R-CHECK: A Model Checker for Verifying Reconfigurable MAS

Extended Abstract

Yehia Abd Alrahman
University of Gothenburg
Gothenburg, Sweden

Shaun Azzopardi
University of Gothenburg
Gothenburg, Sweden

Nir Piterman
University of Gothenburg
Gothenburg, Sweden

ABSTRACT

Reconfigurable multi-agent systems consist of a set of autonomous agents, with integrated interaction capabilities that feature opportunistic interaction. Agents seemingly reconfigure their interactions interfaces by forming collectives, and interact based on mutual interests. Finding ways to design and analyse the behaviour of these systems is a vigorously pursued research goal. We propose a model checker, named R-CHECK, to allow reasoning about these systems both from an individual- and a system- level. R-CHECK also permits reasoning about interaction protocols and joint missions. R-CHECK supports a high-level input language with symbolic semantics, and provides a modelling convenience for interaction features such as reconfiguration, coalition formation, self-organisation, etc.

KEYWORDS

Model-checking; Agent Theories and Models; Verification of Multi-Agent Systems

ACM Reference Format:
Yehia Abd Alrahman, Shaun Azzopardi, and Nir Piterman. 2022. R-CHECK: A Model Checker for Verifying Reconfigurable MAS: Extended Abstract. In *Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), Online, May 9–13, 2022*, IFAAMAS, 3 pages.

1 OVERVIEW OF R-CHECK

R-CHECK[1] accepts a high-level language that is based on the symbolic ReCiPe formalism [2, 3]. We will present the syntax of the R-CHECK language and informally describe its semantics. For a full exposition of the formal definition of R-CHECK and its usage through sizeable case studies, we refer the reader to [4].

An R-CHECK system is composed of several agent instances that communicate by sending messages over a given set of channels. Common predicates, which agents interpret (differently) over their local states, are used to decide for whom messages are intended.

We first introduce the module `agent`, its structure, and how to instantiate it; we introduce the syntax of its behaviour and how to create a system of agents. The module `agent` is reported in Fig. 1.

Each module has a `name` that identifies a specific type of behaviour. We permit creating multiple agent instances with the same type of behaviour. An agent has a local state "`local`" represented by a set of local variables V_T, and interpretations of common predicates to interact with other agents anonymously. The initial state of an agent `init:` θ_T is a predicate characterising the initial assignments to the agent local variables. `receive-guard:` $g^r(V_T, \textsc{ch})$ specifies the connectedness of the agent to channels given a current assignment to its local variables. The non-terminating behaviour of an agent is represented by `repeat: P`, executing the process `P` indefinitely.

```
agent name
  local:
    var_name:type , ···, var_name:type
  init: θ_T
  relabel:
    common_pred <- Exp
      ⋮
    common_pred <- Exp
  receive-guard: g^r (V_T, CH)
  repeat: P
```

Figure 1: An agent module

An agent module of name "A" can be instantiated as follows $A(id, \theta)$. That is, we create an instance of "A" with identity id and an additional initial restriction θ. Here, we take the conjunction of θ with the predicate in the `init` section of the module "A" as the initial condition of this instance. We use the parallel composition operator $\|$ to inductively define a system as in the following production rule:

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

That is, a system is either an agent or a parallel composition (with reconfigurable multicast and broadcast semantics as in [2, 3]) of agents. Agents interact by *state-parametric* message exchange.

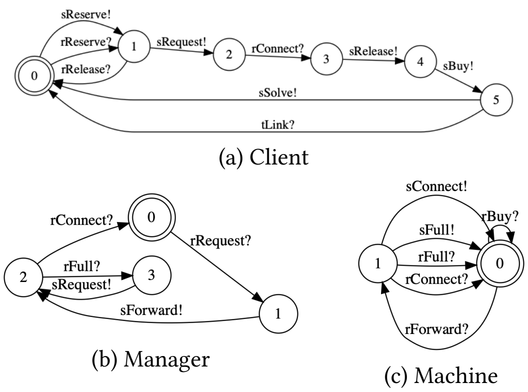
The syntax of an R-CHECK process is inductively defined as:

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

A process P is either a sequential composition of two processes $P; P$, a non-deterministic choice between two processes $P + P$, a loop rep P, or a command C. There are three types of commands: a labelled command, a message-send or a message-receive. A command of the form $l : C$ is a syntactic labelling and is used to allow the model checker to reason about syntactic elements as we will see later. A command of the form $\langle \Phi \rangle\ ch\ !\ \pi\ \mathbf{d}\ \mathsf{U}$ corresponds to a message-send. The predicate Φ is an assertion over the current local state of an agent, i.e., is a pre-condition that must hold before the transition can be taken. As the names suggest, ch, π and (respectively) $\mathbf{d}$ are the communication channel, the sender predicate (specifying the targeted receivers), and the assignment to data variables (i.e., the actual content of the message). Lastly, U is an update to local variables after taking the transition. We use "!" to distinguish send transitions. A command of the form $\langle \Phi \rangle\ ch\ ?\ \mathsf{U}$ corresponds to a message-receive. Differently from message-send, the predicate Φ can also predicate on the received values from the incoming message, i.e., the assignment $\mathbf{d}$.

[1] Find the associated toolkit repository here: https://github.com/dsynma/recipe.

Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), P. Faliszewski, V. Mascardi, C. Pelachaud, M.E. Taylor (eds.), May 9–13, 2022, Online.



(a) Client

(b) Manager

(c) Machine

Figure 2: Symbolic structure automata

For example, we can create an R-CHECK system as in Equation 1 below. There, a set of identical clients anonymously coordinate with a resource manager to acquire virtual machines (VM).

$$\begin{aligned} \text{system} = {} & \text{Client}(\text{client1}, \text{TRUE}) \parallel \text{Client}(\text{client2}, \text{TRUE}) \\ & \parallel \text{Client}(\text{client3}, \text{TRUE}) \parallel \text{Manager}(\text{manager}, \text{TRUE}) \\ & \parallel \text{Machine}(\text{machine1}, \text{gLink} = \text{g1} \wedge \text{pLink} = \text{vmm1}) \\ & \parallel \text{Machine}(\text{machine2}, \text{gLink} = \text{g1} \wedge \text{pLink} = \text{vmm2}) \\ & \parallel \text{Machine}(\text{machine3}, \text{gLink} = \text{g2} \wedge \text{pLink} = \text{vmm3}) \end{aligned} \quad (1)$$

Notice that the machines $\{\text{machine}_1, \ldots, \text{machine}_3\}$, each belongs to a specific group and uses a private link to interact. For instance, machine_1 belongs to group "g_1" (the high performance machines) and has a private link named "vmm1".

The symbolic behaviour of this system is reported in Fig. 2, where send transitions "!" synchronise with receive ones "?" having corresponding labels. The full example is reported in [4].

2 NUXMV AND MODEL-CHECKING

We integrate R-CHECK with the nuXmv model checker [6] to enable an enhanced symbolic LTL model-checking. We also demonstrate our developments using examples. We will show how the combined features of R-CHECK, the symbolic LTL model-checking, and nuXmv provides a powerful tool to verify high-level features of reconfigurable and interactive systems.

R-CHECK provides an interactive simulator that allows the user to simulate the system either randomly or based on predicates that the user supplies. For instance, one can refer to message -send and -receive using command labels. A constraint on a send transition like "client1−sReserve", to denote the sending of the message labelled with "sReserve", means that this transition is feasible in the current state of simulation. R-CHECK is also supported with a web editor, syntax highlighting and a visualising tool. For instance, once the model of Equation 1 is complied, R-CHECK produces the corresponding labelled and symbolic structure automata in Fig. 2, and thus the user may use these automata to reason about interactions.

Symbolic Model Checking. R-CHECK supports symbolic LTL model checking, bounded LTL model checking, and also the IC3 model checking algorithm [7]. We illustrate the capabilities of R-CHECK by several examples based on Equation 1, the automata in Fig. 2 as the system under consideration.

We show how to verify properties about agents both from individual and interaction protocols level by predicating on message exchange rather than on atomic propositions. It should be noted that the transition labels in Fig. 2 are not mere labels, but rather predicates with truth values changing dynamically at run-time, introducing opportunistic interaction. For instance, we can reason about a client and its connection to the system as follows:

$$\text{G}\ (\text{client1}{-}\text{sReserve} \rightarrow \text{F client1}{-}\text{sRelease}) \quad (1)$$
$$\text{G}\ (\text{client1}{-}\text{sRequest} \rightarrow \text{F client1}{-}\text{rConnect}) \quad (2)$$

The liveness condition (1) specifies that the client does not hold a live lock on a shared link. Namely, the client releases the shared link eventually. The liveness condition (2) specifies that the *system* is responsive, i.e., after the client's request, other agents collaborate to eventually supply a connection.

We can also reason about synchronisation and reconfiguration in relation to local state as in the following:

$$\text{G}\ (\text{manager}{-}\text{sForward} \rightarrow \text{X machine1}{-}\text{rForward}) \quad (3)$$
$$\text{F}\ (\text{client1}{-}\text{sRelease} \wedge \text{G}(!\text{client1}{-}\text{rConnect})) \quad (4)$$

In (3), we refer to synchronisation, i.e., the manager has to forward the request before the machine can receive it. We refer to reconfiguration in (4), i.e., eventually the client disconnects from the common link, and can never receive connections on that link.

We can also specify channel mobility and joint missions from a declarative and centralised point of view.

$$\left(\begin{array}{l} \text{F}(\text{client1}{-}\text{mLink} \neq \text{empty})\ \& \\ \qquad \text{F}\ (\text{client2}{-}\text{mLink} \neq \text{empty})\ \& \\ \qquad\qquad \text{F}\ (\text{client3}{-}\text{mLink} \neq \text{empty})) \longrightarrow \\ \text{F}\ (\text{client1}{-}\text{sSolve} \mid \text{client2}{-}\text{sSolve} \mid \text{client3}{-}\text{sSolve}) \end{array}\right)$$

That is, every client will eventually receive a mobile link (i.e., its $\text{mLink} \neq \text{empty}$) where it will use this private link to get a VM, and eventually one client will initiate the termination of the mission by synchronising with the other clients to solve the joint problem.

3 CONCLUDING REMARKS

We introduced the R-CHECK model checking toolkit for verifying and simulating reconfigurable multi-agent system. R-CHECK is supported with a command line tool, a web editor with syntax highlighting and visualisation. We integrated R-CHECK with nuXmv to enable LTL symbolic (bounded) model checking. We showed that this specialised integration provides a powerful tool that permits verifying high-level features such as interaction protocols, joint missions, channel mobility, reconfiguration, self-organisation, etc.

R-CHECK combines the lessons learnt from communication models like *AbC* [1, 5] and ReCiPe [2, 3], and mainstreams model checkers like MCMAS [13] which is based on *Interpreted Systems* [9], MTSA toolkit [8] (based on Hoare's CSP calculus [11] and *Fluent Linear Temporal logic (FLTL)* [10]), SPIN [12] (for protocol design). Furthermore, R-CHECK strives for expressiveness while preserving minimality and simplicity.

ACKNOWLEDGMENTS

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).

REFERENCES

- [1] Yehia Abd Alrahman, Rocco De Nicola, and Michele Loreti. 2019. A calculus for collective-adaptive systems and its behavioural theory. *Inf. Comput.* 268 (2019). https://doi.org/10.1016/j.ic.2019.104457
- [2] Yehia Abd Alrahman, Giuseppe Perelli, and Nir Piterman. 2020. Reconfigurable Interaction for MAS Modelling. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020*, Amal El Fallah Seghrouchni, Gita Sukthankar, Bo An, and Neil Yorke-Smith (Eds.). International Foundation for Autonomous Agents and Multiagent Systems, 7–15.
- [3] Yehia Abd Alrahman and Nir Piterman. 2021. Modelling and verification of reconfigurable multi-agent systems. *Auton. Agents Multi Agent Syst.* 35, 2 (2021), 47. https://doi.org/10.1007/s10458-021-09521-x
- [4] Yehia Abd Alrahman, Shaun Azzopardi, and Nir Piterman. 2022. R-CHECK: A Model Checker for Verifying Reconfigurable MAS. arXiv:2201.06312 [cs.LO]
- [5] Yehia Abd Alrahman, Rocco De Nicola, and Michele Loreti. 2020. Programming interactions in collective adaptive systems by relying on attribute-based communication. *Sci. Comput. Program.* 192 (2020), 102428. https://doi.org/10.1016/j.scico.2020.102428
- [6] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. 2014. The nuXmv Symbolic Model Checker. In *CAV (Lecture Notes in Computer Science, Vol. 8559)*, Armin Biere and Roderick Bloem (Eds.). Springer, 334–342.
- [7] Alessandro Cimatti and Alberto Griggio. 2012. Software Model Checking via IC3. In *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings (Lecture Notes in Computer Science, Vol. 7358)*, P. Madhusudan and Sanjit A. Seshia (Eds.). Springer, 277–293. https://doi.org/10.1007/978-3-642-31424-7_23
- [8] Nicolás D'Ippolito, Dario Fischbein, Marsha Chechik, and Sebastián Uchitel. 2008. MTSA: The Modal Transition System Analyser. In *23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008), 15-19 September 2008, L'Aquila, Italy*. IEEE Computer Society, 475–476. https://doi.org/10.1109/ASE.2008.78
- [9] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. 1995. *Reasoning about Knowledge.* MIT Press.
- [10] Dimitra Giannakopoulou and Jeff Magee. 2003. Fluent model checking for event-based systems. In *Proceedings of the 9th European software engineering and 11th ACM SIGSOFT international symposium on Foundations of software engineering* (Helsinki, Finland). ACM, 257–266.
- [11] C. A. R. Hoare. 2021. Communicating Sequential Processes. In *Theories of Programming: The Life and Works of Tony Hoare*, Cliff B. Jones and Jayadev Misra (Eds.). ACM / Morgan & Claypool, 157–186. https://doi.org/10.1145/3477355.3477364
- [12] Gerard J. Holzmann. 1997. The model checker SPIN. *IEEE Transactions on software engineering* 23, 5 (1997), 279–295.
- [13] Alessio Lomuscio, Hongyang Qu, and Franco Raimondi. 2017. MCMAS: an open-source model checker for the verification of multi-agent systems. *STTT* 19, 1 (2017), 9–30.