreover, a tuple-space is usually modelled as a parallel composition of existing tuples. This means that the size of the tuple space can grow uncontrollably, and thus lead to verification problems.

Our approach consists of eliminating the *verification-problematic* tuple space, and encoding it as parametric supply-transitions in the code of each agent. Namely, we provide two primitives: get and $supply$. The get allows an agent to nondeterministically get data from another agent based on either satisfaction of a predicate g^p or on the identity of the agent (its locality). That is, an agent can ask for data from a potential supplier by either supplying the name of the targeted agent (i.e., its locality ℓ) or predicating on the targeted agent's state. Instead of creating a private tuple space for each agent, we provide local state-parametric $supply$-transitions for agents willing to supply data to others. Namely, a supplier is another agent with a matching supply transition. Note that matching here can be attributed (i.e., based on predicate satisfaction) or directed (i.e., based on named locality). At a system level, the names (localities) that compose the system are known. We introduce a reserved word "any" to denote a wild card over the localities in the system. Thus, when agents refer to localities they can be either from the set of names of agents or the keyword "any". Formally, we extend Def. 1 as follows.

Definition 3 (Point-to-Point Extended Agent)
An *agent* is a tuple $A = \langle V,\ f,\ g^s,\ g^r,\ g^p,\ \mathcal{T}^s,\ \mathcal{T}^r, \mathcal{T}^{\mathsf{G}}, \mathcal{T}^{\mathsf{S}}, \theta\rangle$, where:

- $g^p \subseteq V \times \text{PV}$ is a get-guard specifying the property of the targeted supplier. Similar to send guards, the get-guard g^p is evaluated based on the current evaluation of V of the getter to a predicate over PV and it is evaluated over the state of one supplier j by applying f_j.
- $\mathcal{T}^{\mathsf{G}} \subseteq V \times V' \times \mathbf{d} \times \ell$ is an assertion describing the get-transition relation. Namely, given the current assignment to local variables V, the get-transition relation specifies the data $\mathbf{d}$ the getter is interested in, from what locality ℓ, and the updates to local variables V' if the transition is executed. As mentioned ℓ ranges over the names of agents in the system and "any".
- $\mathcal{T}^{\mathsf{S}} \subseteq V \times V' \times \mathbf{d} \times \ell$ is an assertion describing the supply transition relation. Similarly, the supply transition relation specifies the data that the supplier is willing to provide given that the assertion over V, V', and ℓ is satisfied.
- all other components are defined as before in Def. 1

Now, we are ready to define a RECIPE system and its semantics. The construction of system is exactly as reported in Def. 2. The only thing that substantially changes is the system-level semantics. Our goal is to provide a well-behaved predicate semantics for point-to-point communication while co-existing with the original broadcast and multicast semantics.

The main question that we need to answer is what happens when a point-to-point communication transition is concurrently enabled with a broadcast or multicast in a given state of the system. We could have prioritised one mode of communication over another and define the semantics accordingly. However, we decided to stay general and refrain from resolving nondeterminism at semantic level. Thus, we decided to nondeterministically select one enabled transition. This choice not only abstains from dealing with scheduling issues which are rather implementation concerns, but also simplifies the semantics. Thus, the new semantics is

$$\hat{\rho} = \rho \vee \rho_{gs}, \text{ where } \rho_{gs} \text{ is defined as:}$$

$$\begin{aligned}\rho_{gs} = \exists \ell.\exists \text{D}.\ & \bigvee_k \mathcal{T}_k^{\mathsf{G}}(V_k, V'_k, \mathbf{d}, \ell) \wedge \\ & \bigvee_{j \neq k} \exists \text{PV}.f_j \wedge \mathcal{T}_j^{\mathsf{S}}(V_j, V'_j, \mathbf{d}, \ell) \wedge \\ & \begin{pmatrix} \ell = j \\ \vee \\ \ell = \mathsf{any} \wedge \mathsf{g}^{\mathsf{p}}(V_k, \text{PV}) \end{pmatrix} \wedge \bigwedge_{i \neq k, i \neq j} \mathsf{Id}_i\end{aligned}$$

Since we decided to refrain from resolving nondeterminism at semantic level, we model the nondeterminism of selection as an or-predicate. That is, we consider the original transition relation ρ in Def. 2, and we define an extension relation $\hat{\rho}$ as an or-predicate over the original ρ and the point-to-point semantics.

Now, the extended transition relation $\hat{\rho}$ describes three modes of interaction: blocking multicast, non-blocking broadcast, and blocking unicast (or point-to-point). In case of unicast, $\hat{\rho}$ relates a system state s to its successors s' given an exchanged tuple $t = (\ell, \mathbf{d}, k, \pi)$ where ℓ is a locality, $\mathbf{d}$ is

a data assignment, k is the getter locality, and π is the getter-predicate, obtained by initially evaluating $\mathsf{g}^{\mathsf{p}}(V_k, \text{PV})$ over the getter local state. Namely, there exists an agent k that gets a tuple with data $\mathbf{d}$ (an assignment to D) with assertion π (an assignment to $\mathsf{g}^{\mathsf{p}}(V_k, \text{PV})$) from locality ℓ and there exists another agent j such that either (a) agent j is an exact match of the target locality, i.e., $\ell = j$ and can participate in the interaction (i.e., have a corresponding supply transition for the tuple), or (b) the target locality is any (i.e., any agent can match) and agent j satisfies the get-guard. In either case, all other agents that are different from k and j stay idle. If no supplier exists then the communication is blocked. That is, the whole predicate will evaluate to false. Notice that in the case the locality refers to the identify of an agent, the assertion π is not used.

Example 2
Consider again the system in Example 1, when extended with point-to-point communication as follows: we consider that the locality of agent A_1 and A_2 to be "$\mathsf{A_1}$" and "$\mathsf{A_2}$" respectively. That is, the locality of an agent corresponds to its unique identity. We define the extended components of A_1 and A_2, and the rest are the same as in Example 1.

- g_1^p is true
- $\mathcal{T}_1^{\mathsf{G}}$ is $(\text{KEEP}(\mathsf{role}) \wedge \mathsf{cLink} = \bot \wedge \mathsf{cLink}' = \mathbf{d}(\text{LNK}) \wedge \mathbf{d}(\text{ID} \leftarrowtail \text{"}\mathsf{A_1}\text{"}) \wedge \ell = \text{"}\mathsf{A_2}\text{"})$
- $\mathcal{T}_1^{\mathsf{S}}$ is false

The get-guard g_1^p has no restrictions. The get transition relation $\mathcal{T}_1^{\mathsf{G}}$ defines a single transition where A_1 gets a communication link form the agent with locality "$\mathsf{A_2}$" when its local variable cLink is not assigned. Moreover, A_1 sends its locality in return, i.e., $\mathbf{d}(\text{ID} \leftarrowtail \text{"}\mathsf{A_1}\text{"})$. Consequently, A_1 assigns the value of cLink of the corresponding supplied link from the message $\mathbf{d}(\text{LNK})$.

The supply transition relation of A_1 does not contain any supply transitions, and thus it is set to false. Namely, a get transition targeting the locality of A_1 is always blocked.

The extended components of agent A_2 are defined as follows:

- g_2^p is true
- $\mathcal{T}_2^{\mathsf{G}}$ is false
- $\mathcal{T}_2^{\mathsf{S}}$ is $(\text{KEEP}(V_2) \wedge \mathbf{d}(\text{LNK} \mapsto \mathsf{e}) \wedge \ell = \text{"}\mathsf{A_2}\text{"})$

Conversely, the get transition relation of A_2 does not contain any get transitions, and thus it cannot match any supply transitions of other agents. The supply transition relation, on the other hand, defines a single transition, where it supplies a communication link e to any agent that issues a get transition targeting the locality of agent A_2. Moreover, the supply transition does not use the sent locality from A_1.

Clearly, the composition of A_1 and A_2 can enable a get transition at system level according to the system transition relation $\hat{\rho}$. Indeed, there is a locality match "$\mathsf{A_2}$", and exchanged data are aggregated in the assignment $\mathbf{d}$. Thus, when cLink of agent A_1 is not assigned then this transition is possible.

4 Syntax of the R-CHECK language

Based on the semantics in the previous section, we suggest a deployment in R-CHECK. Here, we define the syntax of the R-CHECK language as reported in Table 1.

The top-level component of R-CHECK syntax is a system. A system is either an agent $\Gamma : P$ or a parallel composition of systems $S_1 \| S_2$. An agent is composed of a configuration $\Gamma = \langle \gamma, \mathsf{g}^{\mathsf{r}}, \mathsf{id}, f \rangle$ and a process P. A configuration consists of: a local store $\gamma : V \to \mathcal{D}$ that maps variables V of an agent to their values $\mathcal{D}$; the locality of an agent id, the receive predicate of an agent $\mathsf{g}^{\mathsf{r}}(V, \text{CH})$, and also the relabelling function $f : \text{PV} \to V$ as explained before. We use the notation $\tilde{\cdot}$ to denote a sequence of elements, and $\tilde{s}, \tilde{s}'$ to denote the sequence resulting from concatenating $\tilde{s}$ and $\tilde{s}'$.

For example, agent A_1 from Example 1 has the initial configuration $\langle(\mathsf{cLink} \mapsto \mathsf{c}, \mathsf{role} \mapsto \mathsf{client}), ch = \star, \text{"}\mathsf{A_1}\text{"}, pv \mapsto \mathsf{role}\rangle$, setting variable cLink to the value c and the variable role to client, declaring that it is listening only to the broadcast channel, identifying itself as "$\mathsf{A_1}$", and stating that the property variable pv is named locally role.

An R-CHECK process can be an action-prefixed process $a; P$, a nondeterministic choice $P + P$, a recursive process rep X. P, a guarded process $\{\pi(\tilde{x})\}P$, and the deadlocked process 0. We assume that processes are closed, i.e., all occurrences of variables X, Y are bound. In practice, we limit our syntax to non-terminating processes., i.e., processes of the form rep X. P. Moreover, our guarded process $\{\pi(\tilde{x})\}P$ may constrain message data. That is, when P is of the form $a; P'$ the predicate $\pi(\tilde{x})$ can both constrain state variables and incoming message data after a substitution to the variables in $\tilde{x}$.

An action can be a get action $\mathsf{Get}(\tilde{x}, \tilde{e})@(\ell, \mathsf{g})\mathsf{U}$, a supply action $\mathsf{Supply}(\tilde{x}, \tilde{e})@(\ell)\mathsf{U}$, a send action $\mathsf{g!c}(\tilde{e})\mathsf{U}$, or a receive action $(\tilde{x})\mathsf{c?U}$.

A get action is used to collect a sequence of data from a supplier, and substitute their values in their corresponding placeholders $\tilde{x}$. The latter can be later used to perform a sequence of updates (U) on local variables. Moreover, if a matching supplier is found, then the getter has the ability to pass a sequence of data $\tilde{e}$ to that supplier, thus providing a bidirectional information flow. The getter can either specify the locality of the targeted supplier ℓ or may accept data from any supplier that satisfies the getter predicate g. The latter is an assertion over the supplier local variables (up to relabelling), and is also parametric to the getter local variables. Parameterisation allows the getter to dynamically scope the communication.

Table 1 The syntax of the R-CHECK language

(System)	$S ::= \Gamma : P \quad \mid \quad S_1 \,\|\|\, S_2$
(Config)	$\Gamma ::= \langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f \rangle$
(Process)	$P ::= a; P \quad \mid \quad P + P \quad \mid \quad \mathsf{rep\ X}.\ P$
	$\mid \{\pi(\tilde{x})\}P \quad \mid \quad \mathsf{X} \quad \mid \quad 0$
(Action)	$a ::= \mathsf{Get}(\tilde{x}, \tilde{e})@(\ell, \mathsf{g})\mathsf{U} \quad \mid \quad \mathsf{Supply}(\tilde{x}, \tilde{e})@(\ell)\mathsf{U}$
	$\mid \mathsf{g!c}(\tilde{e})\mathsf{U} \quad \mid \quad (\tilde{x})\mathsf{c?U}$
(Channels)	$c ::= ch \quad \mid \quad \star \quad \mid \quad \mathsf{self}.v$
(Locality)	$\ell ::= \mathsf{id} \quad \mid \quad \mathsf{self} \quad \mid \quad \mathsf{any}$
(Data)	$e ::= ch \quad \mid \quad d \quad \mid \quad e_1 \times e_2$

A supply action is the get co-action, and it basically supplies a sequence of data $\tilde{e}$ to the getter if it either satisfies the getter predicate or if the getter is uniquely targeting the supplier by its locality ℓ. Moreover, the supplier may receive a sequence of data from the getter and substitute them in their corresponding placeholders $\tilde{x}$, which can be used to perform a sequence of local updates (U).

The get-supply communication mechanism provides a specialised point-to-point communication with bidirectional information flow. R-CHECK also supports group communication through broadcast and multicast using the send and receive actions.

A send action is used to send a message $\tilde{e}$ to all agents listening to a channel c and also satisfying the sender predicate g, which is semantically similar to the getter predicate. Accordingly, the agent may perform a sequence of local updates U as side effects. Note that c serves as a place holder for a channel name which can also be parametric to local variables to provide a dynamic scoping mechanism. A channel c can be a blocking multicast channel ch or the non-blocking broadcast channel $\star$.

Accordingly, a receive action accepts a message on channel c if the agent listens to c and satisfies the sender predicate g. Note that an agent listens to a multicast channel if its receive predicate g^{r} is satisfied. We require that agents cannot disconnect the broadcast channel.

A locality ℓ can be a supplier identity id, a self reference that is evaluated to the identity of the agent, or the keyword any, which denotes that any supplier is accepted to participate in the interaction. Notice in the semantics of R-CHECK, we limit the use of any to attributed point-to-point interaction. That is, any is always evaluated with respect to a getter predicate on the potential supplier.

A data e can be a multicast channel name ch, an immediate value d or a binary operation over data $\times$.

Example 3

Due to the choice to concentrate on non-terminating processes of the form rep X. P, we give the representation of A_1 and A_2 in terms of recursive definitions of Example 1 and Example 2. We explain only the part of the different processes that correspond to the two possible communication exchanges in the example. Thus, A_1's communication includes the recursive choice between the send and get rep X. $(P_1^s + P_1^g)$. Here P_1^s is $(pv = \mathsf{client})! \star (\mathsf{join}, \mathsf{c}); X$ and P_2^g is $\mathsf{Get}(\text{LNK}, \text{“}\mathsf{A}_1\text{”})@(\text{“}\mathsf{A}_2\text{”}, \mathsf{true})\mathsf{cLink} = \text{LNK}; X$. Namely, P_1^s includes the restriction on receivers whose property variable is set to client in the guard and the referral to broadcast in the choice of channel. It then, performs an empty update and then loops back to X. The get disjunct, P_1^g, clarifies that it expects the supplier to supply one value, which A_1 refers to under the name of LNK and it gives its own locality “A_1” to the supplier. It then clarifies that the only supplier it wants to interact with is “A_2” and that following the exchange it will update the value supplied by the supplier into its local variable $cLink$. Following the update it loops back to X allowing the outer loop to continue the execution.

On the other side A_2's communication includes the recursive choice between receive and supply rep X. $(P_2^r + P_2^{su})$. Here P_2^r is $\{\text{MSG} = \mathsf{join}\}(\text{MSG}, \text{LNK})\star?(cLink = \text{LNK}); X$ and P_2^{su} is $\mathsf{Supply}(\_, \mathsf{e})@(\text{“}\mathsf{A}_2\text{”}); X$. Namely, P_2^r restricts in the pre-condition the data in the message to identify the message as a join message, it then clarifies that this receive expect a message in two parts named MSG and LNK, where the former was already used in the guard, that this is a receive on the broadcast channel, and that it updates its local variable $cLink$ with the second part of the message. Following this update it loops back to X allowing the outer loop to continue the execution. Similarly, P_2^{su} indicates that A_2 is only ready to supply to others who know its identity “A_2”, that it will not use the information the getter transfer to it (in this case the locality “A_1”), and that the data that it supplies is the name of the channel e. Accordingly, the update is empty and this part loops back to X.

As mentioned early, the semantics of R-CHECK is given through the RECIPE formalism. More precisely, translate R-CHECK syntax initially to symbolic automata where transitions labels encode R-CHECK commands and states encode the control flow of processes. Once the symbolic automaton is constructed then there is a direct compilation to the RECIPE formalism which serves as the underlying semantics of R-CHECK.

Technically speaking, the behaviour of each R-CHECK agent is represented by a first-order predicate that is defined as a disjunction over the guarded actions of that agent. Moreover, both guarded commands can be represented by a disjunctive normal form predicate of the

form $\bigvee(\bigwedge_j \mathsf{assertion}_j)$. That is, a disjunct of all possible guarded transitions enabled in each step of a computation. For full exposition of the semantics, the reader is referred to [12].

Although this encoding is important to facilitate efficient symbolic analysis (through BDD or SMT), it is still quite complicated to understand and use in other applications. The predicate semantics consider a closed system and does not allow us to build systems incrementally. The latter is important to reason about open systems. For instance, we would like to support bidirectional information flow in point-to-point communication, while making it clear which part of the data concerns the getter or the supplier. As seen in the definition above, bidirectional information flow in point-to-point is modelled by ρ_{gs}, but the quantification over D makes it implicit. Of course, one can force this in ρ_{gs}, but it would defeat the purpose of predicate semantics as being close to BDDs. Thus, in the next section, we propose a symbolic, yet compositional semantics for R-CHECK that describes both agents and their compositions in a clear way. We can also single out components that are rather hidden in the closed semantics. The proposed semantics is intended to provide a clear understanding of how different communication primitives co-exist without interference. They also lay the basis for other compositional verification techniques.

5 A compositional semantics for R-CHECK

The semantics of R-CHECK will be defined both at agent and system level. We use the transition relation $\longmapsto$ to describe the local semantics of an agent. We will build on this and define a system level semantics of an R-CHECK system using the transition relation $\rightarrow$.

5.1 Agent-level semantics

The transition relation $\longmapsto \subseteq (\mathsf{Agent} \times \mathsf{L} \times \mathsf{Agent})$ defines the behaviour of an agent. Intuitively, this relation states that an $A \in \mathsf{Agent}$ executes an action exposed as a label $l \in \mathsf{L}$ and evolves to another agent $A' \in \mathsf{Agent}$. The set L is partitioned into the set of positive labels L' and the set of negative labels $\{(\pi?(\tilde{d})^{\mathsf{c}})\not\downarrow\}$. The latter denotes the case when an agent is not able to participate in message-reception (but does not block it).

The set of positive labels L' is defined below.

$$\mathsf{L}' = \{\pi!(\tilde{d})^{\mathsf{c}},\ \pi?(\tilde{d})^{\mathsf{c}},\ \tilde{d}_1, \tilde{d}_2@(\ell,\pi)^{\mathsf{G}},\ \tilde{d}_1, \tilde{d}_2@(\ell,\pi)^{\mathsf{S}}\}$$

These labels are used to denote the execution of the send, receive, get, and supply actions correspondingly. We will use λ to range over elements in this set.

The positive semantics of an R-CHECK agent is reported in Table 2.

A message send is defined by rule SND. The rule states that when a send action on c is executed, both the channel holder c and the sequence of data $\tilde{e}$ are evaluated according to the local store γ of the agent. Moreover, the closure of the sender predicate $\{\mathsf{g}\}_\gamma$ is computed by evaluating occurrences of local variables according to the local store γ. The closure can be computed compositionally, namely $\{\mathsf{g}\}_{\gamma\circ f}$ is computed by initially computing the closure under f and later under γ. The message is then emitted with a concrete message on a channel that is either broadcast or multicast. As a result, the local store γ may get updated $\gamma \leftarrowtail \mathsf{U}$, and the agent is ready to execute the next step.

Thus, "A_1" from Example 3 has an agent level transition

$$\Gamma_0^1 : P_1^s \xmapsto{(pv=\mathsf{client})!((\mathsf{join},\mathsf{c}))^\star} \Gamma_0^1 : X$$

where $\Gamma_0^1 = \langle(\mathsf{cLink} \mapsto \mathsf{c}, \mathsf{role} \mapsto \mathsf{client}), ch = \star, \text{“}\mathsf{A}_1\text{”}, pv \mapsto \mathsf{role}\rangle$ is the initial configuration of A_1 (mentioned earlier) and P_1^s is given in Example 3. Notice that the configuration of A_1 does not change by this transition.

A message receive is defined by rule RCV. Namely, an agent can receive a broadcast or a multicast message on channel c if the agent is listening to the same channel $[\![\mathsf{c}']\!]_\gamma = \mathsf{c}$, it satisfies the sender predicate (up to relabelling, i.e., the closure $\{\pi\}_{\gamma\circ f}$ holds), and listens to the channel $\{\mathsf{g}^{\mathsf{r}}[\mathsf{c}/\mathsf{ch}]\}_\gamma$. Recall that the latter must hold by assumption for the broadcast channel $\star$. Moreover, a sequence of updates $\gamma_{\leftarrowtail \mathsf{U}[\tilde{d}/\tilde{x}]}$ based on the message data may be executed.

Agent A_2 from Examples 3 has the agent-level receive transition

$$\Gamma_0^2 : P_2^r \xmapsto{pv=\mathsf{client}?((\mathsf{join},\mathsf{c}))^\star} \Gamma_1^2 : X,$$

where $\Gamma_0^2 = \langle(\mathsf{cLink} \mapsto \bot, \mathsf{role} \mapsto \mathsf{client}), ch = \star, \text{“}\mathsf{A}_2\text{”}, pv \mapsto \mathsf{role}\rangle$, $\Gamma_0^2 = \langle(\mathsf{cLink} \mapsto \mathsf{c}, \mathsf{role} \mapsto \mathsf{client}), ch = \star, \text{“}\mathsf{A}_2\text{”}, pv \mapsto \mathsf{role}\rangle$, and P_2^r is given in Example 3. Notice that $(pv = \mathsf{client})$ is satisfied by concretising pv to refer to role and checking the value of role in Γ_0^2.

A get action is defined by rule GET. An agent can get a sequence of data $\tilde{d}_2$ from another agent by issuing a get action, selecting the supplier nondeterministically by means of satisfying the getter predicate g or deterministically by means of targeting its locality ℓ. In practice, the getter predicate is only evaluated when $\ell = \mathsf{any}$, and ignored otherwise. Thus, a get on a locality $\ell \neq \mathsf{any}$ is established without considering the getter predicate. Note that if a supplier is found, the getter can also transfer a sequence of data $\tilde{d}_1$ to the selected supplier in return, while the received data $\tilde{d}_2$ can be used to update the getter local store $\gamma_{\leftarrowtail \mathsf{U}[\tilde{d}_2/\tilde{x}]}$.

Table 2 Positive Process Semantics

$$\frac{[\![\tilde{e}]\!]_\gamma = \tilde{d} \qquad [\![\mathsf{c}]\!]_\gamma = \mathsf{c}' \qquad \{\mathsf{g}\}_\gamma = \pi}{\langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : \mathsf{g}!\mathsf{c}(\tilde{e})\mathsf{U}; P \xmapsto{\pi!(\tilde{d})^{\mathsf{c}'}} \langle \gamma_{\leftarrow \mathsf{U}},\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : P}\ \textsc{Snd}$$

$$\frac{[\![\mathsf{c}']\!]_\gamma = \mathsf{c} \qquad \{\pi\}_{\gamma\circ f} \qquad \{\mathsf{g}^{\mathsf{r}}[\mathsf{c}/\mathsf{ch}]\}_\gamma}{\langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : (\tilde{x})\mathsf{c}'?\mathsf{U}; P \xmapsto{\pi?(\tilde{d})^{\mathsf{c}}} \langle \gamma_{\leftarrow \mathsf{U}[\tilde{d}/\tilde{x}]},\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : P}\ \textsc{Rcv}$$

$$\frac{[\![\tilde{e}]\!]_\gamma = \tilde{d}_1 \qquad \{\mathsf{g}\}_\gamma = \pi}{\langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : \mathsf{Get}(\tilde{x},\tilde{e})@(\ell,\mathsf{g})\mathsf{U}; P \xmapsto{\tilde{d}_1,\tilde{d}_2@(\ell,\pi)^{\mathsf{G}}} \langle \gamma_{\leftarrow \mathsf{U}[\tilde{d}_2/\tilde{x}]},\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : P}\ \textsc{Get}$$

$$\frac{\ell = \mathsf{self} \qquad [\![\tilde{e}]\!]_\gamma = \tilde{d}_2}{\langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : \mathsf{Supply}(\tilde{x},\tilde{e})@(\ell)\mathsf{U}; P \xmapsto{\tilde{d}_1,\tilde{d}_2@(\mathsf{id},\pi)^{\mathsf{S}}} \langle \gamma_{\leftarrow \mathsf{U}[\tilde{d}_1/\tilde{x}]},\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : P}\ \textsc{Dsply}$$

$$\frac{\{\pi\}_{\gamma\circ f} \qquad [\![\tilde{e}]\!]_\gamma = \tilde{d}_2}{\langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : \mathsf{Supply}(\tilde{x},\tilde{e})@(\mathsf{any})\mathsf{U}; P \xmapsto{\tilde{d}_1,\tilde{d}_2@(\mathsf{any},\pi)^{\mathsf{S}}} \langle \gamma_{\leftarrow \mathsf{U}[\tilde{d}_1/\tilde{x}]},\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : P}\ \textsc{Nsply}$$

$$\frac{\langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : P_1 \xmapsto{\lambda} \langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : P_1'}{\langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : P_1 + P_2 \xmapsto{\lambda} \langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : P_1'}\ \textsc{sumL} \qquad \frac{\langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : P_2 \xmapsto{\lambda} \langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : P_2'}{\langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : P_1 + P_2 \xmapsto{\lambda} \langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : P_2'}\ \textsc{sumR}$$

$$\frac{\gamma \vDash \pi[\lambda[\tilde{d}]/\tilde{x}] \qquad \langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : P \xmapsto{\lambda} \langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : P'}{\langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : \{\pi(\tilde{x})\}P \xmapsto{\lambda} \langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : P'}\ \textsc{Guard}$$

$$\frac{\langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : P[\mathsf{rep\ X.}\ P/\mathsf{X}] \xmapsto{\lambda} \langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : P'}{\langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : \mathsf{rep\ X.}\ P \xmapsto{\lambda} \langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : P'}\ \textsc{rep}$$

Agent A_1 from Example 3 has an agent level transition

$$\Gamma_0^1 : P_1^g \xmapsto{\text{“}\mathsf{A_1}\text{”},\mathsf{e}@(\text{“}\mathsf{A_2}\text{”},\mathsf{true})^{\mathsf{G}}} \Gamma_1^1 : X$$

where Γ_0^1 is as before, $\Gamma_1^1 = \langle(\mathsf{cLink} \mapsto \mathsf{e}, \mathsf{role} \mapsto \mathsf{client}), ch = \star, \text{“}\mathsf{A_1}\text{”}, pv \mapsto \mathsf{role}\rangle$, and P_1^g is given in Example 3. In fact, A_1 has such a transition for every possible value of channel replacing e above. Notice that this time the configuration of A_1 is changed by this transition by storing e in cLink.

A supply action is defined by the rules Dsply and Nsply. The rule Dsply is used for a deterministic point-to-point selection where the getter targets the supplier by its locality. In this case, only the targeted supplier is permitted to participate. On the other hand, the rule Nsply is used for anonymous nondeterministic selection where any supplier that satisfies the getter predicate it satisfies the sender predicate (up to relabelling, i.e., the closure $\{\pi\}_{\gamma\circ f}$ holds) may participate. In both rules, the selected supplier is able to both receive data $\tilde{d}_1$ and supply data $\tilde{d}_2$, and may perform store updates based on the received data. The last four rules are standard for modelling nondeterministic, guarded, and recursive behaviours.

Agent A_2 from Example 3 has the agent-level supply transition

$$\Gamma_0^2 : P_2^{su} \xmapsto{\text{“}\mathsf{A_1}\text{”},\mathsf{e}@(\text{“}\mathsf{A_2}\text{”},\mathsf{true})^{\mathsf{S}}} \Gamma_0^2 : X,$$

where Γ_0^2 is as before and P_2^{su} is given in Example 3. As before, a version of this transition for every possible identity replacing “$\mathsf{A_1}$” above will also be available.

Finally, according to rules SumL and SumR, both transitions for A_1 are available from $\Gamma_0^1 : (P_1^s + P_1^g)$ and both transitions for A_2 are available from $\Gamma_0^2 : (P_2^r + P_2^{su})$.

The negative semantics of an R-CHECK agent is reported in Table 3. These rules specify the cases when an agent cannot receive a broadcast or a multicast message. They distinguish broadcast from multicast in the sense that the former cannot be blocked while the latter can be blocked.

It is worth noting that the negative semantics could be abstracted by a predicate specifying when an agent is unable to receive a message. However, adopting such an abstraction would conflict with our objective of providing a fully compositional semantics. While negative semantics are clearly inefficient for implementation or automated reasoning—since

Table 3 Negative Process Semantics

$$\frac{(c \neq \star) \implies (\neg\{\mathsf{g}^{\mathsf{r}}[\mathsf{c}/\mathsf{ch}]\}_{\gamma})}{\Gamma{:}\mathsf{g}!\mathsf{c}'(\tilde{e})\mathsf{U};P \xrightarrow{(\pi?(\tilde{d})^{\mathsf{c}})\not\in} \Gamma{:}\mathsf{g}!\mathsf{c}'(\tilde{e})\mathsf{U};P}\ \textsc{Dsnd} \qquad \frac{(c \neq \star) \implies (\neg\{\mathsf{g}^{\mathsf{r}}[\mathsf{c}/\mathsf{ch}]\}_{\gamma})}{\Gamma{:}\mathsf{Get}(\tilde{x},\tilde{e})@(\ell,\mathsf{g})\mathsf{U};P \xrightarrow{(\pi?(\tilde{d})^{\mathsf{c}})\not\in} \Gamma{:}\mathsf{Get}(\tilde{x},\tilde{e})@(\ell,\mathsf{g})\mathsf{U};P}\ \textsc{Dget}$$

$$\frac{(c = \star) \implies \neg(\{\pi\}_{\gamma\circ f})}{\Gamma{:}(\tilde{x})\mathsf{c}?\mathsf{U};P \xrightarrow{(\pi?(\tilde{d})^{\star})\not\in} \Gamma{:}(\tilde{x})\mathsf{c}?\mathsf{U};P}\ \textsc{Dbrd} \qquad \frac{(c \neq \star) \implies (\neg\{\mathsf{g}^{\mathsf{r}}[\mathsf{c}/\mathsf{ch}]\}_{\gamma})}{\Gamma{:}\mathsf{Supply}(\tilde{x},\tilde{e})@(\ell)\mathsf{U};P \xrightarrow{(\pi?(\tilde{d})^{\mathsf{c}})\not\in} \Gamma{:}\mathsf{Supply}(\tilde{x},\tilde{e})@(\ell)\mathsf{U};P}\ \textsc{Dsply}$$

$$\frac{\mathsf{c} \neq \star \qquad (\neg\{\mathsf{g}^{\mathsf{r}}[\mathsf{c}/\mathsf{ch}]\}_{\gamma})}{\Gamma{:}(\tilde{x})\mathsf{c}'?\mathsf{U};P \xrightarrow{(\pi?(\tilde{d})^{\mathsf{c}})\not\in} \Gamma{:}(\tilde{x})\mathsf{c}'?\mathsf{U};P}\ \textsc{Dmst} \qquad \frac{(\gamma \not\models \pi[\tilde{d}/\tilde{x}] \ \wedge \ c = \star) \ \vee \ \Gamma{:}P \xrightarrow{(\pi'?(\tilde{d})^{\mathsf{c}})\not\in} \Gamma{:}P}{\Gamma{:}\{\pi(\tilde{x})\}P \xrightarrow{(\pi'?(\tilde{d})^{\mathsf{c}})\not\in} \Gamma{:}\{\pi(\tilde{x})\}P}\ \textsc{Dgrd}$$

$$\frac{\Gamma{:}P_1 \xrightarrow{(\pi?(\tilde{d})^{\mathsf{c}})\not\in} \Gamma{:}P_1 \quad \Gamma{:}P_2 \xrightarrow{(\pi?(\tilde{d})^{\mathsf{c}})\not\in} \Gamma{:}P_2}{\Gamma{:}P_1 + P_2 \xrightarrow{(\pi?(\tilde{d})^{\mathsf{c}})\not\in} \Gamma{:}P_1 + P_2}\ \textsc{Dsum} \qquad \frac{\Gamma{:}P[\mathsf{rep\ X}.\ P/\mathsf{X}] \xrightarrow{(\pi?(\tilde{d})^{\mathsf{c}})\not\in} \Gamma{:}P}{\Gamma{:}\mathsf{rep\ X}.\ P \xrightarrow{(\pi?(\tilde{d})^{\mathsf{c}})\not\in} \Gamma{:}\mathsf{rep\ X}.\ P}\ \textsc{rep}$$

$$\frac{}{\Gamma{:}0 \xrightarrow{(\pi?(\tilde{d})^{\mathsf{c}})\not\in} \Gamma{:}0}\ \textsc{Dnil}$$

they introduce a large number of transitions in the underlying LTS—they are nevertheless more intuitive. In particular, negative semantics, and compositional semantics more generally, make proofs and inductive arguments more explicit, which directly aligns with our goal of defining a compositional framework.

For efficient implementation, we therefore additionally provide an equivalent symbolic predicate semantics, closely aligned with a BDD-based representation, thereby preserving efficiency. The intended methodology is to use the compositional semantics for constructing and reasoning about systems in a modular fashion, while relying on the equivalent symbolic predicate semantics for efficient automated analysis. Accordingly, our model checker does not operate on the enumerative LTS, but on its predicate-based encoding.

Rule Dsnd in Table 2 specifies that an agent which can only send in its current state has the ability to discard and stay unchanged if the message is a broadcast (i.e., $\mathsf{c} = \star$) or it is not listening to the channel (i.e., $\neg\{\mathsf{g}^{\mathsf{r}}[\mathsf{ch}/\mathsf{c}]\}_{\gamma}$ is satisfiable). Notice the implication $\implies$ in the premise. We used a negative implication because it is more intuitive to understand in this context.

Similarly, a getter and a supplier can discard a message if it is a broadcast or if they are not listening to the channel as specified in rules Dget and Dsply.

The former rules state that unlike the broadcast, a multicast can be blocked if the receiver is listening to the channel but is not able to supply a matching transition. This is to denote that the receiver can block a multicast message until it is ready to participate. This kind of behaviour is apparent in existing programming languages through barrier synchronisation for instance.

A receiver agent, on the other hand, can discard a message only in two cases, specified by rules Dbrd and Dmst. The former states that a receiver can discard a broadcast if it is either not currently ready to receive a broadcast (i.e., does not have a matching broadcast transition $c \neq \star$) or it does not satisfy the sender predicate it satisfies the sender predicate (up to relabelling, i.e., the closure $\{\pi\}_{\gamma\circ f}$ does not hold). The latter rule, on the other hand, specifies when a receiver can discard a multicast. The only way a receiver can discard a multicast is if it is not listening to its channel.

A guarded agent (rule Dgrd) can discard a broadcast if its local predicate $\pi(\tilde{x})$ (possibly after substitution to message data $\pi[\lambda[\tilde{d}]/\tilde{x}]$) is not satisfied or its process can discard. However, a guarded agent can only discard a multicast if its process can discard, and thus it is not enough to not satisfying the local predicate.

The rules for nondeterminism and recursive calls are standard. Namely, such agents can discard if all parts of their behaviour can discard. Note that the deadlocked agent in rule Dnil is special. Here, we allow this agent to discard any message even without checking the receiver predicate. The latter is important to ensure that the parallel composition is idempotent; as otherwise a process can block the whole system when it is blocked. In practice, all R-CHECK processes are non-terminating processes, i.e., all our agents are of the form $\Gamma{:}\mathsf{rep\ X}.\ P$.

Notice that get actions cannot be discarded, and thus are blocking in nature.

5.2 System-level semantics

We use the transition relation $\longrightarrow \subseteq \mathsf{Sys} \times \mathsf{LAB} \times \mathsf{Sys}$ to define the behaviour of a system. This relation builds on the

Table 4 System Semantics

$$\frac{\langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : P \overset{\lambda}{\longmapsto} \langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : P'}{\langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : P \xrightarrow{\lambda} \langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : P'}\ \textsc{pSys} \qquad \frac{\langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : P \overset{(\pi?(\tilde{d})^{\mathsf{c}})\not\downarrow}{\longmapsto} \langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : P}{\langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : P \xrightarrow{\pi?(\tilde{d})^{\mathsf{c}}} \langle \gamma,\ \mathsf{g}^{\mathsf{r}},\ \mathsf{id},\ f\rangle : P}\ \textsc{nSys}$$

$$\frac{S_1 \xrightarrow{\pi?(\tilde{d})^{\mathsf{c}}} S_1' \quad S_2 \xrightarrow{\pi?(\tilde{d})^{\mathsf{c}}} S_2'}{S_1 \parallel S_2 \xrightarrow{\pi?(\tilde{d})^{\mathsf{c}}} S_1' \parallel S_2'}\ \textsc{Sync}$$

$$\frac{S_1 \xrightarrow{\pi!(\tilde{d})^{\mathsf{c}}} S_1' \quad S_2 \xrightarrow{\pi?(\tilde{d})^{\mathsf{c}}} S_2'}{S_1 \parallel S_2 \xrightarrow{\pi!(\tilde{d})^{\mathsf{c}}} S_1' \parallel S_2'}\ \textsc{ComL} \qquad \frac{S_1 \xrightarrow{\pi?(\tilde{d})^{\mathsf{c}}} S_1' \quad S_2 \xrightarrow{\pi!(\tilde{d})^{\mathsf{c}}} S_2'}{S_1 \parallel S_2 \xrightarrow{\pi!(\tilde{d})^{\mathsf{c}}} S_1' \parallel S_2'}\ \textsc{ComR}$$

$$\frac{S_1 \xrightarrow{\tilde{d}_1,\tilde{d}_2@(\ell,\pi)^{\mathsf{G}}} S_1'}{S_1 \parallel S_2 \xrightarrow{\tilde{d}_1,\tilde{d}_2@(\ell,\pi)^{\mathsf{G}}} S_1' \parallel S_2}\ \textsc{getL} \qquad \frac{S_2 \xrightarrow{\tilde{d}_1,\tilde{d}_2@(\ell,\pi)^{\mathsf{G}}} S_2'}{S_1 \parallel S_2 \xrightarrow{\tilde{d}_1,\tilde{d}_2@(\ell,\pi)^{\mathsf{G}}} S_1 \parallel S_2'}\ \textsc{getR}$$

$$\frac{S_1 \xrightarrow{\tilde{d}_1,\tilde{d}_2@(\ell,\pi)^{\mathsf{S}}} S_1'}{S_1 \parallel S_2 \xrightarrow{\tilde{d}_1,\tilde{d}_2@(\ell,\pi)^{\mathsf{S}}} S_1' \parallel S_2}\ \textsc{supplyL} \qquad \frac{S_2 \xrightarrow{\tilde{d}_1,\tilde{d}_2@(\ell,\pi)^{\mathsf{S}}} S_2'}{S_1 \parallel S_2 \xrightarrow{\tilde{d}_1,\tilde{d}_2@(\ell,\pi)^{\mathsf{S}}} S_1 \parallel S_2'}\ \textsc{supplyR}$$

$$\frac{S_1 \xrightarrow{\tilde{d}_1,\tilde{d}_2@(\ell,\pi)^{\tau}} S_1'}{S_1 \parallel S_2 \xrightarrow{\tilde{d}_1,\tilde{d}_2@(\ell,\pi)^{\tau}} S_1' \parallel S_2}\ \textsc{tauL} \qquad \frac{S_2 \xrightarrow{\tilde{d}_1,\tilde{d}_2@(\ell,\pi)^{\tau}} S_2'}{S_1 \parallel S_2 \xrightarrow{\tilde{d}_1,\tilde{d}_2@(\ell,\pi)^{\tau}} S_1 \parallel S_2'}\ \textsc{tauR}$$

$$\frac{S_1 \xrightarrow{\tilde{d}_1,\tilde{d}_2@(\ell,\pi)^{\mathsf{G}}} S_1' \quad S_2 \xrightarrow{\tilde{d}_1,\tilde{d}_2@(\ell,\pi)^{\mathsf{S}}} S_2'}{S_1 \parallel S_2 \xrightarrow{\tilde{d}_1,\tilde{d}_2@(\ell,\pi)^{\tau}} S_1' \parallel S_2'}\ \textsc{p-pL} \qquad \frac{S_1 \xrightarrow{\tilde{d}_1,\tilde{d}_2@(\ell,\pi)^{\mathsf{S}}} S_1' \quad S_2 \xrightarrow{\tilde{d}_1,\tilde{d}_2@(\ell,\pi)^{\mathsf{G}}} S_2'}{S_1 \parallel S_2 \xrightarrow{\tilde{d}_1,\tilde{d}_2@(\ell,\pi)^{\tau}} S_1' \parallel S_2'}\ \textsc{p-pR}$$

agent-level transition relation to expose relevant agent behaviour at system level. The set of system labels LAB ranges over both positive agent-level labels, ranged by λ, and a special τ label to denote hidden (or internal) communication. Here, we use $\tilde{d}@(\ell,\pi)^{\tau}$ to denote such label. We could have used τ immediately without further details, but our choice is influenced by our need to model check this behaviour. Moreover, the semantics shows that such label cannot propagate in the system and interact with more than one agent.

The semantics of an R-CHECK system is reported in Table 4. Rule pSys states that an agent can expose a λ label if its internal process can as a result of local action execution. On the other hand, rule nSys states that a local discard cannot be observed at system level, and thus it is exposed as a message reception.

Accordingly, the agent-level transitions mentioned above are exposed here as part of Rule pSys.

Rule Sync states that a broadcast or a multicast message can propagate through the system and be received by multiple agents, and thus modelling group communication. Rule ComL (and its symmetrical rule ComR) specifies that two composed system can communicate on a broadcast or a multicast if one of them sends the message and the other receives it.

It follows from ComL that $\xrightarrow{(pv=\mathsf{client})!((\mathsf{join},\mathsf{c}))^{\star}}$ is a possible transition from $\Gamma_0^1 : (P_1^s + P_1^g) \parallel \Gamma_0^2 : (P_2^r + P_2^{su})$. Notice that this exposes the send in case additional processes were available they could also participate in this broadcast.

On the other hand, rule getL (and its symmetrical rule getR) specifies that a get-message does not propagate in the system, but rather interleave with other composed systems. This is also the case for a supply and an internal communication as specified in rules supplyL (and its symmetrical rule supplyR) and tauL (and its symmetrical rule tauR) respectively.

The last two rules specify the bidirectional point-to-point communication in our formalism. That is, p-pL (and its symmetrical rule p-pR) state that if a getter system agrees with a supplier system to perform bidirectional information exchange then both systems privately consume the exchanged data and evolve accordingly.

It follows from P-PL and P-PR that $\xrightarrow{\text{“A}_1\text{”,e@(“A}_2\text{”,true)}^{\tau}}$ is another possible transition from $\Gamma_0^1 : (P_1^s + P_1^g) \parallel \Gamma_0^2 : (P_2^r + P_2^{su})$. Notice that once the get and supply have been “consumed” by P-PL or P-PR the only option for other processes to interact is by not blocking this τ transition according to rules TAUL and TAUR.

In the following lemma, we show that parallel composition $\parallel$ is a commutative monoid. We use a standard strong bisimulation [27] to establish the properties of $\parallel$.

Lemma 1 ($\parallel$ is a commutative monoid)

- $\parallel$ *is commutative:* $S_1 \parallel S_2 \sim S_2 \parallel S_1$
- $\parallel$ *is associative:* $(S_1 \parallel S_2) \parallel S_3 \sim S_1 \parallel (S_2 \parallel S_3)$
- $\parallel$ *has an idempotent element:* $(S \parallel \Gamma{:}0) \sim S$

The proof of this lemma follows by conducting a case analysis on the transition relation $S \xrightarrow{\alpha} S'$ for every system-level label α.

In the future, we would like to prove that this semantics coincides with the symbolic closed one.

Consider system-level label l_i, system state s_i for all $i \geq 0$, an execution of an R-CHECK system is the infinite sequence $s_0, l_0, s_1, l_1, s_2 \ldots$ such that the transition $(s_i \xrightarrow{l_i} s_{i+1})$ is derivable from Table 4, and s_0 is the initial state.

6 Model checking LTOL with point-to-point formulas

To reason about R-CHECK systems, we have previously introduced LTOL [12], an extension of Linear Time Temporal logic (LTL) with the ability to refer and therefore reason about agents interactions using observation descriptors. See Appendix A for full exposition.

Here we augment LTOL observation descriptors to be able to refer to point-to-point communication, the full logic is here:

$$
\begin{array}{lll}
O & ::= & \mathsf{p2p} \mid \neg\mathsf{p2p} \mid \ell \mid \neg\ell \mid pv \mid \neg pv \mid ch \mid \neg ch \mid k \mid \neg k \\
 & & \mid d \mid \neg d \mid \bullet^{\exists} O \mid \bullet^{\forall} O \mid O \vee O \mid O \wedge O \\
\varphi & ::= & \mathit{true} \mid v \mid \neg v \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid X\varphi \\
 & & \mid \varphi \,\mathcal{U}\, \varphi \mid \varphi \,\mathcal{R}\, \varphi \mid \langle O \rangle \varphi \mid [O]\varphi
\end{array}
$$

The extension of LTOL is a straightforward adjustment of the model-checking algorithm in [12], where we introduce two additional propositions to account for the type of communication, the locality, and their respective semantic encoding. Namely, the proposition p2p denotes the type of the communication, ℓ denotes the targeted locality, pv is a property identifier, ch is a channel name (identifying the channel the current message is sent on), k is an agent identifier (indicating the agent initiating the current interaction), and d is a data variable (whose value is determined by the payload of the current message).

Note that φ is a classical LTL in negation normal form, with the next operator replaced by $\langle O \rangle \varphi$ and $[O]\varphi$, which are predicated by observation descriptors O. These are built from referring to the different parts of the message, the added point-to-point descriptors p2p, and their Boolean combinations. Send predicates (part of messages) are interpreted as sets of possible assignments to property identifiers pv. Thus, we include existential $\bullet^{\exists} O$ and universal $\bullet^{\forall} O$ quantifiers over these assignments. Other operators such as $X, \mathcal{U}$, and $\mathcal{R}$ are the standard LTL operators “next”, “until”, and “release”. We also derive the operators $F\varphi = \mathit{true}\,\mathcal{U}\,\varphi$ (“eventually”) and $G\varphi = \neg \mathit{true}\,\mathcal{R}\,\varphi$ (“globally”).

For the full semantics of LTOL see [12]. Here, we describe it informally and only introduce the formal semantics for the new atoms.

Recall that the transition relation of a RECIPE system relates a system state s to its successor s' given an exchanged tuple $t = (\ell, \mathbf{d}, k, \pi)$ in the point-to-point case, while other modes of interaction feature a message $m = (ch, \mathbf{d}, k, \pi)$. Thus, we interpret the modified LTOL formulas over a system computation ρ, a function from natural numbers $\mathbb{N}$ to $2^{\mathcal{V}} \times (M \cup T)$ where $\mathcal{V}$ is the set of state variable propositions, M is the set of messages, and T is the set of tuples. Note that given a point-to-point tuple $(\ell, \mathbf{d}, k, \pi)$, we cannot single out the part of data contributed by the getter or the setter separately. This is because when the connection is established such details are hidden from an external observer.

The satisfaction of ch, d, and k propositions depends whether they exist in the message m or not. However, a message satisfies the pv proposition, written $m \vDash pv$, iff for every assignment $c \vDash \pi$ we have $c \vDash pv$.

The interesting cases are when we quantify over pv, i.e., those of $\bullet^{\exists} O$ and $\bullet^{\forall} O$:

$m \vDash \bullet^{\exists} O$ **iff** there is an assignment $c \vDash \pi$ such that $(ch, d, k, \{c\}) \vDash O$

$m \vDash \bullet^{\forall} O$ **iff** for every assignment $c \vDash \pi$ it holds that $(ch, d, k, \{c\}) \vDash O$

$t \vDash \bullet^{\exists} O$ **iff** there is an assignment $c \vDash \pi$ such that $(\ell, d, k, \{c\}) \vDash O$

$t \vDash \bullet^{\forall} O$ **iff** for every assignment $c \vDash \pi$ it holds that $(\ell, d, k, \{c\}) \vDash O$

To generalise these definitions to the extended setting, we use l_i, called *communication payload*, to range over either a tuple t_i or a message m_i at time i. We replicate these definitions for communications payload, in the obvious way, with the same definitions, except that a ch is never satisfied for a point-to-point payload. Negation and Boolean combinations are dealt with in the standard way.

We give the formal semantics for the new propositions.

$$l \vDash \mathsf{p2p} \ \textbf{iff} \ l(\ell) \neq \bot \quad \text{and} \quad l \vDash \neg \mathsf{p2p} \ \textbf{iff} \ l(\ell) = \bot;$$
$$l \vDash \ell' \ \textbf{iff} \ l(\ell) = \ell' \quad \text{and} \quad l \vDash \neg \ell' \ \textbf{iff} \ l(\ell) \neq \ell;$$

Note $l(\ell) = \bot$ indicates the tuple is a message exchanged during non-point-to-point communication. Intuitively, a payload l satisfies a locality proposition ℓ' if its locality component ℓ equals ℓ' and does not satisfy ℓ' otherwise. The negative case also includes when l is a message, because $l(\ell)$ returns $\bot$ in that case. We assume that $\bot$ is different from all other localities. Moreover, a payload l satisfies p2p if and only if $l(\ell) = \bot$, namely l is a message. Note that we can use the keywords getter, supplier, sender to refer to the agent's locality that is responsible for exchange in R-CHECK, where the first two refer to p2p and the latter for either broadcast or multicast. The embedding of the descriptors for point-to-point to R-CHECK is done similar to [12].

The semantics of an LTOL formula φ is defined for a computation ρ at a time point i. We give semantics for formulas with observation descriptors, and other formulas are interpreted exactly as in LTL.

$$\rho_{\geq i} \vDash v \ \textbf{iff} \ s_i \vDash v \quad \text{and} \quad \rho_{\geq i} \vDash \neg v \ \textbf{iff} \ s_i \nvDash v;$$
$$\rho_{\geq i} \vDash \langle O \rangle \varphi \ \textbf{iff} \ l_i \vDash O \ \text{and} \ \rho_{\geq i+1} \vDash \varphi;$$
$$\rho_{\geq i} \vDash [O] \varphi \ \textbf{iff} \ l_i \vDash O \ \text{implies} \ \rho_{\geq i+1} \vDash \varphi.$$

The temporal formula $\langle O \rangle \varphi$ is satisfied on the computation ρ at point i if the payload l_i satisfies O *and* φ is satisfied on the suffix computation $\rho_{\geq i+1}$. On the other hand, the formula $[O]\varphi$ is satisfied on the computation ρ at point i if l_i satisfying O *implies* that φ is satisfied on the suffix computation $\rho_{\geq i+1}$.

7 The superiority of attributed point-to-point

In this section, we showcase the power of bidirectional attributed point-to-point communication. To do so, we discuss an improved version of the system described in [13], where we only considered omnidirectional flow of information. We also show this expressive power with respect to existing anonymous communication primitives like attributed broadcast. Indeed, despite the undeniable advantages of anonymous communication primitives such as attributed broadcast, they still suffer from serious modelling issues. This is more apparent when considering modelling under open-world assumption, which is the main motivation behind anonymous communication. The latter allows agents to interact while not being aware of the existence of each other. It also facilitates seamless introduction of agents at run-time (or dynamic creation) without disrupting the overall system behaviour (though this is currently not supported by REC1PE and R-CHECK).

Here, we consider the problem of designing protocols with deadlock freedom and guaranteed progress. By definition, a protocol imposes dependence relations among interacting agents where some agents provide services that other agents consume. The problem occurs when an agent anonymously requests for a service and later waits for a response that will never arrive, namely, when no provider exists. The agent gets deadlocked because it cannot determine whether the response is delayed or will never arrive.

We argue that our attributed point-to-point provides an elegant solution to this problem without compromising anonymity. To showcase this, we consider the scenario of stable allocation in content delivery networks that was modelled in the *AbC* calculus [8] which supports anonymous broadcast. The problem is about matching equally sized sets of clients and servers based on an order of preferences such that there are no client and server in different matchings that both would prefer each other rather than their current partners. We argue that the protocol cannot be guaranteed to progress by only relying on anonymous broadcast.

The protocol in [8] relies on an open-world assumption whereby new agents can join at any time. Due to anonymity and non-blocking of *AbC* broadcast, a client broadcasts a proposal for servers to form a pair and waits for a response. However, if the proposal is sent before any server instance is created, then the proposal will be lost and the client will deadlock waiting for a response. To overcome this, the protocol in [8] introduces a counter that starts counting for a sufficiently large threshold, before it times out and the client proposes again. However, it is possible (due to uncontrolled network delays) that in most executions a positive response is received after the threshold is reached. Thus, the protocol gets stuck at the stage of proposal and does not get to progress. Here we show how to simply fix this problem.

We consider that servers and clients use the following data variables in interaction ID, LNK, RT, and D, where ID carries a locality, RT carries the rating of a server, and D carries the demand of a client. A client uses the local variables rating, Partner, xPartner, and demand to control its behaviour, where "rating" stores the rating of current connected server, "Partner" and "xPartner" store the locality of current and previous connected server; "demand" can take "H" for high demand service and "L" for low demand service of the client.

A generic client's initial condition θ_{c} is:

$$\mathsf{rating} = \mathsf{Partner} = \mathsf{xPartner} = \bot,$$

specifying that the client is not connected to any server. We can later create different clients with different demands. The receive guard g_c^r is $(\mathsf{ch} = \star)$. That is, reception is always enabled on broadcast. Now, the behaviour of a client is reported

in the R-CHECK process below:

```
rep X. (
  ⟨rating ≠ "H" ∧ rating ≠ RT⟩Get((RT, ID), (id, demand))
    @any[rating := RT; xPartner := Partner; Partner := ID];
  [
    ⟨xPartner = "⊥"⟩X
    +
    ⟨xPartner ≠ "⊥"⟩Get(, )@xPartner[xPartner := "⊥"]; X
  ]
```

Intuitively, the client is either repeatedly trying to connect to a server when it is not yet paired to a high rating server (rating ≠ "H"). That is, the client uses a blocking get-command to establish connection to any server that enhances its situation. That is, it does not accept a server with rating similar to its own (rating ≠ RT). If interaction is possible, the client sets rating to the rating of the server, swaps its current partner with the new one. Moreover, the client sends its locality and current demand to the new server simultaneously. It also needs to disconnect by issuing a get-command targeting its previous partner if exists, as shown in the nondeterministic choice +.

Now, a server uses the local variables rating, Partner, demand to control its behaviour, where "rating" stores the server rating and all other are defined as before. A generic server's initial condition θ_s is:

$$\text{demand} = \text{Partner} = \bot,$$

specifying that the server is not connected to any client. We can later create different servers with different rating and private links. The receive guard g_s^r is the same as the client's one. Now, the behaviour of a server is reported in the R-CHECK process below:

```
rep X. (
  ⟨Partner = "⊥"⟩Supply((ID, D), (rating, id))
    @any[Partner = ID; demand := D]; X
    +
  ⟨Partner ≠ "⊥" ∧ demand ≠ D ∧ demand ≠ "L"⟩
    Supply((ID, D), (rating, id))@any
    [Partner = ID; demand := D]; X
    +
    ⟨true⟩Supply(, )@self
        [demand := Partner :="⊥"]; X
)
```

Similarly, the server is either willing to supply connection to a client or dissolve from current client (last two lines). In the former cases, if the server does not have a partner, it will accept any connection; otherwise, the server only accepts clients if its improves on its current assigned demand (i.e., optimal case for servers is to accept demands that are better than their current ones). In that case, it supplies a tuple containing its own rating and its locality. Moreover, it stores the identity and the demand of the client locally.

As opposed to the protocol written in *AbC* [8], this one is very simple and compact, and thus more amendable to formal verification. Moreover, progress towards stability and deadlock-freedom is guaranteed given that the number of the clients is equal to the number of servers, which is anyway the assumption in [8] and a necessary condition for stability.

Moreover, this model is much more compact and easier to understand than the one presented in the conference version [13]. Thanks to the bidirectional information flow support for anonymous point-to-point communication. In the conference version, only the supplier (the server) was able to transfer data to the getter (the client), and thus the client had to send its information in a subsequent multicast to the server. The latter checks, afterward, if it wants to stay connected; otherwise it disconnects. This was unavoidable because there was no other way to make the server and client agree. However, these added communications were just a noise and made the protocol much harder to understand. Now that we allow bidirectional flow of information, we can avoid all multicast transitions and establish the protocol merely on unicast, without compromising anonymity.

We can easily create an R-CHECK system and verify its behaviour as follows.

$$\begin{aligned}\text{system} = {} & \mathsf{P_C}(\text{client1}, \text{demand} = \text{“L”}) \parallel \mathsf{P_C}(\text{client2}, \text{demand} = \text{“H”}) \\ & \parallel \mathsf{P_C}(\text{client3}, \text{demand} = \text{“H”}) \parallel \mathsf{P_S}(\text{server}, \text{rating} = \text{“L”}) \\ & \parallel \mathsf{P_S}(\text{server}, \text{rating} = \text{“L”}) \parallel \mathsf{P_S}(\text{server}, \text{rating} = \text{“H”})\end{aligned}$$

Namely, we have 3 clients, one with low demands and two with high demands. We have also created 3 servers with only one high-rating profile. Now, we can use the following formulas to reason individually and collectively.

$$\bigwedge_{k \in P_C} (\text{Partner}_k = \bot) \implies G[\text{getter} = k \wedge \text{p2p}]F(\text{Partner}_k \neq \bot) \tag{1}$$

$$\mathsf{FG}\Big(\bigwedge_{k \in P_C} (\text{Partner}_k \iff \mathsf{X}(\text{Partner}_k)) \wedge \bigwedge_{j \in P_S} (\text{Partner}_j \iff \mathsf{X}(\text{Partner}_j))\Big) \tag{2}$$

Formula (1) specifies that, for any client k, if k is not paired and it attempts paring with a server, it must always eventually get paired. Formula (2) requires that the protocol converges after a while, i.e., all servers and clients stay connected to the same partners.

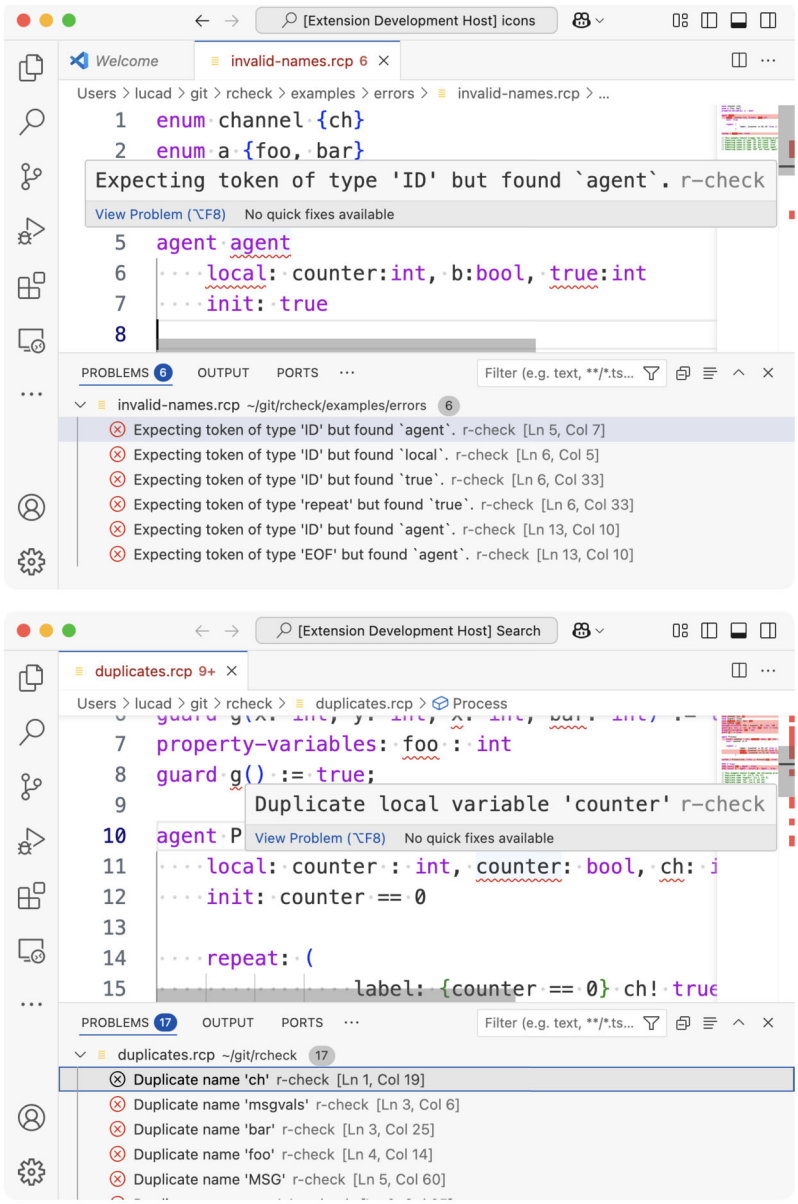

Fig. 1 Screenshots of Visual Studio Code with the R-CHECK extension enabled, showing the outcome of unsuccessful parsing and static checks, and type checking

8 A new R-CHECK implementation

Beyond extending R-CHECK with support for point-to-point communication, as exposed above, we also report on a new open-source implementation[1] of R-CHECK as a Visual Studio Code (VSCode) extension. The extension is written in TypeScript on top of the Langium language engineering framework.[2] Compared to the original implementation [12], it features an improved parser with more accurate error reporting, syntax highlighting, and keyword autocomplete. Furthermore, it lets the user jump to a variable declaration by clicking on any of its references while pressing the Control (or Command) key. Most of these features are automatically implemented by Langium, with very little additional effort on our part. We simply formalised the syntax of R-CHECK in a form of EBNF supported by the framework; this comes with the additional advantage that the EBNF is now the single source of truth for the grammar.

As part of the extension, we implement several additional static checks to validate specifications after successful parsing. For instance, duplicate variable names are now automatically detected and reported to the user. The extension also features a basic type checker, by which we can catch a number of errors in otherwise well-formed specifications (such as assignments to variables of mismatching type, or malformed LTOL observations) [30]. Figure 1 contains several

[1] Code and setup instructions are available at https://github.com/dsynma/rcheck.

[2] https://langium.org/.

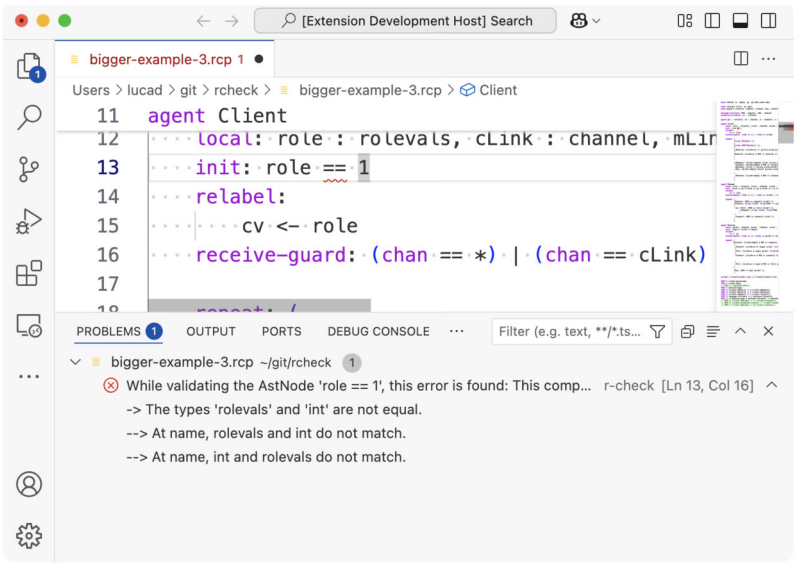

Fig. 2 The R-CHECK extension showing a type checking error

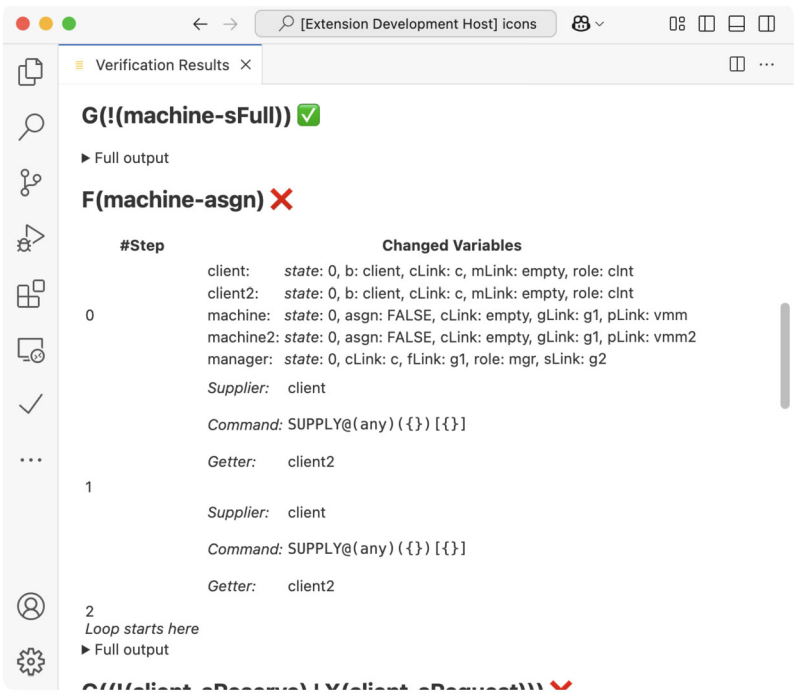

Fig. 3 The R-CHECK extension showing a model-checking report for the RECIPE example presented in [12]

screenshots that show the capabilities of our extension and how issues with a RECIPE specification are reported to the user. These issues can come from parsing, from unsuccessful static validation, or from breaking a type rule (Fig. 2). In any case, the user can see these outcomes both in the *Problems* pane and as red underlines in the document. By hovering on an underlined token, a tooltip appears that describes the problem in detail.

The extension relies on a command-line version of the existing R-CHECK implementation [12] to enable verification of RECIPE systems using IC3. Specifically, whenever the user wants to verify a system, the extension passes a serialized AST of the RECIPE file to the tool, which translates it into an SMV file. Then, this file is model-checked using NuXmv [17]. (The SMV translation may be inspected by running the command *R-CHECK: Show SMV Translation* from the VSCode command palette.) If a negative verdict is returned for any property, we once again rely on the command-line tool to translate the counterexample into a RECIPE execution trace, showing the states of agents and their interactions through messages. This polished counterexample is then shown to the user in a dedicated editor tab (Fig. 3). We

are currently working on integrating our existing interpreter to allow interactive simulations.

9 Related work

Point-to-point communication is a common communication mode systems use to exchange messages in a synchronous manner. The π-calculus [27] uses it as its only mode of communication, but only through reconfigurable channelled communication. There is a classical result that other modes, like channelled broadcast, cannot be encoded well in π-calculus, and vice versa channelled broadcast is not enough to encode channelled point-to-point [21]. The π-calculus does not support attribute-based communication either, unlike our approach.

As for attribute-based formalisms, we find approaches such as *AbC* [3, 4] and *AbU* [28], which (as discussed in Sect. 7) do not handle point-to-point communication and thus require encoding a protocol to enable this over multiple time steps. CARMA is an example of an attribute-based approach that handles both broadcast and unicast. It is a language for defining and reasoning quantitatively about collective adaptive systems [25]. Like R-CHECK [10, 11], they support attribute-based communication, so that communication can be established based on attributes. However, they do not support reconfiguration based on channels. Channels are statically known and cannot be passed at runtime. In CARMA, unicast is defined over channels, guarded by predicates over agent attributes (similar to our guards over pv). While R-CHECK allows for similar unicast-based on predicates, agents can also directly refer to the identity of an agent. Such localities can be encoded as attributes in CARMA; however, and unlike our approach, this does not guarantee that communication is safe from interference by other agents, since agents can modify their attributes maliciously.

Instead of channelled point-to-point communication, our approach supports purely attribute-based or locality-based communication. This allows for a level of anonymity: the supplier and getter do not need to know each other or know the proper channel to communicate on, while localities can be learned at runtime.

Another approach to modelling processes is that of *tuple spaces* (e.g., [19, 20]), dropping entirely channels and using solely localities. Here, agents do not communicate directly, but through retrieving and storing tuples in tuple spaces. In these approaches, tuple insertion cannot be blocked, and retrieval is based on predicates over the desired tuple. Our form of unicast cannot be modelled in these approaches, given its anonymous nature. Consider that relying on tuple spaces makes the communication indirect and asynchronous, while in our approach the communication is synchronous. SCEL [20] is an example of such an approach, allowing higher-order communication, since processes can be stored and retrieved in tuples. However, tuple spaces can grow arbitrarily large, which poses a challenge to model checking.

With regards to connector-based approaches such as BIP [16], the communication structure is defined a priori using a set of connectors allowing a wide variety of possible communication modes. This is an example of an *exogenous* coordination model, which enforces a clear separation between behavioural and communication concerns. Recent work on BIP introduced reconfigurable communication structures [14, 15]. In keeping with the exogenous model, reconfiguration decisions are not taken by the agents (components) but rather at a separate network layer. R-CHECK, in turn, clearly adopts an *endogenous* model where reconfiguration is carried out by the agents themselves in a dynamic, possibly opportunistic fashion.

10 Concluding remarks

We have augmented RECIPE and R-CHECK with bidirectional point-to-point communication as a primitive, beyond their original broadcast and multicast communication modes. Our focus is on raising the level of abstraction and the feasibility of design. The idea is that the objective of any modelling activity is to eventually reason about the design and verify its goals. Thus, we need to provide a high-level set of primitives that make modelling easier and produce models that are amenable to formal verification. We have argued how this new set of primitives enables better modelling of multi-agent systems, through an illustrative case study we can succinctly express in our extended language, but more challenging for existing languages. Previously, RECIPE could only encode point-to-point communication through a protocol of coordination on existing broadcast and multicast channels, allowing for interference. With the new primitives, point-to-point communication can be modelled in a way that preserves the integrity of the communication. We formally provided a compositional semantics that specifies how the different communication modes co-exist without interference and showed that they behave as expected. In the future, we would like to provide a proof of correspondence between the symbolic closed semantics and the compositional one.

We have also presented a new implementation of R-CHECK as an extension to the Visual Studio Code editor. This implementation improves the experience of language users by providing syntax highlighting and static checks to point out mistakes in the specifications, including a simple type checker for expressions and assignments. As future

work, we plan to improve this extension along several directions. We might extend our type system to perform more advanced static reasoning: for instance, we might type commands by the type of data they are willing to send or receive, and warn the user about potential deadlocks if no matching receive (or send) is present in the system. In time, we would like to port all features from the original R-CHECK implementation into the new one. This will require some effort, as the former contains approximately 10,000 lines of Java code.

We have recently initiated efforts to leverage R-CHECK for correct-by-construction code generation targeting ROS2 (the Robot Operating System). Our goal is to develop a formal implementation semantics for R-CHECK, building on approaches found in [1, 5, 6].

Appendix A: LTOL: an extension of LTL

LTOL is an extension of LTL with the ability to refer and therefore reason about agents interactions. It replaces the next operator of LTL with the observation descriptors: *possible* $\langle O \rangle$ and *necessary* $[O]$, to refer to messages and the intended set of receivers. The syntax of formulas φ and *observation descriptors* O is as follows:

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

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

The syntax of LTOL is presented in *positive normal form*. 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., $\overline{\varphi_1 \,\mathcal{U}\, \varphi_2} = \overline{\varphi_1} \,\mathcal{R}\, \overline{\varphi_2}$, $\overline{\varphi_1 \wedge \varphi_2} = \overline{\varphi_1} \vee \overline{\varphi_2}$, $\overline{pv} = \neg pv$, 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 property variables in PV. These predicates are interpreted as sets of possible assignments to property variables, and therefore we include existential $\bullet^{\exists} O$ and universal $\bullet^{\forall} O$ quantifiers over these assignments.

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

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

$m \vDash pv$ **iff** for every assignment $c \vDash \pi$ we have $c \vDash pv$
$m \vDash \neg pv$ **iff** exists an assignment $c \vDash \pi$ such that $c \nvDash pv$
$m \vDash \bullet^{\exists} O$ **iff** exists an assignment $c \vDash \pi$ and $(ch, d, k, \{c\}) \vDash O$
$m \vDash \bullet^{\forall} O$ **iff** for every assignment $c \vDash \pi$ implies $(ch, d, k, \{c\}) \vDash O$
$m \vDash O_1 \vee O_2$ **iff** either $m \vDash O_1$ or $m \vDash O_2$
$m \vDash O_1 \wedge O_2$ **iff** $m \vDash O_1$ and $m \vDash O_2$

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 $(ch, \mathbf{d}, k, pv_1 \vee \neg pv_2)$ and $(ch, \mathbf{d}, k, pv_1)$ satisfy $\bullet^{\exists} pv_1$ while only the latter satisfies $\bullet^{\forall} pv_1$. Furthermore, the observation descriptor $\bullet^{\forall}\mathsf{false} \wedge ch = \star$ says that a message is sent on the broadcast channel with a false predicate. That is, the message cannot be received by other agents. For example, the descriptor $\bullet^{\exists}(@\mathtt{type} = \mathtt{t1}) \wedge \bullet^{\forall}(@\mathtt{type} = \mathtt{t1})$ says that the message is intended exactly for agents of type-1.

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 pv and respectively $\neg pv$ appear outside the scope of a quantifier ($\bullet^{\forall}$ or $\bullet^{\exists}$), they are semantically equivalent to the descriptors $\bullet^{\forall} pv$ and respectively $\bullet^{\exists}\neg pv$. Thus, we assume that they are written in the latter normal form.

We interpret LTOL formulas over system computations:

Definition 4 (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 $M = \mathrm{CH} \times 2^{\mathrm{D}} \times K \times 2^{2^{\mathrm{PV}}}$ is the set of possible observations. That is, ρ includes values for the variables in $2^{\mathcal{V}}$ and an observation in M at each time instant.

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

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

$$\rho_{\geq i} \vDash v \ \textbf{iff}\ s_i \vDash v \quad \text{and} \quad \rho_{\geq i} \vDash \neg v \ \textbf{iff}\ s_i \nvDash v;$$
$$\rho_{\geq i} \vDash \varphi_2 \vee \varphi_2 \ \textbf{iff}\ \rho_{\geq i} \vDash \varphi_1 \ \text{or}\ \rho_{\geq i} \vDash \varphi_2;$$
$$\rho_{\geq i} \vDash \varphi_2 \wedge \varphi_2 \ \textbf{iff}\ \rho_{\geq i} \vDash \varphi_1 \ \text{and}\ \rho_{\geq i} \vDash \varphi_2;$$
$$\rho_{\geq i} \vDash \varphi_1 \,\mathcal{U}\, \varphi_2 \ \textbf{iff}\ \text{there exists } j \geq i \text{ s.t. } \rho_{\geq j} \vDash \varphi_2 \text{ and, for every } i \leq k < j,\ \rho_{\geq k} \vDash \varphi_1;$$
$$\rho_{\geq i} \vDash \varphi_1 \,\mathcal{R}\, \varphi_2 \ \textbf{iff}\ \text{for every } j \geq i \text{ either } \rho_{\geq j} \vDash \varphi_2 \text{ or, there exists } i \leq k < j, \rho_{\geq k} \vDash \varphi_1;$$
$$\rho_{\geq i} \vDash \langle O \rangle \varphi \ \textbf{iff}\ m_i \vDash O \ \text{and}\ \rho_{\geq i+1} \vDash \varphi;$$
$$\rho_{\geq i} \vDash [O] \varphi \ \textbf{iff}\ m_i \vDash O \ \text{implies}\ \rho_{\geq i+1} \vDash \varphi.$$

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

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.

Funding information Open access funding provided by University of Gothenburg.

References

1. Abd Alrahman, Y., Garbi, G.: A distributed API for coordinating AbC programs. Int. J. Softw. Tools Technol. Transf. (2020). https://doi.org/10.1007/s10009-020-00553-4
2. Abd Alrahman, Y., Piterman, N.: Modelling and verification of reconfigurable multi-agent systems. Auton. Agents Multi-Agent Syst. **35**(2), 47 (2021). https://doi.org/10.1007/s10458-021-09521-x
3. Abd Alrahman, Y., De Nicola, R., Loreti, M., Tiezzi, F., Vigo, R.: A calculus for attribute-based communication. In: Wainwright, R.L., Corchado, J.M., Bechini, A., Hong, J. (eds.) Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13–17, 2015, pp. 1840–1845. ACM, New York (2015). https://doi.org/10.1145/2695664.2695668
4. Abd Alrahman, Y., De Nicola, R., Loreti, M.: On the power of attribute-based communication. In: Albert, E., Lanese, I. (eds.) Formal Techniques for Distributed Objects, Components, and Systems - 36th IFIP WG 6.1 International Conference, FORTE 2016, Held as Part of the 11th International Federated Conference on Distributed Computing Techniques, DisCoTec 2016, Heraklion, Crete, Greece, June 6–9, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9688, pp. 1–18. Springer, Berlin (2016). https://doi.org/10.1007/978-3-319-39570-8_1
5. Abd Alrahman, Y., De Nicola, R., Garbi, G.: Goat: attribute-based interaction in Google go. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Distributed Systems - 8th International Symposium, ISoLA 2018, Limassol, Cyprus, November 5–9, 2018, Proceedings, Part III. Lecture Notes in Computer Science, vol. 11246, pp. 288–303. Springer, Berlin (2018). https://doi.org/10.1007/978-3-030-03424-5_19
6. Abd Alrahman, Y., De Nicola, R., Garbi, G., Loreti, M.: A distributed coordination infrastructure for attribute-based interaction. In: Baier, C., Caires, L. (eds.) Formal Techniques for Distributed Objects, Components, and Systems - 38th IFIP WG 6.1 International Conference, FORTE 2018, Held as Part of the 13th International Federated Conference on Distributed Computing Techniques, DisCoTec 2018, Madrid, Spain, June 18–21, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10854, pp. 1–20. Springer, Berlin (2018). https://doi.org/10.1007/978-3-319-92612-4_1
7. Abd Alrahman, Y., De Nicola, R., Loreti, M.: A calculus for collective-adaptive systems and its behavioural theory. Inf. Comput. **268** (2019). https://doi.org/10.1016/j.ic.2019.104457
8. Abd Alrahman, Y., De Nicola, R., Loreti, M.: Programming interactions in collective adaptive systems by relying on attribute-based communication. Sci. Comput. Program. **192**, 102428 (2020). https://doi.org/10.1016/j.scico.2020.102428
9. Abd Alrahman, Y., Perelli, G., Piterman, N.: Reconfigurable interaction for MAS modelling. In: Seghrouchni, A.E.F., Sukthankar, G., An, B., Yorke-Smith, N. (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 (2020). https://doi.org/10.5555/3398761.3398768
10. Abd Alrahman, Y., Azzopardi, S., Piterman, N.: Model checking reconfigurable interacting systems. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Adaptation and Learning - 11th International Symposium, ISoLA 2022, Rhodes, Greece, October 22–30, 2022, Proceedings, Part III. Lecture Notes in Computer Science, vol. 13703, pp. 373–389. Springer (2022). https://doi.org/10.1007/978-3-031-19759-8_23
11. Abd Alrahman, Y., Azzopardi, S., Piterman, N.: R-check: a model checker for verifying reconfigurable mas. In: Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems, AAMAS '22, pp. 1518–1520. International Foundation for Autonomous Agents and Multiagent Systems, Richland (2022). https://doi.org/10.5555/3535850.3536020
12. Abd Alrahman, Y., Azzopardi, S., Di Stefano, L., Piterman, N.: Language support for verifying reconfigurable interacting systems. Int. J. Softw. Tools Technol. Transf. **25**(5), 765–784 (2023). https://doi.org/10.1007/S10009-023-00729-8
13. Abd Alrahman, Y., Azzopardi, S., Di Stefano, L., Piterman, N.: Attributed point-to-point communication in R-CHECK. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Rigorous Engineering of Collective Adaptive Systems - 12th International Symposium, ISoLA 2024, Crete, Greece, October 27–31, 2024, Proceedings, Part II. Lecture Notes in Computer Science, vol. 15220, pp. 333–350. Springer (2024). https://doi.org/10.1007/978-3-031-75107-3_20
14. Ahrens, E., Bozga, M., Iosif, R., Katoen, J.P.: Reasoning about distributed reconfigurable systems. Proc. ACM Program. Lang. **6**(OOPSLA2), 145–174 (2022). https://doi.org/10.1145/3563293

15. Ballouli, R.E., Bensalem, S., Bozga, M., Sifakis, J.: Programming dynamic reconfigurable systems. Int. J. Softw. Tools Technol. Transf. **23**(5), 701–719 (2021). https://doi.org/10.1007/s10009-020-00596-7
16. Bliudze, S., Sifakis, J.: The algebra of connectors—structuring interaction in BIP. IEEE Trans. Comput. **57**(10), 1315–1330 (2008). https://doi.org/10.1109/TC.2008.26
17. Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuxmv symbolic model checker. In: Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18–22, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8559, pp. 334–342. Springer, Berlin (2014). https://doi.org/10.1007/978-3-319-08867-9_22
18. Cohen, P.R., Levesque, H.J.: Intention is choice with commitment. Artif. Intell. **42**(2–3), 213–261 (1990). https://doi.org/10.1016/0004-3702(90)90055-5
19. De Nicola, R., Gorla, D., Pugliese, R.: On the expressive power of KLAIM-based calculi. Theor. Comput. Sci. **356**(3), 387–421 (2006). https://doi.org/10.1016/J.TCS.2006.02.007
20. De Nicola, R., Latella, D., Lluch-Lafuente, A., Loreti, M., Margheri, A., Massink, M., Morichetta, A., Pugliese, R., Tiezzi, F., Vandin, A.: The SCEL language: design, implementation, verification. In: Wirsing, M., Hölzl, M.M., Koch, N., Mayer, P. (eds.) Software Engineering for Collective Autonomic Systems - the ASCENS Approach. Lecture Notes in Computer Science, vol. 8998, pp. 3–71. Springer, Berlin (2015). https://doi.org/10.1007/978-3-319-16310-9_1
21. Ene, C., Muntean, T.: Expressiveness of point-to-point versus broadcast communications. In: Ciobanu, G., Păun, G. (eds.) Fundamentals of Computation Theory, pp. 258–268. Springer, Berlin (1999)
22. Fagin, R., Halpern, J., Moses, Y., Vardi, M.Y.: Reasoning About Knowledge. MIT Press, Cambridge (1995)
23. Hannebauer, M.: Autonomous Dynamic Reconfiguration in Multi-Agent Systems, Improving the Quality and Efficiency of Collaborative Problem Solving. Lecture Notes in Computer Science, vol. 2427. Springer, Berlin (2002). https://doi.org/10.1007/3-540-45834-4
24. Huang, X., Chen, Q., Meng, J., Su, K.: Reconfigurability in reactive multiagent systems. In: Kambhampati, S. (ed.) Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9–15 July 2016, pp. 315–321. IJCAI/AAAI Press (2016). http://www.ijcai.org/Abstract/16/052
25. Loreti, M., Hillston, J.: Modelling and analysis of collective adaptive systems with CARMA and its tools. In: Bernardo, M., De Nicola, R., Hillston, J. (eds.) Formal Methods for the Quantitative Evaluation of Collective Adaptive Systems - 16th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2016, Bertinoro, Italy, June 20–24, 2016, Advanced Lectures. Lecture Notes in Computer Science, vol. 9700, pp. 83–119. Springer, Berlin (2016). https://doi.org/10.1007/978-3-319-34096-8_4
26. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, I. Inf. Comput. **100**(1), 1–40 (1992). https://doi.org/10.1016/0890-5401(92)90008-4
27. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, II. Inf. Comput. **100**(1), 41–77 (1992). https://doi.org/10.1016/0890-5401(92)90009-5
28. Pasqua, M., Miculan, M.: AbU: a calculus for distributed event-driven programming with attribute-based interaction. Theor. Comput. Sci. **958**, 113841 (2023). https://doi.org/10.1016/j.tcs.2023.113841
29. Piterman, N., Pnueli, A.: Temporal logic and fair discrete systems. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) Handbook of Model Checking, pp. 27–73. Springer, Berlin (2018). https://doi.org/10.1007/978-3-319-10575-8_2
30. Stolz, B.: Type checking a novel language for reconfigurable multi-agent systems. Bachelor's Thesis, TU Wien, Austria (2025). https://www.lucadistefano.eu/theses/bsc_stolz_2025.pdf
31. Wooldridge, M.J.: An Introduction to MultiAgent Systems, 2nd edn. Wiley, New York (2009)