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:

1 OVERVIEW OF R-CHECK

R-CHECK accepts a high-level language that is based on the symbolic RClPs formalism [2, 3]. We will present the syntax of 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].

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

Each agent class has a name that identifies a specific type of behaviour; and uses a set of channels to interact with others. We permit creating multiple instances/copies with the same class of behaviour. An agent has a local state “local” represented by a set of local variables $V_T$, and a relabelling function to interact with other agents anonymously. The initial state of an agent $\text{init}: \theta_T$ is a predicate characterising the initial assignments to the agent local variables. The section $\text{receive-guard}: q'(V_T, ch)$ 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 $\text{repeat}: P$, executing the process $P$ indefinitely.

An agent type 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 $\theta$. Here, we take the conjunction of $\theta$ with the predicate in the $\text{init}$ section of the type “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:

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

That is, a system is either an instance of agent type or a parallel composition (with reconfigurable multicast and broadcast semantics as in [2, 3]) of set of instances of (possibly) different types. Agents interact by $\text{state-parametric}$ message exchange.

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

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

A process $P$ is either a sequential composition of two processes $P; P$, a non-deterministic choice between two processes $P + P$, a loop $\text{rep} P$, 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 $I : C$ 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 $\langle \Phi \rangle ch ! \pi d U$ corresponds to a message-send. The predicate $\Phi$ is an assertion over the current local state of an agent, i.e., is a precondition that must hold before the transition can be taken. As the names suggest, $ch$, $\pi$, and (respectively) $d$ 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 $\langle \Phi \rangle ch ? U$ corresponds to a message-receive. Differently from message-send, the predicate $\Phi$ can also predicate on the received values from the incoming message, i.e., the assignment $d$.

We can easily create an R-CHECK system as in Equation 1 below. There, a set of identical clients anonymously coordinate with a resource manager to get virtual machines (VM).
system = Client(client1, TRUE) || Client(client2, TRUE) 
  || Client(client3, TRUE) || Manager(manager, TRUE) 
  || Machine(machine1, gLink = g1 ∧ pLink = vmm1) (1) 
  || Machine(machine2, gLink = g1 ∧ pLink = vmm2) 
  || Machine(machine3, gLink = g2 ∧ pLink = vmm3)

Notice that the machines \{machine1, …, machines\}, each belongs to a specific group and uses a private link to interact. For instance, machine1 belongs to group “g1” (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 compiled, 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 both symbolic LTL model checking and bounded LTL model checking. 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:

\[
G (client1 \rightarrow sReserve \rightarrow F client1 \rightarrow sRelease) \quad (1)
\]

\[
G (client1 \rightarrow sRequest \rightarrow F client1 \rightarrow rConnect) \quad (2)
\]

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:

\[
G (manager \rightarrow sForward \rightarrow X machine1 \rightarrow rForward) \quad (3)
\]

\[
F (client1 \rightarrow sRelease \land G (\neg (client1 \rightarrow rConnect))) \quad (4)
\]

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

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

\[
F (client1 \rightarrow mLink \neq empty) \land
F (client2 \rightarrow mLink \neq empty) \land
F (client3 \rightarrow mLink \neq empty) \land
F (client1 \rightarrow sSolve | client2 \rightarrow sSolve | client3 \rightarrow sSolve)
\]

That is, every client will eventually receive a mobile link (i.e., its mLink \neq 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.

We are unaware of a model-checker that enables reasoning at such a high-level.

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 [12] which is based on Interpreted Systems [8], MTSA toolkit [7] (based on Hoare’s CSP calculus [10] and Fluent Linear Temporal logic (FLTL) [9]), SPIN [11] (for protocol design). Furthermore, R-CHECK strives for expressiveness while preserving minimality and simplicity.

Future works. We plan to integrate LTLOL to R-CHECK from [3]. Indeed, the authors in [3] provide a EXPSPACE algorithm for LTLOL model checking (improved from EXPSPACE in [2]). This way, we would not only be able to refer to message exchange in logical formulas, but also to identify the intentions of agents in the interaction and characterise potential interacting partners.
REFERENCES


