Attributed Point-to-Point Communication in R-CHECK[⋆]

Yehia Abd Alrahman[1] [0000−0002−4866−6931], Shaun Azzopardi[2] [0000−0002−2165−3698], Luca Di Stefano[3] [0000−0003−1922−3151], and Nir Piterman[1] [0000−0002−8242−5357]

[1] University of Gothenburg and Chalmers University of Technology
[2] University of Malta
[3] TU Wien, Institute of Computer Engineering, Treitlstraße 3, 1040 Vienna, Austria

Abstract. Autonomous multi-agent, or more generally, collective adaptive 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. That is, the communication is not merely derived based on connectivity to channels but in addition based on properties of targeted receivers. 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, inefficient, and prone to interference.

Here, we extend R-CHECK with attributed point-to-point communication, which can be established based on identity or properties of participants. We also support model-checking of point-to-point by extending linear temporal logic with observation descriptors related to the participants in this communication mode. We argue that these extensions simplify the design of models, and demonstrate their benefits by means of an illustrative case study.

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 run-time based on changes in their context. We call such systems *Reconfigurable MAS* [15,14]. We are interested in designing such systems and, due to the

[⋆] 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).

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 [2,20,3], interactions [4], intentions [8], knowledge [13], and so forth. These descriptions are very 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 description of programs, actions, protocols, reconfiguration, self-organisation, etc.

We have previously presented ReCiPe [5,4] and R-CHECK [1], 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 [7].

One of the challenges of modeling 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, inefficient, 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. Here we extend ReCiPe and R-CHECK by supporting point-to-point communication.

Particularly, our new contributions are:

(i) we extend the theoretical model ReCiPe and its implementation in R-CHECK by point-to-point communication. Satisfying the spirit of the formalism, point-to-point communication is predicated on attributes, allowing to request information from suppliers based on their features. It is also possible to communicate based on known identities. As usual, messages have a payload but come with the assurance that the interaction has happened directly between two participants.

(ii) we extend LTOL with additional observation predicates allowing to analyze the contents of point-to-point communications. Modelers can distinguish between point-to-point and "cast" communication. They are also able to reason about the intentions of getters and suppliers.

This specialised integration provides a powerful tool that permits verifying high-level features of Reconfigurable MAS. Indeed, we can reason about systems both from an individual and a system level.

This article is structured as follows: in Sect. 2, we give a background on ReCiPe [5,4], the underlying theory of R-CHECK, on the modelling language of R-CHECK, and the specification language LTOL. In Sect. 3, we augment the language of R-CHECK with point-to-point communication and its symbolic semantics. In Sect. 4 we extend the LTOL logic to allow specification of point-to-point communication. In Sect. 5, we provide a nontrivial case study to model autonomous resource allocation. Finally, we report concluding remarks in Sect. 6.

2 Background Materials

We present background materials necessary to introduce our language extension, namely the ReCiPe formalism and the R-CHECK language.

2.1 The ReCiPe Formalism

ReCiPe [5,4] 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 [2,3]. 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 *property identifiers*. That is, identifiers that senders use to specify properties required of targeted receivers. The set of property identifiers is $\textsc{pv}$. For instance, agent k may specify that it wants to communicate on channel a with all agents that listen to a and satisfy the property $\mathsf{BatteryLevel} \geq 30\%$. In other words, property identifiers $\textsc{pv}$ are used by agents to indirectly specify constraints on the targeted receivers in a similar manner to the attribute-based paradigm [2,3].

However, each agent has a way to relate property identifiers to its local state through a re-labelling function. 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 re-labelling.

Formally, an agent is defined as a Discrete System (DS) [19]:

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.*
- $f : \textsc{pv} \to V$ *is a function, associating propriety identifiers to local variables.*
- $g^s(V, \textsc{ch}, \textsc{d}, \textsc{pv})$ *is a send guard specifying the property of the targeted receivers, based on the current evaluation of* V, $\textsc{ch}$, *and* $\textsc{d}$, *which is checked against every receiver* j *after applying* f_j.
- $g^r(V, \textsc{ch})$ *is a receive guard describing the connectedness of an agent to a channel ch. We let* $g^r(V, \star) = \mathsf{true}$, *i.e., every agent is always connected to the broadcast channel.*
- $\mathcal{T}^s(V, V', \textsc{d}, \textsc{ch})$ *and* $\mathcal{T}^r(V, V', \textsc{d}, \textsc{ch})$ *are assertions describing, respectively, the send and receive transition relations. We assume that an agent is broadcast input-enabled, i.e.,* $\forall v, \mathbf{d}\ \exists v'\ \mathsf{s.t.}\ \ \mathcal{T}^r(v, v', \mathbf{d}, \star)$.
- 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 $\textsc{d}$. We also abuse the notation and use f for the assertion $\bigwedge_{pv \in \textsc{pv}} pv = f(pv)$.

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. While receive transition relations $\mathcal{T}^r$ characterise the reaction of a receiving agent to a message.

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

A set of agents agreeing on property identifiers $\textsc{pv}$, data variables $\textsc{d}$, and channels $\textsc{ch}$ defines a *system.* We give the semantics of systems in terms of predicates to facilitate efficient symbolic analysis (through BDD or SMT). We use $\biguplus$ 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* $S = \langle \mathscr{V},\ \rho,\ \theta \rangle$, *where* $\mathscr{V} = \biguplus_i V_i$, *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 *of* S *is as follows:*

$$\begin{aligned}\rho =\ & \exists ch.\ \exists \textsc{d}.\ \bigvee_k \mathcal{T}^s_k(V_k, V_k', \textsc{d}, ch) \wedge \\ & \bigwedge_{j \neq k} \left(\exists \textsc{pv}.f_j \wedge \left(\begin{array}{l} g^r_j(V_j, ch) \wedge g^s_k(V_k, ch, \textsc{d}, \textsc{pv}) \wedge \\ \hfill \mathcal{T}^r_j(V_j, V_j', \textsc{d}, ch) \\ \vee \hfill \neg g^r_j(V_j, ch) \wedge \mathsf{Id}_j \\ \vee\ \neg g^s_k(V_k, ch, \textsc{d}, \textsc{pv}) \wedge ch = \star \wedge \mathsf{Id}_j \end{array} \right) \right)\end{aligned}$$

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 $\textsc{d}$) with assertion π (an assignment to g^s_k) 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 $\exists \textsc{pv}.f_j$) and connected to channel ch (i.e., $g^r_j(s^j, ch)$) 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., $\neg g^r_j(s^j, ch)$) 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 $\exists \textsc{pv}.f_j \wedge \pi$. 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.

2.2 The R-CHECK Language

ReCiPe is a low-level formalism that is geared towards efficient BDD representation and model-checking; and thus is not meant to be used as a modelling language. R-CHECK, in turn, is proposed to support high-level modelling of the ReCiPe formalism. It allows the user to model with high-level primitives, while hiding the underlying compilation to ReCiPe.

The syntax of R-CHECK is reported below.

$$\text{(System)} \qquad S ::= \quad A(id, \theta) \quad | \quad S_1 \| S_2$$

An agent A is the basic building block of an R-CHECK system. We treat agents as types where "A" can be instantiated as follows $A(id, \theta)$. That is, we create an instance of "A" with identity id and an initial condition θ. Moreover, each agent must define a receive predicate that specifies which channels the agent is connected to. We assume that all agents are always connected to broadcast channels, but they can choose to connect/disconnect multicast channels dynamically at run-time.

An R-CHECK system is a set of parallel agents. Here, we use the parallel composition operator $\|$ to inductively define a system.

That is, a system is either an instance of agent type or a parallel composition of set of instances of (possibly) different types. The semantics of $\|$ is fully captured by ρ in Def. 2. It should be noted that the $\|$ operator is used to build a closed system as in Def. 2, which is later used to give a system-level predicate semantics. That is, the semantics cannot be defined compositionally in terms of predicates as otherwise we would need to use higher-order predicates to capture the semantics, and thus inhibit efficient analysis. Recall that messages carry predicates that the receiver must quantify over in usual compositional semantics.

The syntax of an R-CHECK process is inductively defined as follows.

$$\begin{array}{lll} \text{(Process)} & P ::= & C; P \quad | \quad P + P \quad | \quad \mathsf{rep}\ P \quad | \quad C \\ \text{(Command)} & C ::= & l : C \quad | \quad \langle \Phi \rangle x!\ \pi\ \mathbf{d}\ \mathsf{U} \quad | \quad \langle \Phi \rangle x?\ \mathsf{U} \end{array}$$

An agent behaviour corresponds to an infinite repetition of a process P, denoted by $\mathsf{repeat} : P$. A process P is either a command prefix process $C; P$, a non-deterministic choice between two processes $P + P$, a loop $\mathsf{rep}\ P$, or a command C. There are three types of commands corresponding to either a labelled command, a message-send, or a message-receive. A command of the form $l{:}C$ is a syntactic labelling, only used to allow the model checker to reason about syntactic elements. A command of the form $\langle \Phi \rangle x!\ \pi\ \mathbf{d}\ \mathsf{U}$ corresponds to a message-send. Intuitively, the predicate Φ is an assertion over the current assignments to local variables, i.e., is a pre-condition that must hold for the transition to be taken; x is a placeholder (or a bound name) for a channel name. Note that x may refer to the value of a local variable, since we allow local variables to have the type `channels`. As the names suggest π and $\mathbf{d}$ are respectively the sender predicate, and the assignment to data variables (i.e., the actual content of the message). Lastly, U is the next assignment to local variables after taking the transition. We use "!" to distinguish send transitions. A command of the form $\langle \Phi \rangle x?\ \mathsf{U}$ corresponds to a message-receive. Differently from message-send, Φ can also predicate on the incoming message, i.e., the assignment $\mathbf{d}$. We use "?" to distinguish receive transitions. It is important to note that receive commands are evaluated based on the receive predicate mentioned above. That is, an agent cannot receive on a channel that does not satisfy its receive predicate.

The semantics of R-CHECK is given by translating 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 send and the receive commands of that agent. Moreover, both send and receive 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 send/receive transitions enabled in each step of a computation. For full exposition of the semantics, the reader is referred to [1].

3 Extending ReCiPe & R-CHECK with Attributed Point-to-Point Communication

We propose a Point-to-Point, or unicast, communication extension to R-CHECK. However, to be able to support such extension, we first need to extend the semantic framework, i.e., ReCiPe.

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 [17] or the tuple-space approach as in Klaim [10]. 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, implies 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 either based on satisfaction of a predicate g^p or based on a named locality ℓ. That is, an agent can ask for data from another agent by either supplying the name of the targeted agent (i.e., its locality ℓ) or predicating on the state of the potential supplier. 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). Moreover, a locality ℓ can be either a static name like an agent identity or the reserved word "any" to denote that any locality is allowed to supply. Formally, we extend Def. 1 as follows.

Definition 3 (P-to-P Agent). $A = \langle V,\ f,\ g^s,\ g^r,\ g^p,\ \mathcal{T}^s,\ \mathcal{T}^r, \mathcal{T}^{\mathsf{G}}, \mathcal{T}^{\mathsf{S}}, \theta\rangle$,

- $g^p(V, \textsc{pv})$ *is a get-guard specifying the property of the targeted supplier, based on the current evaluation of* V *of the getter and* $\textsc{pv}$ *which is checked against one supplier* j *after applying* f_j.
- $\mathcal{T}^{\mathsf{G}}(V, V', \mathbf{d}, \ell)$ *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.*
- $\mathcal{T}^{\mathsf{S}}(V, V', \mathbf{d}, \ell)$ *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 $\hat{\rho} = \rho \vee \rho_{gs}$ as reported below.

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

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 identity, and π is the getter-predicate, obtained by initially evaluating $\mathsf{g}^{\mathsf{p}}(V_k, \textsc{pv})$ over the getter local state. Namely, there exists an agent k that gets a tuple with data $\mathbf{d}$ (an assignment to $\textsc{d}$) with assertion π (an assignment to $\mathsf{g}^{\mathsf{p}}(V_k, \textsc{pv})$) 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 predicate. 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.

R-CHECK with Point-to-Point Communication. we are now ready to extend R-CHECK with point-to-point communication. Intuitively, we provide two commands for get and supply transitions, and specify the syntax of localities as reported below. Here, we use $\cdots$ to refer to the existing commands in R-CHECK.

$$\begin{array}{rcl} \text{(Command)}\ C ::= & \cdots \quad | \quad \langle \Phi \rangle \mathbf{Get}(\pi)@\ell\mathsf{U} \quad | \quad \langle \Phi \rangle \mathbf{Supply}@\ell\ \mathbf{d}\ \mathsf{U} \\ \text{(locality)}\ \ell ::= & k \quad | \quad \mathsf{self} \quad | \quad \mathsf{any} \quad | \quad \mathscr{B}(\ell) \end{array}$$

A command of the form $\langle \Phi \rangle \mathbf{Get}(\pi)@\ell\mathsf{U}$ corresponds to a tuple-get. Intuitively, the predicate Φ is an assertion over the current assignments to local variables for the getter, i.e., a pre-condition that must hold before the transition can be taken, π is the getter predicate, and ℓ is the locality of a potential supplier. Note that Φ may also have place holders (or bound names) for tuple/message variables to allow constraining the supplied data. As before, we use update U to denote the next assignment to local variables after taking the transition. Similarly, a command of the form $\langle \Phi \rangle \mathbf{Supply}@\ell\ \mathbf{d}\ \mathsf{U}$ corresponds to a tuple-supply transition. Here, the component $\mathbf{d}$ represents the actual data that the supplier is providing.

A locality ℓ can be a supplier identity k, a self reference that is evaluated to the identity of the agent, the keyword any which denotes that any supplier is accepted to participate in the interaction, or a boolean expression over a locality $\mathscr{B}(\ell)$. Notice in the new semantics of ReCiPe, 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. In case of missing getter predicate, we treat the getter predicate as a true predicate. Moreover, a getter predicate with $\ell \neq \mathsf{any}$ is always ignored.

The translation from the syntax of the new R-CHECK to ReCiPe is done exactly as in [1]. The idea is that we translate R-CHECK agents into symbolic automata, where the state of the automaton encodes the structure of an R-CHECK process and the label of a transition encodes the current enabled command.

4 Model Checking Point-to-Point LTOL Formulas

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

Here we augment LTOL observation descriptors to be able to refer to point-to-point communication, the full logic is here (new elements in blue):

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

where the proposition $\mathsf{p2p}$ denotes the type of the observation, 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 φ is 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, and their Boolean combinations. Send predicates (part of messages) are interpreted as sets of possible assignments to property identifiers. Thus we include existential $\bullet^{\exists} O$ and universal $\bullet^{\forall} O$ quantifiers over these assignments. Other operators such as $\mathcal{U}$ and $\mathcal{R}$ are standard "until" and "release" operators in LTL.

For the full semantics of LTOL see [1], here, due to lack of space, 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 $2^{\mathscr{V}} \times (M \cup T)$ where $\mathscr{V}$ is the set of state variable propositions, M is the set of messages, and T is the set of tuples.

The original semantics considers only channelled messages, e.g., $m \models pv$ iff for every assignment $c \models \pi$ we have $c \models pv$. Satisfaction of ch and k propositions depends whether they are in the tuple, similarly for d by comparing it to $\mathbf{d}$.

The more interesting cases are those of $\bullet^{\exists} O$ and $\bullet^{\forall} O$:

$$\begin{array}{lll} m \models \bullet^{\exists} O & \textbf{iff} & \text{there is an assignment } c \models \pi \text{ such that } (ch, d, k, \{c\}) \models O \\ m \models \bullet^{\forall} O & \textbf{iff} & \text{for every assignment } c \models \pi \text{ it holds that } (ch, d, k, \{c\}) \models O \end{array}$$

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.

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

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 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 [1].

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.

$$\begin{array}{l} \rho_{\geq i} \models v \ \textbf{ iff } \ s_i \models v \quad \text{and} \quad \rho_{\geq i} \models \neg v \ \textbf{ iff } \ s_i \not\models v; \\ \rho_{\geq i} \models \langle O \rangle \varphi \ \textbf{ iff } \ l_i \models O \ \text{ and } \ \rho_{\geq i+1} \models \varphi; \\ \rho_{\geq i} \models [O]\varphi \ \textbf{ iff } l_i \models O \ \text{ implies } \ \rho_{\geq i+1} \models \varphi. \end{array}$$

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 $\rho_{\geq i+1}$.

5 The Superiority of Attributed Point-to-Point

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 ReCiPe and R-CHECK).

Here, we consider the problem of designing protocols with deadlock freedom and guaranteed progress in open systems. 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 [3] which supports anonymous broadcast. The problem is about matching equally sized sets of clients and servers based on 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 [3] 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 instance of servers is created, then the proposal will be lost and the client will deadlock waiting for a response. To overcome this, the protocol in [3] 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, D, Act, where ID carries a locality, LNK carries a channel, RT carries the rating of a server, D carries the demand of a client, and Act carries an action name. A client uses the local variables rating, $\mathsf{Partner}$, $\mathsf{xPartner}$, lnk, demand to control its behaviour, where “rating” stores the rating of current connected server, “$\mathsf{Partner}$” and “$\mathsf{xPartner}$” store the locality of current and previous connected server; “lnk” stores a link that may be received; “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: $\mathsf{rating} = \mathsf{Partner} = \mathsf{xPartner} = \mathsf{lnk} = \bot$, 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 $(\mathsf{ch} = \star) \vee (\mathsf{ch} = \mathsf{lnk})$. That is, reception is always enabled on broadcast and on a channel that matches the value of lnk. Now, the behaviour of a client is reported in the R-CHECK process P_C below:

```
$P_C \triangleq$ **repeat**
    $\langle \mathsf{rating} \neq \text{“H”} \wedge \mathsf{rating} \neq \mathsf{RT} \rangle \mathbf{Get}(\mathsf{true})\mathbf{@any}$
        $[\mathsf{rating} := \mathsf{RT}; \mathsf{xPartner} := \mathsf{Partner}; \mathsf{Partner} := \mathsf{ID}; \mathsf{lnk} := \mathsf{LNK}];$

    [

      $\langle \mathsf{xPartner} = \text{“}\bot\text{”} \rangle lnk!(\mathsf{true})(\mathsf{ID} = \mathsf{id}, \mathsf{D} = \mathsf{demand})[\mathsf{lnk} := \bot]$

      $+$

      $\langle \mathsf{xPartner} \neq \text{“}\bot\text{”} \rangle \mathbf{Get}(\mathsf{true})\mathbf{@}\mathsf{xPartner}[\mathsf{xPartner} := \text{“}\bot\text{”}];$

      $lnk!(\mathsf{true})(\mathsf{ID} = \mathsf{id}, \mathsf{D} = \mathsf{demand})[\mathsf{lnk} := \bot]$

    ]

  $+ \langle \mathsf{true} \rangle \mathbf{Supply@self}(\mathsf{Act} = \text{“dissolve”})$
        $[\mathsf{rating} := \mathsf{Partner} := \mathsf{xPartner} := \mathsf{lnk} := \text{“}\bot\text{”}]$
```

Intuitively, the client is either repeatedly trying to connect to a server when it is not yet paired to a high rating server ($\mathsf{rating} \neq$ “H”) as in lines 2–3, or is ready to supply a dissolve to its current partner as in lines 10–11; notice the top-level nondeterministic choice $+$ at line 10. In the former case, 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 ($\mathsf{rating} \neq \mathsf{RT}$). If interaction is possible, the client sets rating to the rating of the server, swaps its current partner with the new one and stores the link communicated in the variable lnk. Afterwards, the client sends its information to the new server using the received link, but in case it has a previous connection, it also needs to disconnect by issuing a get-command targeting its previous partner, as shown in the first nondeterministic choice $+$.

Now, a server uses the local variables $\mathsf{rating},\ \mathsf{Partner},\ \mathsf{xPartner},\ \mathsf{lnk}, \mathsf{demand},\ \mathsf{pid}$ to control its behaviour, where “rating” stores the server rating, pid temporarily stores the locality of a potential partner, and all other are defined as before.

A generic server's initial condition θ_{s} is: $\mathsf{demand} = \mathsf{Partner} = \mathsf{xPartner} = \mathsf{pid} = \perp$, 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 P_S below:

```
$P_S \triangleq$ repeat
      $\langle \mathsf{demand} \neq \text{“L”} \rangle$ Supply@any($\mathsf{Act} = \text{“connect”}$,
                    $\mathsf{RT} = \mathsf{rating}, \mathsf{ID} = \mathsf{id}, \mathsf{LNK} = \mathsf{lnk}$)[];
      [
        $\langle \mathsf{Partner} = \text{“}\perp\text{”} \rangle lnk?[\mathsf{Partner} = \mathsf{ID}; \mathsf{demand} := \mathsf{D}]$
        $+$
        $\langle \mathsf{Partner} \neq \text{“}\perp\text{”} \wedge \mathsf{demand} = \mathsf{D} \rangle lnk?[\mathsf{pid} := \mathsf{ID}]$;
        Get(true)@pid[$\mathsf{pid} := \text{“}\perp\text{”}$]
        $+$
        $\langle \mathsf{Partner} \neq \text{“}\perp\text{”} \wedge \mathsf{demand} \neq \mathsf{D} \wedge \mathsf{D} \neq \text{“H”} \rangle lnk?$
                    $[\mathsf{Partner} = \mathsf{ID}; \mathsf{demand} := \mathsf{D}]$
      ]
    $+$ $\langle \mathsf{true} \rangle$Supply@self($\mathsf{Act} = \text{“dissolve”}$)
          $[\mathsf{rating} := \mathsf{Partner} := \mathsf{xPartner} := \mathsf{lnk} := \text{“}\perp\text{”}]$
```

Similarly, the server is either willing to supply connection (line 2-3) to a client or dissolve from current client (last two lines). In the former case, the server only accepts clients if its current assigned demand is not low “L” (i.e., optimal case for servers). In that case, it supplies a connect tuple, its own rating, locality, and a private link for further communication. Afterwards, the server decides if it should establish this connection. If the server does not have a partner, it will accept any connection. Thus, it stores the client locality and demand. If the server is connected, but the demands of the new client is the same of current one, the server has no incentive to connect, and thus issues a get command targeting the new client (through the locality stored in pid) to dissolve. The last case, if the new client improves the server's condition, the server accepts it. That is, if the demand of the new client is different from the current one, the server establishes the connection.

As opposed to the protocol written in *AbC* [3], this one is very simple and compact, and thus more amendable to formal verification. Moreover, progress towards stability and deadlock-freedom are guaranteed given that the number of the clients is equal to the number of server, which is anyway the assumption in [3] and a necessary condition for stability.

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

$$\begin{array}{l}\mathsf{system} = \mathsf{P_C}(\mathsf{client1}, \mathsf{demand} = \text{“L”}) \parallel \mathsf{P_C}(\mathsf{client2}, \mathsf{demand} = \text{“H”}) \parallel \mathsf{P_C}(\mathsf{client3}, \mathsf{demand} = \text{“H”}) \\ \quad \parallel \mathsf{P_S}(\mathsf{server}, \mathsf{rating} = \text{“L”}) \quad \parallel \mathsf{P_S}(\mathsf{server}, \mathsf{rating} = \text{“L”}) \quad \parallel \mathsf{P_S}(\mathsf{server}, \mathsf{rating} = \text{“H”})\end{array} \tag{1}$$

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.

$$\bigwedge_{\mathsf{k} \in \mathsf{P_C}} ((\mathsf{k} - \mathsf{Partner} = \bot) \rightarrow \mathsf{G}\,[\mathsf{getter} = \mathsf{k} \wedge \mathsf{p2p}]\mathsf{F}\,\langle \mathsf{sender} = \mathsf{k} \wedge \mathsf{ch} \neq \star \rangle \mathsf{true}) \qquad \text{(i)}$$

$$\mathsf{FG}(\bigwedge_{\mathsf{k} \in \mathsf{P_C}} (\mathsf{k} - \mathsf{Partner} \neq \bot) \wedge \bigwedge_{\mathsf{j} \in \mathsf{P_S}} (\mathsf{j} - \mathsf{Partner} \neq \bot)) \qquad \text{(ii)}$$

The first formula specifies that if any client is not paired then when it attempts paring with a server, it must always eventually initiates a non-broadcast communication with it. Notice that this formula impose an order such that p2p communication happens first.

The second formula ensures that after a while the protocol converges, and all servers and clients stay connected.

6 Concluding Remarks

We have augmented ReCiPe and R-CHECK with 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.

Related work Point-to-point communication is a common communication mode systems use to exchange messages in a synchronous manner. The π-calculus [18] 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 that vice-versa channelled broadcast is not enough to encode channelled point-to-point [12]. The π-calculus does not support attribute-based communication either, unlike our approach.

As for attribute-based formalisms, we find approaches such as *AbC* [9], which (as discussed in Sect. 5) do not handle point-to-point communication and thus requires 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 [16]. Like R-CHECK, 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, the 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., [10,11]), 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 [11] 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 [6], the communication structure is defined a priori using a set of connectors allowing a wide variety of possible communication modes. However, the structure is static and thus there is no reconfiguration as in R-CHECK.

References

1. 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, https://doi.org/10.1007/s10009-023-00729-8

2. 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
3. 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
4. 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
5. 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
6. Bliudze, S., Sifakis, J.: The algebra of connectors—structuring interaction in BIP. IEEE Transactions on Computers **57**(10), 1315–1330 (2008). https://doi.org/10.1109/TC.2008.26
7. 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 (2014). https://doi.org/10.1007/978-3-319-08867-9_22
8. 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
9. De Nicola, R., Duong, T., Inverso, O.: Verifying abc specifications via emulation. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation: Engineering Principles - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20-30, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12477, pp. 261–279. Springer (2020). https://doi.org/10.1007/978-3-030-61470-6_16, https://doi.org/10.1007/978-3-030-61470-6_16
10. 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
11. 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 (2015). https://doi.org/10.1007/978-3-319-16310-9_1, https://doi.org/10.1007/978-3-319-16310-9_1
12. 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 Heidelberg, Berlin, Heidelberg (1999)
13. Fagin, R., Halpern, J., Moses, Y., Vardi, M.Y.: Reasoning about Knowledge. MIT Press (1995)

14. 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 (2002). https://doi.org/10.1007/3-540-45834-4
15. 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
16. 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 (2016). https://doi.org/10.1007/978-3-319-34096-8_4, https://doi.org/10.1007/978-3-319-34096-8_4
17. 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
18. 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
19. 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 (2018). https://doi.org/10.1007/978-3-319-10575-8_2, https://doi.org/10.1007/978-3-319-10575-8_2
20. Wooldridge, M.J.: An Introduction to MultiAgent Systems, Second Edition. Wiley (2009)