SCHOOL OF
COMPUTING

Title: Model Checking for Temporal-Epistemic Properties of Distributed Reaction Systems

Names: Artur Męski, Maciej Koutny, and Wojciech Penczek

TECHNICAL REPORT SERIES

No. CS-TR- 1526 April 2019

TECHNICAL REPORT SERIES

No. CS-TR- 1526 Date: April 2019

Title: Model Checking for Temporal-Epistemic Properties of Distributed Reaction Systems

Authors: Artur Męski, Maciej Koutny, and Wojciech Penczek

Abstract: Reaction systems are a model of computation inspired by the biochemistry exhibited by living cells. In this paper, we introduce the notion of agency as an extension to the reaction systems formalism, leading to distributed reaction systems. Adding agents in the reaction systems setting, allows for the natural modelling and representation of multiagent and distributed systems. To support the specification of temporalepistemic properties of distributed reaction systems, we introduce rsctlk and present its associated model checking procedure. Finally, we show that model checking for rsctlk is pspace-complete.

Printed and published by Newcastle University,
School of Computing, Urban Sciences Building,
Newcastle upon Tyne, NE4 5TG, England.

Bibliographical Details

Title and Authors: Model Checking for Temporal-Epistemic Properties of Distributed Reaction Systems.

Artur Męski, Maciej Koutny and Wojciech Penczek

NEWCASTLE UNIVERSITY
School of Computing. Technical Report Series. CS-TR- 1526

Abstract: Reaction systems are a model of computation inspired by the biochemistry exhibited by living cells. In this paper, we introduce the notion of agency as an extension to the reaction systems formalism, leading to distributed reaction systems. Adding agents in the reaction systems setting, allows for the natural modelling and representation of multi-agent and distributed systems. To support the specification of temporal epistemic properties of distributed reaction systems, we introduce rsCTLK and present its associated model checking procedure. Finally, we show that model checking for rsCTLK is PSPACE-complete.

About the authors: Artur Męski is a Ph.D. candidate in the Institute of Computer Science of Polish Academy of Sciences, under the supervision of Professor Wojciech Penczek. His research interests include formal verification methods for concurrent systems and unconventional models of computation.

Maciej Koutny is a Professor of Computing Science in the School of Computing, Newcastle University. His research interests centre on the theory of distributed and concurrent systems, including both theoretical aspects of their semantics and application of formal techniques to the modelling, synthesis, and verification of such systems. He has also been working on the development of a formal model combining Petri nets and process algebras as well as on Petri net based behavioural models of membrane systems and reaction systems.

Wojciech Penczek is Professor of Computer Science and Deputy Director for Research of the Institute of Computer Science PAS and Professor of Computer Science at the University of Natural Sciences and Humanities. His main research interests are models of distributed systems, multi-agent systems, modelling of knowledge and belief, temporal logics for concurrent systems, automated verification, planning, composition of web services, and model checking of concurrent systems.

Suggested keywords: reaction system, multi-agent system, symbolic model checking, modal logics.

Model Checking for Temporal-Epistemic Properties of Distributed Reaction Systems

Artur Męski[1,2], Maciej Koutny[3], and Wojciech Penczek[1,4]

[1] Institute of Computer Science, PAS, Poland
[2] Vector GB Limited, London, UK
[3] School of Computing, Newcastle University, Newcastle upon Tyne, UK
[4] University of Natural Sciences and Humanities, ICS, Siedlce, Poland

Abstract. Reaction systems are a model of computation inspired by the biochemistry exhibited by living cells. In this paper, we introduce the notion of agency as an extension to the reaction systems formalism, leading to *distributed* reaction systems. Adding agents in the reaction systems setting, allows for the natural modelling and representation of multi-agent and distributed systems. To support the specification of temporal-epistemic properties of distributed reaction systems, we introduce rsCTLK and present its associated model checking procedure. Finally, we show that model checking for rsCTLK is PSPACE-complete.

1 Introduction

Natural computing is a fast developing research field concerned with investigating models and computational techniques inspired by nature, as well as investigating, in terms of information processing, complex phenomena taking place in nature (cf. [18] and [33]). Within the sub-field of biomolecular computation, a particular strand dealing with the latter kind of investigation aims at establishing how biocomputations drive natural processes. A prominent formal model introduced in order to support such an endeavour are *reaction systems* (cf. [15, 9, 10]). They were proposed as a formal model of the functioning of the living cell, where this functioning is viewed in terms of formal processes resulting from interactions between biochemical reactions. Crucially, these interactions are driven by the mechanisms of facilitation and inhibition: the (products of the) reactions may facilitate or inhibit each other. The basic model of reaction systems is qualitative rather than quantitative. However, it takes into account the basic flow of energy of the living cell (cf. [22]), and that the behaviour of the living cell is influenced by its environment. Having originally been inspired by the behaviour of the living cell, recent research on reaction systems has been motivated by both biological considerations (cf. [5, 14, 16, 4, 8, 11]) and the general considerations concerned with the understanding of computations taking place in reaction systems (cf. [13, 12, 17, 9, 21, 34]). As a result, reaction systems are increasingly perceived as a novel general model of interactive computation.

There are two key advantages of reaction systems when it comes to the modelling and subsequent verification of real-life systems. One is the simplicity

of reactions, which is a direct result of threshold modelling applied to the representation of entities, and the resulting finiteness coming from set theoretical representations of states. The other is that the explicit representation of contexts and the resulting separation allows one to deal with complex behaviours using very simple and finite 'core' system model.

Verification of reaction systems is a very important area of research, discussed in, e.g., [2, 3, 25]. There are results showing model checking methods for branching time temporal properties specified in rsCTL [25] as well as for linear time temporal properties [23, 24]. Recently, we have defined an encoding of parametric reaction systems in SMT, and proposed a synthesis procedure based on bounded model checking for properties specified in linear time temporal logic [27].

In this paper, we introduce the notion of agency as an extension to the reaction systems formalism, leading to *distributed* reaction systems (DRS). Adding agents in the reaction systems setting, allows for the natural modelling and representation of multi-agent and distributed systems. By doing this we also bring together the two key advantages of reaction systems, and the practically significant ability of membrane systems (cf. [1, 19]) and tissue systems (cf. [29, 32, 30]) to model networks of cells. The introduced formalism extends the basic reaction systems model with the idea of compartmentisation derived from membrane and tissue systems that allows for modelling of distributed systems. In more detail, the paper extends the basic reaction systems to allow modelling of distributed systems, and different synchronisation schemes for concurrent systems. Crucially, it is possible to model systems employing both synchronous and asynchronous execution semantics, and an extended notion of context automata allowing conditional generation of contexts (based on the current state of a DRS). We demonstrate how the formalism of DRS can be applied to modelling of multi-agent systems.

To enable specifying temporal-epistemic properties of DRS we introduce a new logic for reaction systems: rsCTLK, which extends rsCTL (cf. [24]) with epistemic operators. Then, we introduce a model checking method for DRS and properties expressed in rsCTLK. The proposed method is implemented using binary decision diagrams for symbolic model checking and the method is evaluated experimentally. We prove that model checking for rsCTLK is PSPACE-complete.

The paper is organised as follows. In the next section we recall the basic notions and definitions used by reaction systems. Section 3 introduces distributed reaction systems and the following section defines rsCTLK. In Section 5 a model checking method for DRS and rsCTLK is introduced. Section 6 presents a Boolean encoding for DRS which is used for the implementation of the model checking method which we evaluate experimentally in Section 7. In the last section of this paper we draw some conclusions.

2 Preliminaries

A *reaction system* is a pair $\mathcal{R} = (S, A)$, where S is a finite *background* set and A is a set of *reactions* over the background set. Each reaction in A is a triple $b = (R, I, P)$ such that R, I, P are nonempty subsets of S with $R \cap I = \emptyset$.

The sets R, I, and P are respectively denoted by R_b, I_b, and P_b and called the *reactant*, *inhibitor*, and *product set* of reaction b.

A reaction $b \in A$ is *enabled* by $T \subseteq S$, denoted $en_b(T)$, if $R_b \subseteq T$ and $I_b \cap T = \emptyset$. The *result* of b on T is given by $res_b(T) = P_b$ if $en_b(T)$, and by $res_b(T) = \emptyset$ otherwise. Then the *result* of A on T is $res_A(T) = \bigcup\{res_b(T) \mid b \in A\} = \bigcup\{P_b \mid b \in A \text{ and } en_b(T)\}$.

Intuitively, T represents a state of a biochemical system being modelled by listing all present biochemical entities. A reaction b is enabled by T and can take place if all its reactants are present and none of its inhibitors is present in T.

Example 1. Let $(S, A) = (\{1, 2, 3, 4\}, \{a_1, a_2, a_3, a_4\})$ be a reaction system, where:

$$a_1 = (\{1, 4\}, \{2\}, \{1, 2\}) \quad a_2 = (\{2\}, \{3\}, \{1, 3, 4\})$$
$$a_3 = (\{1, 3\}, \{2\}, \{1, 2\}) \quad a_4 = (\{3\}, \{2\}, \{1\})$$

In state $T = \{1, 3, 4\}$ reactions a_1, a_3, and a_4 are enabled, while a_2 is not. Hence $res_A(T) = res_{a_1}(T) \cup res_{a_3}(T) \cup res_{a_4}(T) = \{1, 2\} \cup \{1, 2\} \cup \{1\} = \{1, 2\}$. □

Entities in reaction systems are *non-permanent*, i.e., if entity x is present in the successor state T' of a current state T then it must have been produced (sustained) by a reaction enabled by T (thus $x \in res_A(T)$). Also, there are no conflicts between reactions enabled by T.

A reaction system is a finite system in the sense that the size of each state is a priori limited (by the size of the background set), and the state transformations it describes are deterministic since there are no conflicts between enabled reactions. This changes once we decided to take account of the external environment which is necessary to reflect the fact that the living cell is an open system. Such an environment can be represented by a context automaton.

A *context automaton* over a finite set Ct, is a triple $\mathfrak{A} = (\mathcal{Q}, \mathfrak{q}^{init}, R)$, where $\mathcal{Q}$ is a finite set of *states*, $\mathfrak{q}^{init} \in \mathcal{Q}$ is the *initial state*, and $R \subseteq \mathcal{Q} \times Ct \times \mathcal{Q}$ is a *transition relation* labelled with elements of Ct.

A *context restricted reaction system* is a pair CR-$\mathcal{R}$ $= (\mathcal{R}, \mathfrak{A})$ such that $\mathcal{R} = (S, A)$ is a reaction system, and $\mathfrak{A} = (\mathcal{Q}, \mathfrak{q}^{init}, R)$ is a *context automaton* over 2^S. The dynamic behaviour of CR-$\mathcal{R}$ is then captured by the state sequences of its interactive processes. An *interactive process* in CR-$\mathcal{R}$ is $\pi = (\zeta, \gamma, \delta)$, where:

- $\zeta = (z_0, z_1, \ldots, z_n)$, $\gamma = (C_0, C_1, \ldots, C_n)$, and $\delta = (D_0, D_1, \ldots, D_n)$
- $z_0, z_1, \ldots, z_n \in Q$ with $z_0 = q_0$
- $C_0, C_1, \ldots, C_n, D_0, D_1, \ldots, D_n \subseteq S$ with $D_0 = \emptyset$
- $(z_i, C_i, z_{i+1}) \in R$, for every $i \in \{0, \ldots, n-1\}$
- $D_i = res_A(D_{i-1} \cup C_{i-1})$, for every $i \in \{1, \ldots, n\}$.

Then the *state sequence* of π is $\tau = (W_0, \ldots, W_n) = (C_0 \cup D_0, \ldots, C_n \cup D_n)$.

Intuitively, the state sequence of π captures the observed behaviour of CR-$\mathcal{R}$ by recording the successive states of the evolution of the reaction system rs in the environment represented by the context automaton $\mathfrak{A}$.

For dealing with tuples we introduce the following notation: if $x = (x_1, x_2, \ldots, x_n)$ is a tuple then $x[i] = x_i$, for every $i \leq n$.

3 Distributed reaction systems

A distributed reaction system models a system that consists of m *agents*. Each agent has its own set of reactions over a common background set.

Definition 1. *Let $\mathcal{A} = \{1, \ldots, m\}$ be a set of agents. A distributed reaction system (DRS) is a tuple $\mathcal{D} = (S, \{A_i\}_{i \in \mathcal{A}})$, where S is a finite nonempty set, and each A_i for $i \in \mathcal{A}$ is a subset of $rac(S)$.*

The set S is the background set of $\mathcal{D}$, and the reactions of A_i belong to the i^{th} agent. Each agent maintains its own local state, and the tuples of

$$St_{\mathcal{D}} = \underbrace{2^S \times \cdots \times 2^S}_{m \text{ times}}$$

are called the *states* of $\mathcal{D}$. Throughout this section, $\mathcal{A} = \{1, \ldots, m\}$ is a fixed set of agents.

At each transition from one global state to the next, the environment provides DRS with a context which, for each agent, 'throws in' a set of entities and, in addition, specifies which of the agents are to be active in the current transition. The contexts are defined as pairs $Ct_{\mathcal{D}} = St_{\mathcal{D}} \times 2^{\mathcal{A}}$, and we use $\mathbf{C}^c$ and $\mathbf{C}^a$ to respectively denote the first and the second component of a context $\mathbf{C} \in Ct_{\mathcal{D}}$. Thus $\mathbf{C}^c$ represent a tuple of sets of entities arriving from the environment, and $\mathbf{C}^a$ denotes the *activated* agents.

The evolution of a DRS in an unconstrained environment is captured by a suitably re-defined notion of an interactive process, where the activated agents combine their local states to derive the next local states.

Definition 2. *Let $\mathcal{D} = (S, \{A_i\}_{i \in \mathcal{A}})$ be a drs. An (n-step)* interactive process *in $\mathcal{D}$ is $\pi = (\gamma, \delta)$, where:*

- *$\gamma = (\mathbf{C}_0, \mathbf{C}_1, \ldots, \mathbf{C}_n)$ and $\delta = (\mathbf{D}_0, \mathbf{D}_1, \ldots, \mathbf{D}_n)$,*
- *$\mathbf{C}_0, \mathbf{C}_1, \ldots, \mathbf{C}_n \in Ct_{\mathcal{D}}$,*
- *$\mathbf{D}_0, \mathbf{D}_1, \ldots, \mathbf{D}_n \in St_{\mathcal{D}}$,*
- *for each $i \in \{1, \ldots, n\}$ and $k \in \mathcal{A}$:*

$$\mathbf{D}_i[k] = \begin{cases} res_{A_k}\left(\mathbf{C}^c_{i-1}[k] \cup \bigcup_{j \in \mathbf{C}^a_{i-1}} \mathbf{D}_{i-1}[j]\right) & \text{if } k \in \mathbf{C}^a_{i-1} \\ \mathbf{D}_{i-1}[k] & \text{if } k \notin \mathbf{C}^a_{i-1} \end{cases}.$$

Example 2. Let us consider a simple example where two agents eventually synchronise to produce an entity by sharing their local states. Let $\mathcal{D} = (S, \{A_1, A_2, A_3\})$, where:

- $S = \{e_1, e_2, e_3, e_4, h\}$,
- $A_1 = \{(\{e_1\}, \{h\}, \{e_2\})\}$,
- $A_2 = \{(\{e_2\}, \{h\}, \{e_3\})\}$,
- $A_3 = \{(\{e_4\}, \{h\}, \{e_5\})\}$.

There exists a process $\pi = (\gamma, \delta)$ in $\mathcal{D}$ such that $\gamma = (\mathbf{C}_0, \mathbf{C}_1, \mathbf{C}_2)$ and $\delta = (\mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2)$ where $\mathbf{C}_i$ and $\mathbf{D}_i$ for $i \in \{0, \ldots, 2\}$ are defined as follows:

i	$\mathbf{C}_i$	$\mathbf{D}_i$
0	$((\{e_1\}, \varnothing, \{e_4\}), \{1, 3\})$	$(\varnothing, \varnothing, \varnothing)$
1	$((\varnothing, \varnothing, \varnothing), \{1, 2\})$	$(\{e_2\}, \varnothing, \{e_5\})$
2	$((\varnothing, \varnothing, \varnothing), \varnothing)$	$(\varnothing, \{e_3\}, \{e_5\})$

In the first step, i.e., for $i = 1$ we consider $\mathbf{C}_0$ and $\mathbf{D}_0$. The agents 1 and 3 are active and *receive* e_1 and e_4, respectively. Then, in $\mathbf{D}_1$ the entity e_2 and e_5 are produced for agent 1 and 3, respectively. In the next step, i.e., for $i = 2$, the context $\mathbf{C}_1$ indicates that 1 and 2 are the active agents and they receive no entities. As a result, in $\mathbf{D}_2$, these agents produce $\varnothing$ and e_3, respectively. Notice that e_2 is required for e_3 to be produced by 2 and the entity is provided by 1 by sharing the local states of the active agents. The agent 3 is inactive in this step thus its local state remains the same as in $\mathbf{D}_1$, i.e., it retains the previously produced entity e_5 without any reaction being executed locally. △

General interactive processes capture all possible evolutions of a drs. In practice, however, the behaviour of an environment is both constrained and may depend on the current state of the drs. We capture such an environment through the notion of an extended context automaton, which is a variant of context automaton where the transitions between the states are guarded by formulae restricting the allowed transitions for local states of drs' agents.

The *state constraints* $\mathcal{SC}_\mathcal{D}$ are Boolean formulae with propositions in the form of $i.e$, where $i \in \mathcal{A}$ and $e \in S$. Their grammar is as follows:

$$\mathfrak{sc} ::= \mathit{true} \mid i.e \mid \neg\mathfrak{sc} \mid \mathfrak{sc} \vee \mathfrak{sc}.$$

The fact that $\mathfrak{sc} \in \mathcal{SC}_\mathcal{D}$ holds in $\mathbf{W} \in \mathit{St}_\mathcal{D}$ is denoted by $\mathbf{W} \models_{sc} \mathfrak{sc}$. The satisfaction relation $\models_{sc}$ is defined as follows:

- $\mathbf{W} \models_{sc} \mathit{true}$,
- $\mathbf{W} \models_{sc} i.e$ iff $e \in \mathbf{W}[i]$,
- $\mathbf{W} \models_{sc} \neg\mathfrak{sc}$ iff $\mathbf{W} \not\models_{sc} \mathfrak{sc}$,
- $\mathbf{W} \models_{sc} \mathfrak{sc}_1 \vee \mathfrak{sc}_2$ iff $\mathbf{W} \models_{sc} \mathfrak{sc}_1$ or $\mathbf{W} \models_{sc} \mathfrak{sc}_2$.

Definition 3. *An* extended context automaton *(ECA) over $\mathcal{D}$, is a triple $\mathfrak{E} = (\mathcal{Q}, \mathfrak{q}^{init}, R)$ where:*

- *$\mathcal{Q}$ is a finite set of* locations*,*
- *$\mathfrak{q}^{init} \in \mathcal{Q}$ is the* initial location*, and*
- *$R \subseteq \mathcal{Q} \times \mathcal{SC}_\mathcal{D} \times \mathit{Ct}_\mathcal{D} \times \mathcal{Q}$ is a* transition relation*.*

We say that:

- *$\mathfrak{E}$ is* progressive *if, for all $\mathfrak{q} \in \mathcal{Q}$ and $\mathbf{W} \in \mathit{St}_\mathcal{D}$, there exists $(\mathfrak{q}, \mathfrak{sc}, \mathbf{C}, \mathfrak{q}') \in R$ such that $\mathbf{W} \models_{sc} \mathfrak{sc}$.*
- *$\mathfrak{E}$ is* state-oblivious *if $\mathfrak{sc} = \mathit{true}$, for each $(q, \mathfrak{sc}, \mathbf{C}, q') \in R$.*

For $(q, \mathfrak{sc}, \mathbf{C}, q') \in R$ *we also write* $q \xrightarrow{\mathfrak{sc},\mathbf{C}} q'$*. An extended context automaton over* $\mathcal{D}$ *is simply called* extended context automaton *if* $\mathcal{D}$ *is clear from the context.*

Being a progressive eca means that the environment never 'stops' the operation of a DRS with which it interacts. A state-oblivious ECA can be regarded as an extended *state-oblivious context controller* [20].

Progressiveness can be easily imposed on any extended context automaton $\mathfrak{E} = (\mathcal{Q}, \mathfrak{q}^{init}, R)$. Let $\bot \notin \mathcal{Q}$ and $\overline{\varnothing}^m$ be a tuple consisting of m empty sets. Then $prg(\mathfrak{E}) = (\mathcal{Q}', \mathfrak{q}^{init}, R')$ is an eca such that $\mathcal{Q}' = \mathcal{Q} \cup \{\bot\}$ and $R' = R \cup R_\bot \cup \{(\bot, true, (\varnothing^m, \varnothing), \bot)\}$, where $R_\bot = \{(q, \neg\mathfrak{sc}_q, (\varnothing^m, \varnothing), \bot) \mid q \in Q\}$ and $\mathfrak{sc}_q = \bigvee\{\mathfrak{sc} \mid (\mathfrak{q}, \mathfrak{sc}, \mathbf{C}, \mathfrak{q}') \in R\}$, for every $\mathfrak{q} \in \mathcal{Q}$,

The following result follows immediately from the above construction.

Lemma 1 *$prg(\mathfrak{E})$ is a progressive ECA, for every $\mathfrak{E}$.*

We then can formalise the notion of a DRS evolving in an environment provided by a progressive eca.

Definition 4. *A* context-restricted distributed reaction system *(CRDRS) is a pair* CR-$\mathcal{D} = (\mathcal{D}, \mathfrak{E})$ *such that $\mathcal{D}$ is a DRS and $\mathfrak{E}$ is a progressive ECA.*

The set of states of CR-$\mathcal{D}$ is defined as $St_{\text{CR-}\mathcal{D}} = St_\mathcal{D} \times \mathcal{Q}$. Let $\mathcal{S} \in St_{\text{CR-}\mathcal{D}}$ and $\mathcal{S} = (\mathbf{W}, \mathfrak{q})$, then with $\mathcal{D}(\mathcal{S})$ and $\mathfrak{E}(\mathcal{S})$ we denote, respectively, $\mathbf{W}$ and $\mathfrak{q}$.

Definition 5 (restricted interactive process). *Let* CR-$\mathcal{D} = (\mathcal{D}, \mathfrak{E})$ *be a CRDRS such that $\mathcal{D} = (S, \{A_i\}_{i\in\mathcal{A}})$ and $\mathfrak{E} = (\mathcal{Q}, \mathfrak{q}^{init}, R)$. An* ($n$-step) restricted interactive process in CR-$\mathcal{D}$ *is $\pi = (\zeta, \gamma, \delta)$, where:*

- $\zeta = (z_0, z_1, \ldots, z_n)$, $\gamma = (\mathbf{C}_0, \mathbf{C}_1, \ldots, \mathbf{C}_n)$, *and* $\delta = (\mathbf{D}_0, \mathbf{D}_1, \ldots, \mathbf{D}_n)$,
- $z_0, z_1, \ldots, z_n \in \mathcal{Q}$ *with* $z_0 = \mathfrak{q}^{init}$,
- $\mathbf{C}_0, \mathbf{C}_1, \ldots, \mathbf{C}_n \in Ct_\mathcal{D}$,
- $\mathbf{D}_0, \mathbf{D}_1, \ldots, \mathbf{D}_n \in St_\mathcal{D}$,
- *for each* $i \in \{1, \ldots, n\}$ *there exists* $\mathfrak{sc} \in \mathcal{SC}_\mathcal{D}$ *such that* $\mathbf{D}_{i-1} \models_{sc} \mathfrak{sc}$ *and* $(z_{i-1}, \mathfrak{sc}, \mathbf{C}_{i-1}, z_i) \in R$ *and for each* $k \in \mathcal{A}$*:*

$$\mathbf{D}_i[k] = \begin{cases} res_{A_k}(\mathbf{C}^c_{i-1}[k] \cup \bigcup_{j\in\mathbf{C}^a_{i-1}} \mathbf{D}_{i-1}[j]) & \text{if } k \in \mathbf{C}^a_{i-1}, \\ \mathbf{D}_{i-1}[k] & \text{if } k \notin \mathbf{C}^a_{i-1}. \end{cases}$$

Example 3. Here we introduce the train-gate-controller system (TGC) modelled using CRDRS. It is a commonly considered benchmark when dealing with multi-agent systems [26]. It consists of $m \geq 2$ trains trying to access a tunnel and a train controller that regulates access to the tunnel. At any given time there is at most one train in the tunnel allowed. The set of agents is defined as $\mathcal{A} = \{1, \ldots, m\}$. In this model the controller is modelled in the context automaton and there is no interaction between the agents. Then, we define the background set of the DRS modelling the system:

$$S = \{req, allowed, in, out, leave, h\}.$$

Now we give an intuition for the functioning of the system and the interpretation of the entities used. Initially all the trains are outside of the tunnel and are not trying to access it, which is indicated by the presence of *out*. The agent modelling a train approaching the tunnel produces *approach*. Any train may request access to the tunnel by producing the entity *req*. If access to the tunnel is requested then, provided the tunnel is empty, it is granted in the next step to one of the processes making the request via a nondeterministic choice in the context automaton. If a train is allowed to access the tunnel the context automaton provides it with the *allowed* entity as the context and then the agent produces *in* as a consequence, meaning it entered the tunnel. When a train leaves the tunnel it produces *leave* and *out*. The train requesting access to the tunnel keeps producing the *req* entity until it enters the tunnel, i.e., the production of *req* is inhibited by the *in* entity.

For each train we define the following reactions:

- $a_1^t = (\{out\}, \{h\}, \{approach\})$,
- $a_2^t = (\{approach\}, \{req\}, \{req\})$,
- $a_3^t = (\{allowed\}, \{h\}, \{in\})$,
- $a_4^t = (\{in\}, \{h\}, \{out, leave\})$,
- $a_5^t = (\{req\}, \{in\}, \{req\})$.

Then, for each train $i \in \mathcal{A}$ the set of its reactions is defined as follows:

$$A_i = \{a_1^t, a_2^t, a_3^t, a_4^t, a_5^t\}.$$

The distributed reaction system is defined as $\mathcal{D}_{\text{TGC}} = (S, \{A_i\}_{i \in \mathcal{A}})$.

To define a context-restricted distributed reaction system we introduce the context automaton $\mathfrak{E}_{\text{TGC}} = (\mathcal{Q}, \mathfrak{q}_0, R)$ such that $\mathcal{Q} = \{\mathfrak{q}_0, \mathfrak{q}_{green}, \mathfrak{q}_{red}\}$, and the set R consists of the following transitions:

- $\mathfrak{q}_0 \xrightarrow{true, (\{\{out\}, \ldots, \{out\}), \{1, \ldots, m\})} \mathfrak{q}_{green}$,
- for each $i \in \mathcal{A}$:
  - $\mathfrak{q}_{green} \xrightarrow{i.req, \mathbf{C}} \mathfrak{q}_{red}$, where: $\mathbf{C}^a = \{i\}$ and for each $j \in \mathcal{A}$:

$$\mathbf{C}^c[j] = \begin{cases} \{allowed\} & j = i, \\ \varnothing & j \in \mathcal{A} \setminus \{i\}, \end{cases}$$

  - $\mathfrak{q}_{green} \xrightarrow{\mathfrak{sc}, (\overline{\varnothing}^m, \{i\})} \mathfrak{q}_{green}$, where $\mathfrak{sc} = \neg \bigvee_{j \in \mathcal{A}} j.req$,
  - $\mathfrak{q}_{red} \xrightarrow{i.leave, (\overline{\varnothing}^m, \{i\})} \mathfrak{q}_{green}$,
  - $\mathfrak{q}_{red} \xrightarrow{\mathfrak{sc}, (\overline{\varnothing}^m, \{i\})} \mathfrak{q}_{red}$, where $\mathfrak{sc} = \neg \bigvee_{j \in \mathcal{A}} j.leave$.

The context automaton provides the set $\{out\}$ as the initial context sets to all the agents except the controller which is initialised with the empty set. Finally, the CRDRS for TGC is defined as:

$$\text{CR-}\mathcal{D}_{\text{TGC}} = (\mathcal{D}_{\text{TGC}}, prg(\mathfrak{E}_{\text{TGC}})).$$

△

Example 4. Here we present a variant of the model from Example 3, where the controller is modelled as an agent, and the system relies entirely on the communication between the agents. The system consists of $m \geq 2$ trains. The set of agents is defined as $\mathcal{A} = \{1, \ldots, n\}$. We assume n to correspond to the controller agent and $m = n - 1$ to the agent representing the last train. Then, we define the background set of the DRS modelling the system:

$$S = \{lock, leave, req, out, in, h\}.$$

For each train we define the following reactions:

- $a_1^t = (\{out\}, \{h\}, \{approach\})$,
- $a_2^t = (\{approach\}, \{req\}, \{req\})$,
- $a_3^t = (\{req\}, \{lock\}, \{in\})$,
- $a_4^t = (\{in\}, \{h\}, \{out, leave\})$,
- $a_5^t = (\{req\}, \{in\}, \{req\})$.

Then, for each train $i \in \{1, \ldots, m\}$ the set of its reactions is defined as follows:

$$A_i = \{a_1^t, a_2^t, a_3^t, a_4^t, a_5^t\}.$$

The set A_n of the reactions for the controller agent consists of the following reactions:

- $a_1^c = (\{lock\}, \{leave\}, \{lock\})$,
- $a_2^c = (\{req\}, \{h\}, \{lock\})$.

The reaction a_1^c ensures the tunnel is locked until the train that is currently inside leaves. The reaction a_2^c immediately acquires the lock reserving its exclusive access to the tunnel after a train requests access to it. Finally, the distributed reaction system modelling the protocol is defined as:

$$\mathcal{D}_{\text{TGC}} = (S, \{A_i\}_{i \in \mathcal{A}}).$$

To define the context-restricted distributed reaction system we introduce the extended context automaton $\mathfrak{E}_{\text{TGC}} = (\mathcal{Q}, \mathfrak{q}_0, R)$ such that $\mathcal{Q} = \{\mathfrak{q}_0, \mathfrak{q}_1\}$, and the set R consists of the following transitions:

- $\mathfrak{q}_0 \xrightarrow{true, (\{\{out\}, \ldots, \{out\}, \varnothing), \{1, \ldots, n\})} \mathfrak{q}_1$,
- $\mathfrak{q}_1 \xrightarrow{true, ((\varnothing, \ldots, \varnothing), \{i, n\})} \mathfrak{q}_1$ for all $i \in \{1, \ldots, m\}$.

Such a definition of the context automaton allows for the agents to communicate with the controller in pairs only. Note that $\mathfrak{E}_{\text{TGC}}$ is state-oblivious. Finally, the CRDRS for the train-gate-controller system is defined as $\text{CR-}\mathcal{D}_{\text{TGC}} = (\mathcal{D}_{\text{TGC}}, prg(\mathfrak{E}_{\text{TGC}}))$.

If access to the critical section is requested then, provided the tunnel is empty, it is granted in the next step to one of the agents making the request via a nondeterministic choice in the context automaton, and the requesting agent produces *in*. This results in producing *lock* entity by the controller which prevents other processes from entering the critical section. When a train leaves the tunnel it produces the *leave* entity which inhibits the controller from sustaining the *lock* entity.

△

4 Logic for temporal-epistemic properties

The language of *Computation Tree Logic of Knowledge for Reaction Systems*, rsCTLK for short, is defined by the following grammar:

$$\phi := i.e \mid \neg\phi \mid \phi \vee \phi \mid \mathsf{E}_{\mathfrak{sc}}\mathsf{X}\phi \mid \mathsf{E}_{\mathfrak{sc}}\mathsf{G}\phi \mid \mathsf{E}_{\mathfrak{sc}}[\phi\mathsf{U}\phi] \mid \overline{\mathrm{K}}_i\phi \mid \overline{\mathrm{C}}_\Gamma\phi,$$

where $i \in \mathcal{A}$, $e \in S$, $\mathfrak{sc} \in \mathcal{SC}_\mathcal{D}$ and $\Gamma \subseteq \mathcal{A}$.

The aim of the logic is to resemble CTLK [31], while retaining the expressive power of rsCTL. The grammar uses the propositional and temporal operators of rsCTL [25]. In place of propositional variables we use $i.e$ which allows for specifying entities in local states of the individual agents. Additionally, the logic uses epistemic operators for specifying knowledge properties: the operators $\overline{\mathrm{K}}_i\phi$ and $\overline{\mathrm{C}}_\Gamma\phi$ are the existential counterparts of the universal $\mathrm{K}_i\phi$ and $\mathrm{C}_\Gamma\phi$, respectively, which we derive later. Here we provide the intuitive meaning of the universal operators: $\mathrm{K}_i\phi$ means the agent $i \in \mathcal{A}$ *knows* ϕ and $\mathrm{C}_\Gamma\phi$ means ϕ is *common knowledge* amongst the agents of Γ.

Definition 6. *Let ϕ be an rsCTLK formula. Then, $\mathsf{d}(\phi)$ is the* depth of ϕ *and is defined recursively as follows:*

- *if $\phi = i.e$, where $i \in \mathcal{A}$ and $e \in S$, then $\mathsf{d}(\phi) = 1$,*
- *if $\phi \in \{\neg\phi', \mathsf{E}_{\mathfrak{sc}}\mathsf{X}\phi', \mathsf{E}_{\mathfrak{sc}}\mathsf{G}\phi', \overline{\mathrm{K}}_i\phi', \overline{\mathrm{C}}_\Gamma\phi'\}$, then $\mathsf{d}(\phi) = \mathsf{d}(\phi') + 1$,*
- *if $\phi \in \{\phi' \vee \phi'', \mathsf{E}_{\mathfrak{sc}}[\phi'\mathsf{U}\phi'']\}$, then $\mathsf{d}(\phi) = max(\{\mathsf{d}(\phi'), \mathsf{d}(\phi'')\}) + 1$.*

Definition 7. *Let CR-$\mathcal{D}$ $= (\mathcal{D}, \mathfrak{E})$ where $\mathcal{D} = (S, \{A_i\}_{i\in\mathcal{A}})$ is a distributed reaction system and $\mathfrak{E} = (\mathcal{Q}, \mathfrak{q}^{init}, R)$ is an extended context automaton over $\mathcal{D}$. Then, the* model for CR-$\mathcal{D}$ *is a tuple $\mathcal{M}_{\text{CR-}\mathcal{D}} = (St_{\text{CR-}\mathcal{D}}, \mathcal{S}_{init}, \longrightarrow)$, where:*

1. *$St_{\text{CR-}\mathcal{D}}$ is the set of states of $\mathcal{M}_{\text{CR-}\mathcal{D}}$,*
2. *$\mathcal{S}_{init} = \left(\mathfrak{q}^{init}, (\varnothing, \ldots, \varnothing)\right) \in St_{\text{CR-}\mathcal{D}}$ is the initial state,*
3. *$\longrightarrow \subseteq St_{\text{CR-}\mathcal{D}} \times Ct_\mathcal{D} \times St_{\text{CR-}\mathcal{D}}$ is the transition relation such that for all $\mathbf{S}, \mathbf{S}' \in St_\mathcal{D}$, $\mathfrak{q}, \mathfrak{q}' \in \mathcal{Q}$, $\mathbf{C} \in Ct_\mathcal{D}$: $((\mathbf{S}, \mathfrak{q}), \mathbf{C}, (\mathbf{S}', \mathfrak{q}')) \in \longrightarrow$ iff*
   (a) $(\mathfrak{q}, \mathfrak{sc}, \mathbf{C}, \mathfrak{q}') \in R$ for some $\mathfrak{sc} \in \mathcal{SC}_\mathcal{D}$ and $\mathbf{S} \models_{sc} \mathfrak{sc}$,
   (b) for each $k \in \mathcal{A}$:
   - *$\mathbf{S}'[k] = res_{A_i}\left(\mathbf{C}^c[k] \cup \bigcup\limits_{j\in\mathbf{C}^a} \mathbf{S}[j]\right)$ if $k \in \mathbf{C}^a$,*
   - *$\mathbf{S}'[k] = \mathbf{S}[k]$ if $k \notin \mathbf{C}^a$.*

Each element $(\mathcal{S}, \mathbf{C}, \mathcal{S}') \in \longrightarrow$ is denoted by $\mathcal{S} \xrightarrow{\mathbf{C}} \mathcal{S}'$.

The following lemma follows from Definition 7 and seriality of the transition relation of $\mathfrak{E}$.

Lemma 2 *For each $\mathcal{S} \in St_{\text{CR-}\mathcal{D}}$ there exists $\mathbf{C} \in Ct_\mathcal{D}$ and $\mathcal{S}' \in St_{\text{CR-}\mathcal{D}}$ such that $\mathcal{S} \xrightarrow{\mathbf{C}} \mathcal{S}'$.*

Definition 8. *Let* CR-$\mathcal{D} = (\mathcal{D}, \mathfrak{E})$. *A path over* $\mathfrak{sc} \in \mathcal{SC}_{\mathcal{D}}$ *in* $\mathcal{M}_{\text{CR-}\mathcal{D}}$ *is an infinite sequence* $\sigma = (\mathcal{S}_0, \mathbf{C}_0, \mathcal{S}_1, \mathbf{C}_1, \dots)$ *such that for* $i \geq 0$*:* $\mathcal{S}_i \xrightarrow{\mathbf{C}_i} \mathcal{S}_{i+1}$ *and* $\mathbf{C}_i^c \models_{sc} \mathfrak{sc}$.

The set of all the paths over $\mathfrak{sc}$ is denoted by $\Pi_{\mathfrak{sc}}$. For each $i \geq 0$, the i^{th} state of the path σ is denoted by $\sigma_s(i)$, and the i^{th} action of the path σ is denoted by $\sigma_a(i)$. By $\Pi_{\mathfrak{sc}}(\mathcal{S})$ we denote the set of all the paths over $\mathfrak{sc}$ that start in $\mathcal{S} \in St_{\text{CR-}\mathcal{D}}$, i.e., $\Pi_{\mathfrak{sc}}(\mathcal{S}) = \{\sigma \in \Pi_{\mathfrak{sc}} \mid \sigma_s(0) = w\}$.

Let $\mathcal{S}, \mathcal{S}' \in St_{\text{CR-}\mathcal{D}}$ and $\mathfrak{sc} \in \mathcal{SC}_{\mathcal{D}}$. We say that $\mathcal{S}'$ is a $\mathfrak{sc}$-successor of $\mathcal{S}$ (denoted by $\mathcal{S} \longrightarrow_{\mathfrak{sc}} \mathcal{S}'$) iff there exists $\mathbf{C} \in Ct_{\mathcal{D}}$ such that $\mathbf{C}^c \models_{sc} \mathfrak{sc}$ and $\mathcal{S} \xrightarrow{\mathbf{C}} \mathcal{S}'$. The relation $\twoheadrightarrow \subseteq St_{\text{CR-}\mathcal{D}} \times St_{\text{CR-}\mathcal{D}}$ is defined as follows:

$$(\mathcal{S}, \mathcal{S}') \in \twoheadrightarrow \text{ iff there exists } \mathbf{C} \in Ct_{\mathcal{D}} \text{ such that } \mathcal{S} \xrightarrow{\mathbf{C}} \mathcal{S}'.$$

The relation $\twoheadrightarrow_r \subseteq St_{\text{CR-}\mathcal{D}} \times St_{\text{CR-}\mathcal{D}}$ is the transitive closure of $\twoheadrightarrow$.

Definition 9. *A state* $\mathcal{S} \in St_{\text{CR-}\mathcal{D}}$ *is* reachable *iff* $\mathcal{S}_{init} \twoheadrightarrow_r \mathcal{S}$.

Then, $Reach(\text{CR-}\mathcal{D}) \subseteq St_{\text{CR-}\mathcal{D}}$ is the set of the reachable states of $\mathcal{M}_{\text{CR-}\mathcal{D}}$, i.e.,

$$Reach(\text{CR-}\mathcal{D}) = \{\mathcal{S} \in St_{\text{CR-}\mathcal{D}} \mid \mathcal{S}_{init} \twoheadrightarrow_r \mathcal{S}\}.$$

Definition 10. *Let* CR-$\mathcal{D} = (\mathcal{D}, \mathfrak{E})$ *where* $\mathcal{D} = (S, \{A_i\}_{i \in \mathcal{A}})$. *For each agent* $i \in \mathcal{A}$, *the epistemic indistinguishability relation* $\sim_i \subseteq St_{\text{CR-}\mathcal{D}} \times St_{\text{CR-}\mathcal{D}}$ *is defined by:*

$$\mathcal{S} \sim_i \mathcal{S}' \quad \textit{iff} \quad \mathcal{D}(\mathcal{S})[i] = \mathcal{D}(\mathcal{S}')[i] \text{ and } \mathcal{S}, \mathcal{S}' \in Reach(\text{CR-}\mathcal{D}).$$

Then, for a group of agents $\Gamma \subset \mathcal{A}$, *the union of the indistinguishability relations of* Γ *is defined as* $\sim_\Gamma^{\text{E}} = \bigcup_{i \in \Gamma} \sim_i$. *By* $\sim_\Gamma^{\text{C}}$ *we denote the transitive closure of* $\sim_\Gamma^{\text{E}}$.

If $\mathcal{S} \sim_i \mathcal{S}'$ *for some* $i \in \Gamma$, *we say* $\mathcal{S}$ *is a* Γ*-neighbour, or an* i*-neighbour, of* $\mathcal{S}'$.

Definition 11. *Let* $\mathcal{M}_{\text{CR-}\mathcal{D}} = (St_{\text{CR-}\mathcal{D}}, \mathcal{S}_{init}, \longrightarrow)$ *be a model for* CR-$\mathcal{D}$, $\mathcal{S} \in St_{\text{CR-}\mathcal{D}}$ *be a state of* $\mathcal{M}_{\text{CR-}\mathcal{D}}$, *and* ϕ *be an rs*CTLK *formula. The fact that* ϕ *holds in* $\mathcal{S}$ *is denoted by* $\mathcal{M}_{\text{CR-}\mathcal{D}}, \mathcal{S} \models \phi$, *or simply* $\mathcal{S} \models \phi$ *when* $\mathcal{M}_{\text{CR-}\mathcal{D}}$ *is clear from the context. The relation* $\models$ *is defined recursively as follows:*

$$\begin{array}{ll}
\mathcal{S} \models i.e & \textit{iff } e \in \mathcal{D}(\mathcal{S})[i], \\
\mathcal{S} \models \neg\phi & \textit{iff } \mathcal{S} \not\models \phi, \\
\mathcal{S} \models \phi \vee \psi & \textit{iff } \mathcal{S} \models \phi \textit{ or } \mathcal{S} \models \psi, \\
\mathcal{S} \models \mathsf{E}_{\mathfrak{sc}}\mathsf{X}\phi & \textit{iff } (\exists \sigma \in \Pi_{\mathfrak{sc}}(\mathcal{S}))\ \sigma_s(1) \models \phi, \\
\mathcal{S} \models \mathsf{E}_{\mathfrak{sc}}\mathsf{G}\phi & \textit{iff } (\exists \sigma \in \Pi_{\mathfrak{sc}}(\mathcal{S}))(\forall i \geq 0)\big(\sigma_s(i) \models \phi\big), \\
\mathcal{S} \models \mathsf{E}_{\mathfrak{sc}}[\phi\mathsf{U}\psi] & \textit{iff } (\exists \sigma \in \Pi_{\mathfrak{sc}}(\mathcal{S}))(\exists i \geq 0)\big(\sigma_s(i) \models \psi \\
 & \qquad \textit{and } (\forall 0 \leq j < i)\ \sigma_s(j) \models \phi\big), \\
\mathcal{S} \models \overline{\mathrm{K}}_i\phi & \textit{iff } (\exists \mathcal{S}' \in St_{\text{CR-}\mathcal{D}})(\mathcal{S} \sim_i \mathcal{S}' \textit{ and } \mathcal{S}' \models \phi), \\
\mathcal{S} \models \overline{\mathrm{C}}_\Gamma\phi & \textit{iff } (\exists \mathcal{S}' \in St_{\text{CR-}\mathcal{D}})(\mathcal{S} \sim_\Gamma^{\text{C}} \mathcal{S}' \textit{ and } \mathcal{S}' \models \phi).
\end{array}$$

Next, we define derived operators which also introduce the universal path quantifier $\mathsf{A}_{\mathfrak{sc}}$ meaning 'for all the paths over $\mathfrak{sc}$':

– $true \stackrel{def}{=} i.e \vee \neg i.e$ for any $i \in \mathcal{A}$, $e \in S$,
– $\phi \wedge \psi \stackrel{def}{=} \neg(\neg\phi \vee \neg\psi)$,
– $\phi \Rightarrow \psi \stackrel{def}{=} \neg\phi \vee \psi$,
– $\mathsf{E}_{\mathfrak{sc}}\mathsf{F}\phi \stackrel{def}{=} \mathsf{E}_{\mathfrak{sc}}[true\mathsf{U}\phi]$,
– $\mathsf{A}_{\mathfrak{sc}}\mathsf{F}\phi \stackrel{def}{=} \neg\mathsf{E}_{\mathfrak{sc}}\mathsf{G}\neg\phi$,
– $\mathsf{A}_{\mathfrak{sc}}\mathsf{X}\phi \stackrel{def}{=} \neg\mathsf{E}_{\mathfrak{sc}}\mathsf{X}\neg\phi$,
– $\mathsf{A}_{\mathfrak{sc}}\mathsf{G}\phi \stackrel{def}{=} \neg\mathsf{E}_{\mathfrak{sc}}[true\mathsf{U}\neg\phi]$.

Moreover, we assume $\mathfrak{sc} = true$ when $\mathfrak{sc}$ is not explicitly specified for any of the rsCTLK operators, e.g., $\mathsf{EF}\phi \stackrel{def}{=} \mathsf{E}_{true}\mathsf{F}\phi$. We also define the following derived epistemic operators:

– $\mathrm{K}_i\phi \stackrel{def}{=} \neg\overline{\mathrm{K}}_i\neg\phi$,
– $\mathrm{C}_\Gamma\phi \stackrel{def}{=} \neg\overline{\mathrm{C}}_\Gamma\neg\phi$,
– $\overline{\mathrm{E}}_\Gamma\phi \stackrel{def}{=} \bigvee_{i\in\Gamma}\overline{\mathrm{K}}_i\phi$, and
– $\mathrm{E}_\Gamma\phi \stackrel{def}{=} \neg\overline{\mathrm{E}}_\Gamma\neg\phi$.

We assume that an rsCTLK formula ϕ *holds in the model* $\mathcal{M}_{\text{CR-}\mathcal{D}}$ if and only if ϕ holds in the initial state of $\mathcal{M}_{\text{CR-}\mathcal{D}}$:

$$\mathcal{M}_{\text{CR-}\mathcal{D}} \models \phi \text{ iff } \mathcal{M}_{\text{CR-}\mathcal{D}}, \mathcal{S}_{init} \models \phi.$$

Example 5. Here we specify some rsCTLK properties of the TGC system presented in Example 3.

1. It is possible for each train to eventually enter the tunnel when they receive the *allowed* entity in the context:

$$\phi_1 = \bigwedge_{i\in\{1,\ldots,m\}} \mathsf{EF}\left(\mathsf{E}_{i.allowed}\mathsf{X}\left(i.in\right)\right).$$

2. In all the states of all the paths, if the i^{th} train is in the tunnel, then it *knows* that no other train is in the tunnel, i.e., it is the only train that is in the tunnel:

$$\phi_2 = \mathsf{A}_{\mathfrak{sc}}\mathsf{G}\left(i.in \implies \mathrm{K}_i\left(\bigwedge_{\substack{j\in\{1,\ldots,m\},\\ i\neq j}} \neg j.in\right)\right),$$

where the paths are constrained with

$$\mathfrak{sc} = \bigwedge_{i\in\{1,\ldots,m\}} (\neg i.s)$$

to only those where h is not provided as the context for any of the trains.

3. In all the states of all the paths the i^{th} train *knows* about the mutual exclusion property of the access to the tunnel:

$$\phi_3 = \mathsf{AG}\left(\mathrm{K}_i \bigwedge_{\substack{j,k\in\{1,\ldots,m\},\\ j<k}} \neg\left(j.in \wedge k.in\right)\right).$$

4. In all the states of all the paths if the i^{th} train is in the tunnel then it is a common knowledge amongst all the agents that it is the only train in the tunnel:

$$\phi_4 = \mathsf{AG}\left(i.in \implies \mathrm{C}_{\{1,\ldots,m\}}\left(\bigwedge_{\substack{j\in\{1,\ldots,m\},\\ i\neq j}} \neg j.in\right)\right).$$

5 Model checking for rsCTLK

In this section we describe a model checking method for rsCTLK, which is based on computing fixed points. For calculating the results for temporal operators we use the algorithms presented in [25]. Firstly, we describe an algorithm for computing all the reachable states of $\mathcal{M}_{\text{CR-}\mathcal{D}}$ and, later on, we provide a method for computing the set of states, where a given rsCTLK formula holds.

For the purpose of computing the set of all the reachable states we need the notion of a fixed point (we use $|W|$ to denote the cardinality of a set W). Let W be a finite set and $\tau : 2^W \longrightarrow 2^W$ be a *monotone* function, i.e., $X \subseteq Y$ implies $\tau(X) \subseteq \tau(Y)$ for all $X, Y \subseteq W$. Let $\tau^i(X)$ be defined by $\tau^0(X) = X$ and $\tau^{i+1}(X) = \tau(\tau^i(X))$. We say that $X' \subseteq W$ is a *fixed point* of τ if $\tau(X') = X'$. It can be proved that if τ is monotone and W is a finite set, then there exist $m, n \leq |W|$ such that $\tau^m(\emptyset)$ is the least fixed point of τ (denoted by $\mu X.\tau(X)$) and $\tau^n(W)$ is the greatest fixed point of τ (denoted by $\nu X.\tau(X)$).

Let $\mathcal{M}_{\text{CR-}\mathcal{D}} = (St_{\text{CR-}\mathcal{D}}, \mathcal{S}_{init}, \longrightarrow)$ be a model. We define the function that assigns the set of the $\mathfrak{sc}$-successors to the states in $W \subseteq St_{\text{CR-}\mathcal{D}}$:

$$\mathsf{post}_{\mathfrak{sc}}(W) = \{\mathcal{S}' \in St_{\text{CR-}\mathcal{D}} \mid (\exists \mathcal{S} \in W)\ \mathcal{S} \longrightarrow_{\mathfrak{sc}} \mathcal{S}')\},$$

where $\mathfrak{sc} \in \mathcal{SC}_{\mathcal{D}}$.

The set $\mathit{Reach}(\text{CR-}\mathcal{D}) \subseteq \mathcal{S}$ can be characterised by the following fixed point equation:

$$\mathit{Reach}(\text{CR-}\mathcal{D}) = \mu X.\big(\mathcal{S}_{init} \cup X \cup \mathsf{post}_{true}(X)\big).$$

Algorithm 1 implements the fixed-point computation of the reachable states for a given model $\mathcal{M}_{\text{CR-}\mathcal{D}} = (St_{\text{CR-}\mathcal{D}}, \mathcal{S}_{init}, \longrightarrow)$. On Line 7 the procedure returns the set X which is equal to $\mathit{Reach}(\text{CR-}\mathcal{D})$.

The set of all the reachable states of $\mathcal{M}_{\text{CR-}\mathcal{D}}$ in which ϕ holds is denoted by $[\![\mathcal{M}_{\text{CR-}\mathcal{D}}, \phi]\!]$ or by $[\![\phi]\!]$ if $\mathcal{M}_{\text{CR-}\mathcal{D}}$ is implicitly understood. For $W \subseteq \mathit{Reach}(\text{CR-}\mathcal{D})$ we define a function that assigns the set of the $\mathfrak{sc}$-predecessors to W:

$$\mathsf{pre}^{\exists}_{\mathfrak{sc}}(W) \stackrel{def}{=} \{\mathcal{S} \in \mathit{Reach}(\text{CR-}\mathcal{D}) \mid (\exists \mathcal{S}' \in W)\ \mathcal{S} \longrightarrow_{\mathfrak{sc}} \mathcal{S}')\}.$$

Algorithm 1 The algorithm for computing the set $Reach(\text{CR-}\mathcal{D})$

```
1: X := S_init
2: X_p := ∅
3: while X ≠ X_p do
4:     X_p := X
5:     X := X ∪ post_{Ct_D}(X)
6: end while
7: return X
```

Let ϕ, ψ be some rsCTLK formulae. We define then the following sets:

- $[\![\neg\phi]\!] \stackrel{def}{=} Reach(\text{CR-}\mathcal{D}) \setminus [\![\phi]\!]$,
- $[\![\phi \wedge \psi]\!] \stackrel{def}{=} [\![\phi]\!] \cap [\![\psi]\!]$, $[\![\phi \vee \psi]\!] \stackrel{def}{=} [\![\phi]\!] \cup [\![\psi]\!]$,
- $[\![\mathsf{E}_{\mathfrak{sc}}\mathsf{X}\phi]\!] \stackrel{def}{=} \mathsf{pre}^{\exists}_{\mathfrak{sc}}([\![\phi]\!])$.

The remaining temporal operators are defined as the following fixed points:

- $[\![\mathsf{E}_{\mathfrak{sc}}\mathsf{G}\phi]\!] \stackrel{def}{=} \nu X.\big([\![\phi]\!] \cap \mathsf{pre}^{\exists}_{\mathfrak{sc}}(X)\big)$,
- $[\![\mathsf{E}_{\mathfrak{sc}}[\phi\mathsf{U}\psi]]\!] \stackrel{def}{=} \mu X.\big([\![\psi]\!] \cup ([\![\phi]\!] \cap \mathsf{pre}^{\exists}_{\mathfrak{sc}}(X))\big)$.

In the case of $[\![\mathsf{E}_{\mathfrak{sc}}\mathsf{G}\phi]\!]$ the greatest fixed point computation is involved, which in each iteration removes states that do not have a $\mathfrak{sc}$-predecessor in which ϕ is satisfied. In the case of $[\![\mathsf{E}_{\mathfrak{sc}}[\phi\mathsf{U}\psi]]\!]$ the least fixed point computation is involved, such that in each iteration the $\mathfrak{sc}$-predecessors, in which ϕ is satisfied are added to the set of states in which ψ is satisfied. See Algorithm 5 and 4 for the pseudocode of the procedures implementing the described fixed point computations. For $\Gamma \subseteq \mathcal{A}$ we introduce the set of *Γ-neighbours* of the states in W:

$$\mathsf{nb}_\Gamma(W) \stackrel{def}{=} \{\mathcal{S} \in Reach(\text{CR-}\mathcal{D}) \mid (\exists \mathcal{S}' \in W)(\exists i \in \Gamma)\ \mathcal{S} \sim_i \mathcal{S}'\}.$$

Then, the sets for the epistemic operators are defined as follows:

- $[\![\overline{\mathrm{K}}_i\phi]\!] \stackrel{def}{=} \mathsf{nb}_{\{i\}}([\![\phi]\!])$,
- $[\![\overline{\mathrm{C}}_\Gamma\phi]\!] \stackrel{def}{=} \mu X.\,([\![\phi]\!] \cup \mathsf{nb}_\Gamma(X))$.

To compute the set of states for $\overline{\mathrm{K}}_i\phi$ we find all the i-neighbours of the states in which ϕ holds. In the case of $\overline{\mathrm{C}}_\Gamma\phi$, to obtain the set of states in which the formula holds, we calculate a least fixed point. The procedures for $[\![\overline{\mathrm{K}}_i\phi]\!]$ and $[\![\overline{\mathrm{C}}_\Gamma\phi]\!]$ are presented in Algorithm 2 and 3, respectively.

The overall procedure $\mathsf{check}_{rs\text{CTLK}(\phi)}$ for computing the set of states in which an rsCTLK formula ϕ holds is outlined in Algorithm 6.

Definition 12. *Given* CR-$\mathcal{D}$ *and an rs*CTLK *formula* ϕ, rsCTLK model checking *is the problem of deciding whether* $\mathcal{M}_{\text{CR-}\mathcal{D}} \models \phi$.

Algorithm 2 $\mathsf{checkNK}(i, \phi)$

$X := \mathsf{check}_{rs\mathrm{CTLK}}(\phi)$
return $\mathsf{nb}_{\{i\}}(X)$

Algorithm 3 $\mathsf{checkNC}(\Gamma, \phi)$

$X := \varnothing$, $X_p := \mathit{Reach}(\text{CR-}\mathcal{D})$
$Y_\psi = \mathsf{check}_{rs\mathrm{CTLK}}(\phi)$
while $X \neq X_p$ **do**
    $X_p := X$
    $X := \mathsf{nb}_\Gamma(Y_\psi \cup X)$
end while
return X

Algorithm 4 $\mathsf{checkEU}(\mathfrak{sc}, \phi, \psi)$

$X := \varnothing$, $X_p := \mathit{Reach}(\text{CR-}\mathcal{D})$
$Y_\phi := \mathsf{check}_{rs\mathrm{CTLK}}(\phi)$, $Y_\psi := \mathsf{check}_{rs\mathrm{CTLK}}(\psi)$
while $X \neq X_p$ **do**
    $X_p := X$
    $X := Y_\psi \cup (Y_\phi \cap \mathsf{pre}^{\exists}_{\mathfrak{sc}}(X))$
end while
return X

Algorithm 5 $\mathsf{checkEG}(\mathfrak{sc}, \phi)$

$X := \mathit{Reach}(\text{CR-}\mathcal{D})$, $X_p := \varnothing$
$Y_\phi := \mathsf{check}_{rs\mathrm{CTLK}}(\phi)$
while $X \neq X_p$ **do**
    $X_p := X$
    $X := \big(Y_\phi \cap \mathsf{pre}^{\exists}_{\mathfrak{sc}}(X)\big)$
end while
return X

Algorithm 6 $\mathsf{check}_{rs\text{CTLK}}(\phi)$

if $\phi = i.e$ **then**
  return $\{\mathcal{S} \in St_{\text{CR-}\mathcal{D}} \mid e \in \mathcal{D}(\mathcal{S})[i]\} \cap \mathit{Reach}(\text{CR-}\mathcal{D})$
else if $\phi = \neg\phi_1$ **then**
  return $\mathit{Reach}(\text{CR-}\mathcal{D}) \setminus \mathsf{check}_{rs\text{CTLK}}(\phi_1)$
else if $\phi = \phi_1 \vee \phi_2$ **then**
  return $\mathsf{check}_{rs\text{CTLK}}(\phi_1) \cup \mathsf{check}_{rs\text{CTLK}}(\phi_2)$
else if $\phi = \mathsf{E}_{\mathfrak{sc}}\mathsf{X}\phi_1$ **then**
  return $\mathsf{pre}^{\exists}_{\mathfrak{sc}}(\mathsf{check}_{rs\text{CTLK}}(\phi_1))$
else if $\phi = \mathsf{E}_{\mathfrak{sc}}\mathsf{G}\phi_1$ **then**
  return $\mathsf{checkEG}(\mathfrak{sc}, \phi_1)$
else if $\phi = \mathsf{E}_{\mathfrak{sc}}[\phi_1\mathsf{U}\phi_2]$ **then**
  return $\mathsf{checkEU}(\mathfrak{sc}, \phi_1, \phi_2)$
else if $\phi = \overline{\mathsf{K}}_i\phi_1$ **then**
  return $\mathsf{checkNK}(i, \phi_1)$
else if $\phi = \overline{\mathsf{C}}_\Gamma\phi_1$ **then**
  return $\mathsf{checkNC}(\Gamma, \phi_1)$
end if

Lemma 3 *The rs*CTLK *model checking problem is* PSPACE*-hard.*

Proof. Follows from the fact that QSAT can be reduced to the rsCTLK model checking problem. Every initialised context restricted reaction system (ICRRS) can be translated into CRDRS with a single agent and rsCTL is a subset of rsCTLK. Therefore, the reduction is similar to the one for ICRRS and rsCTL [25].

Lemma 4 *The rs*CTLK *model checking problem is in* PSPACE*.*

Proof. We show a nondeterministic algorithm for deciding whether $\mathcal{M}_{\text{CR-}\mathcal{D}} \models \phi$, which requires at most polynomial space in the size of the input, i.e., the formula ϕ and CR-$\mathcal{D}$. The proof is similar to the one of [25] for rsCTL and initialised context-restricted reaction systems.

The algorithm uses a recursive procedure $\mathit{label}(\mathcal{S}, \phi)$ which returns *true* iff $\mathcal{M}_{\text{CR-}\mathcal{D}}, \mathcal{S} \models \phi$, where $\mathcal{S} \subseteq St_{\text{CR-}\mathcal{D}}$; otherwise, it returns *false*. The encoding of each state requires space $\mathcal{O}(|S| \cdot |\mathcal{A}|)$ and each successor can be generated in space $\mathcal{O}(|S| \cdot |\mathcal{A}|)$, whereas the overall algorithm requires space $\mathcal{O}\big(|S| \cdot |\mathcal{A}| \cdot \mathsf{d}(\phi)\big)$. The proof follows by the induction on the length of the formula ϕ. The cases in which ϕ does not contain any temporal and epistemic operators, or $\phi = \mathsf{E}_{\mathfrak{sc}}\mathsf{X}\phi_1$, are straightforward.

The intuition for the nondeterministic procedure for checking $\phi = \mathsf{E}_{\mathfrak{sc}}[\phi_1\mathsf{U}\phi_2]$ in $\mathcal{S} \subseteq St_{\text{CR-}\mathcal{D}}$ is outlined in Algorithm 7. It nondeterministically selects states and contexts $\mathbf{C} \in Ct_{\mathcal{D}}$ of a path over $\mathfrak{sc}$, verifying at each step if the state chosen is an $\mathfrak{sc}$-successor of the previous state via the action $\mathbf{C}$, and if ϕ_1 holds in that state. If not, then an action and a state are selected again. At each step of the procedure, only two states are stored: the current state and its successor. If in the current state ϕ_2 holds, then the algorithm returns *true*.

Algorithm 7 Nondeterministic procedure for checking $\mathsf{E}_{\mathfrak{sc}}[\phi_1 \mathsf{U} \phi_2]$ in $\mathcal{S} \in St_{\text{CR-}\mathcal{D}}$

1: $\widehat{\mathcal{S}} := \mathcal{S}$
2: *checking*:
3: **if** $label(\widehat{\mathcal{S}}, \phi_2)$ **then**
4: **return** *true*
5: **end if**
6: **if** $\neg label(\widehat{\mathcal{S}}, \phi_1)$ **then**
7: **return** *false*
8: **end if**
9: *guessing*:
10: guess $\widehat{\mathcal{S}}' \in St_{\text{CR-}\mathcal{D}}$
11: guess $\mathbf{C} \in Ct_{\mathcal{D}}$
12: **if** $\mathbf{C}^c \not\models_{sc} \mathfrak{sc}$ **then**
13: **goto** *guessing*
14: **else if** $\neg label(\widehat{\mathcal{S}}', \phi_1)$ **then**
15: **goto** *guessing*
16: **else if** $\left((\mathcal{S}, \mathbf{C}, \widehat{\mathcal{S}}') \notin \longrightarrow\right)$ **then**
17: **goto** *guessing*
18: **end if**
19: $\widehat{\mathcal{S}} := \widehat{\mathcal{S}}'$
20: **goto** *checking*

The intuition for the procedure for checking $\phi = \mathsf{E}_{\mathfrak{sc}} \mathsf{G} \phi_1$ in $\mathcal{S} \in St_{\text{CR-}\mathcal{D}}$ is outlined in Algorithm 8. Similarily to the previous case, the algorithm guesses a path over $\mathfrak{sc}$, and nondeterministically chooses a state of that path, for the purpose of detecting a loop. At each step only three states are stored: the current state, its successor, and a state used for loop detection. The procedure ensures that ϕ_1 holds in the current state and generates its successor. If the state used for detecting a loop has appeared again in the sequence, then the search stops, and the algorithm returns *true*.

The intuition for the procedure for checking $\phi = \overline{\mathrm{K}}_i \phi_1$ in $\mathcal{S} \in St_{\text{CR-}\mathcal{D}}$ is outlined in Algorithm 9. The procedure nondeterministically selects a state $\widehat{\mathcal{S}}$ and checks if $\mathcal{S} \sim_i \widehat{\mathcal{S}}$ and if $\widehat{\mathcal{S}}$ is reachable in $\mathcal{M}_{\text{CR-}\mathcal{D}}$. The reachability is tested by using the *label* procedure to check if $\mathsf{EF}\phi_{\mathcal{D}(\widehat{\mathcal{S}})}$ holds in the initial state of the model, where $\phi_{\mathcal{D}(\widehat{\mathcal{S}})}$ is the formula corresponding to the state $\mathcal{D}(\widehat{\mathcal{S}})$, i.e., where the state is encoded using the rsCTLK syntax.

The intuition for the procedure for checking $\phi = \overline{\mathrm{C}}_\Gamma \phi_1$ in $\mathcal{S} \in St_{\text{CR-}\mathcal{D}}$ is outlined in Algorithm 10. The algorithm searches for a path via the relation $\sim_\Gamma^{\mathrm{C}}$ from $\mathcal{S}$ to a state in which ϕ_1 holds and it is a reachable state of the model. Initially, $\widehat{\mathcal{S}}$ is set to $\mathcal{S}$. If ϕ_1 holds in $\widehat{\mathcal{S}}$ and the state is reachable, then the algorithm returns *true*; otherwise it nondeterministically selects $\widehat{\mathcal{S}}' \in St_{\text{CR-}\mathcal{D}}$ accessible via $\sim_i$ for $i \in \Gamma$ and then it jumps to the beginning of the procedure and checks if ϕ_1 holds in that state.

The procedure returns *false* if no sequence for which the procedure returns *true* could be found. To ensure the procedure terminates for each sequence guessed,

Algorithm 8 Nondeterministic procedure for checking $\mathsf{E}_{\mathfrak{sc}}\mathsf{G}\phi_1$ in $\mathcal{S} \in St_{\text{CR-}\mathcal{D}}$

$\widehat{\mathcal{S}} := \mathcal{S}$
$L := \mathit{false}$
if $\neg \mathit{label}(\widehat{\mathcal{S}}, \phi_1)$ **then**
  return *false*
end if
guessing:
guess $\widehat{\mathcal{S}}' \in St_{\text{CR-}\mathcal{D}}$ and $\mathbf{C} \in Ct_{\mathcal{D}}$
if $\mathbf{C}^c \not\models_{sc} \mathfrak{sc}$ **then**
  goto *guessing*
else if $\neg \mathit{label}(\widehat{\mathcal{S}}', \phi_1)$ **then**
  goto *guessing*
else if $(\mathcal{S}, \mathbf{C}, \widehat{\mathcal{S}}') \notin \longrightarrow$ **then**
  goto *guessing*
end if
if $\neg L$ **then**
  guess $\gamma \in \{\mathit{true}, \mathit{false}\}$
  if γ **then**
    $\widehat{\mathcal{S}}_l := \widehat{\mathcal{S}}'$
    $L := \mathit{true}$
  end if
else
  if $\widehat{\mathcal{S}}' = \widehat{\mathcal{S}}_l$ **then**
    return *true*
  end if
end if
$\widehat{\mathcal{S}} := \widehat{\mathcal{S}}'$
goto *guessing*

Algorithm 9 Nondeterministic procedure for checking $\overline{\mathsf{K}}_i\phi_1$ in $\mathcal{S} \in St_{\text{CR-}\mathcal{D}}$

guessing:
guess $\widehat{\mathcal{S}} \in St_{\text{CR-}\mathcal{D}}$
if $\neg(\mathit{label}(\mathcal{S}_{\mathit{init}}, \mathsf{EF}\phi_{\widehat{\mathcal{S}}}) \wedge \mathcal{S} \sim_i \widehat{\mathcal{S}})$ **then**
  goto *guessing*
else
  return *true*
end if

Algorithm 10 Nondeterministic procedure for checking $\overline{\mathrm{C}}_\Gamma \phi_1$ in $\mathcal{S} \in St_{\text{CR-}\mathcal{D}}$

1: $\widehat{\mathcal{S}} := \mathcal{S}$
2: *checking*:
3: **if** *label*$(\widehat{\mathcal{S}}, \phi_1)$ and *label*$(\mathcal{S}_{init}, \mathsf{EF}\phi_{\widehat{\mathcal{S}}})$ **then**
4: **return** *true*
5: **end if**
6: *guessing*:
7: guess $\widehat{\mathcal{S}}' \in St_{\text{CR-}\mathcal{D}}$ and $i \in \Gamma$
8: **if** $\neg(\widehat{\mathcal{S}} \sim_i \widehat{\mathcal{S}}')$ **then**
9: **goto** *guessing*
10: **end if**
11: $\widehat{\mathcal{S}} := \widehat{\mathcal{S}}'$
12: **goto** *checking*

the procedure nondeterministically selects a state $\widehat{\mathcal{S}}_r$ of that sequence. The guessing of the sequence stops when the newly guessed state is $\widehat{\mathcal{S}}_r$. For simplicity of the presentation, this part of the procedure is not included in Algorithm 7 and Algorithm 8. We do not explicitly refer to the context automaton of CR-$\mathcal{D}$ in the algorithms as it does not affect the complexity considerations. However, when selecting a CRDRS state $\widehat{\mathcal{S}}$, in fact, a state of $\mathcal{D}$ and a location of $\mathfrak{E}$ are selected.

To verify if the rsCTLK formula ϕ holds in the model $\mathcal{M}_{\text{CR-}\mathcal{D}}$, the *label* procedure is called for the initial state: $\mathcal{M}_{\text{CR-}\mathcal{D}} \models \phi$ iff *label*$(\mathcal{S}_{init}, \phi)$.

The procedure is called recursively for each subformula of ϕ. At a given recursion level the procedure requires only a constant number of variables to be stored.

The total space requirement depends on $\mathcal{O}(\mathsf{d}(\phi))$ calls of the *label* procedure, where a single call needs space $\mathcal{O}(|S| \cdot |\mathcal{A}|)$. The space requirement for the procedure is not affected by the size of $\mathfrak{sc}$ as it is only used in nondeterministic choices. For each call of the *label* procedure, i.e., for each nesting level of ϕ, the *label* procedure is called recursively at most twice, as each operator of rsCTLK has at most two arguments. Thus, the overall space requirement of the procedure is $\mathcal{O}\big(|S| \cdot |\mathcal{A}| \cdot \mathsf{d}(\phi)\big)$.

Therefore, by Savitch's theorem, the deterministic algorithm can be implemented in polynomial space.

The following theorem follows directly from Lemma 3 and Lemma 4.

Theorem 5. *The rs*CTLK *model checking problem is* PSPACE*-complete.*

6 Boolean encoding

In this section we provide an encoding for the context-restricted distributed reaction systems into Boolean functions intended for symbolic model checking.

Let CR-$\mathcal{D} = (\mathcal{D}, \mathfrak{E})$ where $\mathcal{D} = (S, \{A_i\}_{i \in \mathcal{A}})$ and $\mathfrak{E} = (\mathcal{Q}, \mathfrak{q}^{init}, R)$ is an ECA over $\mathcal{D}$. Then, $\mathcal{M}_{\text{CR-}\mathcal{D}} = (St_{\text{CR-}\mathcal{D}}, \mathcal{S}_{init}, \longrightarrow)$ is the model for CR-$\mathcal{D}$. In the

remainder of this section we describe an encoding of $\mathcal{M}_{\text{CR-}\mathcal{D}}$. The elements of the background set S are denoted by $e_1, \ldots, e_n$ where $n = |S|$. The following sets of propositional variables are used in the encoding:

$$\begin{aligned}
\mathbf{P} &= \{\mathtt{p}_{1,1}, \ldots, \mathtt{p}_{1,n}, \ldots, \mathtt{p}_{m,1}, \ldots, \mathtt{p}_{m,n}\} \\
\mathbf{P}^{\mathcal{E}} &= \{\mathtt{p}^{\mathcal{E}}_{1,1}, \ldots, \mathtt{p}^{\mathcal{E}}_{1,n}, \ldots, \mathtt{p}^{\mathcal{E}}_{m,1}, \ldots, \mathtt{p}^{\mathcal{E}}_{m,n}\} \\
\mathbf{P}' &= \{\mathtt{p}'_{1,1}, \ldots, \mathtt{p}'_{1,n}, \ldots, \mathtt{p}'_{m,1}, \ldots, \mathtt{p}'_{m,n}\} \\
\mathbf{P}^{a} &= \{\mathtt{p}^{a}_{1}, \ldots, \mathtt{p}^{a}_{m}\}
\end{aligned}$$

The variables of $\mathbf{P}$ are used to encode the states of $\mathcal{D}$. The actions representing context entities are encoded with the variables of $\mathbf{P}^{\mathcal{E}}$. To encode the successors in the transition relation, we use the primed variables of $\mathbf{P}'$. To distinguish active and inactive agents in each step of the execution we use the set $\mathbf{P}^{a}$ of the *activity* variables. The states of $\mathfrak{E}$ are encoded using $n_{\mathfrak{E}} = \lceil \log_2 |\mathcal{Q}| \rceil$ variables. Therefore, for the encoding of the locations of $\mathfrak{E}$ we use the set $\mathbf{Q} = \{\mathtt{q}_1, \ldots, \mathtt{q}_{n_{\mathfrak{E}}}\}$. To encode the successors in the transition relation of $\mathfrak{E}$ we use the set $\mathbf{Q}' = \{\mathtt{q}'_1, \ldots, \mathtt{q}'_{n_{\mathfrak{E}}}\}$ of the primed variables of $\mathbf{Q}$. The set of the reactions of the j^{th} component that produce $e \in S$ is defined as $Prod_j(e) = \{a \in A_j \mid e \in P_a\}$, where $j \in \{1, \ldots, m\}$. Additionally, we introduce the following vectors of variables:

$$\begin{aligned}
\overline{\mathbf{p}} &= (\mathtt{p}_{1,1}, \ldots, \mathtt{p}_{1,n}, \ldots, \mathtt{p}_{m,1}, \ldots, \mathtt{p}_{m,n}) \\
\overline{\mathbf{p}}^{\mathcal{E}} &= (\mathtt{p}^{\mathcal{E}}_{1,1}, \ldots, \mathtt{p}^{\mathcal{E}}_{1,n}, \ldots, \mathtt{p}^{\mathcal{E}}_{m,1}, \ldots, \mathtt{p}^{\mathcal{E}}_{m,n}) \\
\overline{\mathbf{p}}' &= (\mathtt{p}'_{1,1}, \ldots, \mathtt{p}'_{1,n}, \ldots, \mathtt{p}'_{m,1}, \ldots, \mathtt{p}'_{m,n}) \\
\overline{\mathbf{p}}^{a} &= (\mathtt{p}^{a}_{1}, \ldots, \mathtt{p}^{a}_{m})
\end{aligned}$$

Moreover, we define the following functions: $\mathtt{m} : S \to \mathbf{P}$, $\mathtt{m}' : S \to \mathbf{P}'$, $\mathtt{m}^{\mathcal{E}} : S \to \mathbf{P}^{\mathcal{E}}$, such that $\mathtt{m}(e_i) = \mathtt{p}_i$, $\mathtt{m}'(e_i) = \mathtt{p}'_i$, $\mathtt{m}^{\mathcal{E}}(e_i) = \mathtt{p}^{\mathcal{E}}_i$, for all $1 \leq i \leq n$. These functions map the background set entities to the corresponding variables of the encoding. For the encoding of $\mathfrak{E}$ we use two additional vectors of variables: $\overline{\mathbf{q}} = (\mathtt{q}_1, \ldots, \mathtt{q}_{n_{ca}})$, $\overline{\mathbf{q}}' = (\mathtt{q}'_1, \ldots, \mathtt{q}_{n_{ca}})$. To denote an encoded location $\mathtt{q} \in \mathcal{Q}$ of $\mathfrak{E}$ we write $\mathtt{e}(\mathtt{q})$, and to denote its primed encoding used to encode that $\mathtt{q}$ is a successor we write $\mathtt{e}'(\mathtt{q})$. Then, we introduce functions $\mathtt{e} : \mathcal{Q} \to \mathfrak{B}(\mathbf{Q})$ and $\mathtt{e}' : \mathcal{Q} \to \mathfrak{B}(\mathbf{Q}')$ which map the states of $\mathfrak{E}$ and their successors to their encodings using unprimed and primed variables, respectively. To encode the state constraints which serve as transition guards of $\mathfrak{E}$ we introduce a function $\mathtt{esc} : \mathcal{SC}_{\mathcal{D}} \to \mathfrak{B}(\mathbf{P})$ which maps the state constraints to their corresponding Boolean encodings over the set $\mathbf{P}$ of unprimed variables. We do not define the function explicitly as its encoding is straighforward.

Single state. A state $\mathcal{S} \in St_{\text{CR-}\mathcal{D}}$ is encoded as the conjunction of all the variables corresponding to the entities that are in $\mathcal{S}$, and the conjunction of all the negations of the variables that are not in $\mathcal{S}$:

$$\mathsf{St}_{\mathcal{S}}(\overline{\mathbf{p}}) = \bigwedge_{1 \leq i \leq m} \left(\left(\bigwedge_{e \in \mathcal{S}[i]} \mathtt{m}_i(e) \right) \wedge \left(\bigwedge_{e \in (S \setminus \mathcal{S}[i])} \neg \mathtt{m}_i(e) \right) \right).$$

Sets of states. A set $W \subseteq St_{\text{CR-}\mathcal{D}}$ is encoded as the disjunction of all the encoded states that are in $St_{\text{CR-}\mathcal{D}}$:

$$\mathsf{SetSt}_W(\overline{\mathtt{p}}) = \bigvee_{\mathcal{S} \in W} \mathsf{St}_{\mathcal{S}}(\overline{\mathtt{p}}).$$

The sets of states are not directly encoded in the translation – we provide their encoding only for completeness.

Contexts. A context $\mathbf{C} \subseteq Ct_{\mathcal{D}}$ is encoded as follows:

$$\mathsf{Ct}_{\mathbf{C}}(\overline{\mathtt{p}}^{\mathcal{E}}, \overline{\mathtt{p}}^{a}) = \bigwedge_{1 \leq i \leq m} \left(\left(\bigwedge_{e \in \mathbf{C}^c[i]} \mathtt{m}_i^{\mathcal{E}}(e) \right) \wedge \left(\bigwedge_{e \in (S \setminus \mathbf{C}^c[i])} \neg \mathtt{m}_i^{\mathcal{E}}(e) \right) \right) \wedge \left(\bigwedge_{i \in \mathbf{C}^a} \mathtt{p}_i^a \right) \wedge \left(\bigwedge_{i \in (\mathcal{A} \setminus \mathbf{C}^a)} \neg \mathtt{p}_i^a \right).$$

Entity production condition. Let $j \in \mathcal{A}$. Then, a single entity $e \in S$ can be produced in the j^{th} agent if there exists a reaction $a \in Prod_j(e)$ that is enabled, that is, all of the variables corresponding to reactants of a (including the variables representing the context) evaluate to *true* and all of the variables corresponding to inhibitors of a (also including the variables representing the context) are *false*. For the j^{th} agent the formula encoding this is defined as follows:

– if $Prod_j(e) \neq \emptyset$, then

$$\mathsf{En}_e^j\big(\overline{\mathtt{p}}, \overline{\mathtt{p}}^{\mathcal{E}}, \overline{\mathtt{p}}^{a}\big) = \bigvee_{b \in Prod_j(e)} \left(\bigwedge_{e' \in R_b} \left(\mathtt{m}_j^{\mathcal{E}}(e') \vee \bigvee_{1 \leq i \leq m} (\mathtt{m}_i(e') \wedge \mathtt{p}_i^a) \right) \right) \wedge \left(\bigwedge_{e' \in I_b} \left(\neg \mathtt{m}_j^{\mathcal{E}}(e') \wedge \bigwedge_{1 \leq i \leq m} (\neg \mathtt{m}_i(e') \vee \neg \mathtt{p}_i^a) \right) \right)$$

– if $Prod_j(e) = \emptyset$, then

$$\mathsf{En}_e^j\big(\overline{\mathtt{p}}, \overline{\mathtt{p}}^{\mathcal{E}}, \overline{\mathtt{p}}^{a}\big) = \mathit{false}.$$

Entity production. For $j \in \mathcal{A}$, the function $\mathsf{Pr}_e\big(\overline{\mathtt{p}}, \overline{\mathtt{p}}^{\mathcal{E}}, \overline{\mathtt{p}}'\big)$ encodes the production of a single entity $e \in S$ in the j^{th} agent. If the j^{th} agent is active, i.e., $\mathtt{p}_j^a$ holds, and the production of e is enabled, then the variable corresponding to e in the successor state is required to be *true*. If the production of e is not enabled, then the variable corresponding to e in the successor state is required to be *false*. If

the j^{th} agent is inactive, i.e., $\mathtt{p}_j^a$ does not hold, then the presence of e in the successor state is encoded to remain unchanged.

$$\mathsf{Pr}_e^j(\overline{\mathtt{p}}, \overline{\mathtt{p}}^{\mathcal{E}}, \overline{\mathtt{p}}^a, \overline{\mathtt{p}}') = \Bigg(\mathtt{p}_j^a \wedge \Big(\big(\mathsf{En}_e^j(\overline{\mathtt{p}}, \overline{\mathtt{p}}^{\mathcal{E}}, \overline{\mathtt{p}}^a) \wedge \mathtt{m}'(e)\big) \vee \big(\neg\mathsf{En}_e^j(\overline{\mathtt{p}}, \overline{\mathtt{p}}^{\mathcal{E}}, \overline{\mathtt{p}}^a) \wedge \neg\mathtt{m}'(e)\big)\Big)\Bigg) \vee \Big(\neg\mathtt{p}_j^a \wedge \big(\mathtt{m}_j(e) \leftrightarrow \mathtt{m}'_j(e)\big)\Big).$$

Transition relation for reactions. To encode the state changes introduced by the reactions of $\mathcal{D}$ we define a function that is the conjunction of the entity production encodings for all the background set entities and restricts the allowed context sets.

$$\mathsf{Tr}_{\mathcal{D}}(\overline{\mathtt{p}}, \overline{\mathtt{p}}^{\mathcal{E}}, \overline{\mathtt{p}}^a, \overline{\mathtt{p}}') = \bigwedge_{\substack{1 \le j \le m, \\ e \in S}} \mathsf{Pr}_e^j(\overline{\mathtt{p}}, \overline{\mathtt{p}}^{\mathcal{E}}, \overline{\mathtt{p}}^a, \overline{\mathtt{p}}').$$

Transition relation for context automaton. The encoding of the transition relation of $\mathfrak{E}$ is a disjunction of the encodings for each transition.

$$\mathsf{Tr}_{\mathfrak{E}}(\overline{\mathtt{q}}, \overline{\mathtt{p}}^{\mathcal{E}}, \overline{\mathtt{p}}^a, \overline{\mathtt{q}}') = \bigvee_{(\mathtt{q}, \mathfrak{sc}, \mathbf{C}, \mathtt{q}') \in R} \Big(\mathsf{e}(\mathtt{q}) \wedge \mathsf{esc}(\mathfrak{sc}) \wedge \mathsf{Ct}_{\mathbf{C}}(\overline{\mathtt{p}}^{\mathcal{E}}, \overline{\mathtt{p}}^a) \wedge \mathsf{e}'(\mathtt{q}') \Big).$$

Global transition relation. To encode the transition relation of $\mathcal{M}_{\text{CR-}\mathcal{D}}$ we build a conjunction of the transition relation for reactions and the transition relation for context automaton providing the conditions for the active components and the context sets.

$$\mathsf{Tr}\big((\overline{\mathtt{p}}, \overline{\mathtt{q}}), \overline{\mathtt{p}}^{\mathcal{E}}, \overline{\mathtt{p}}^a, (\overline{\mathtt{p}}', \overline{\mathtt{q}}')\big) = \mathsf{Tr}_{\mathcal{D}}(\overline{\mathtt{p}}, \overline{\mathtt{p}}^{\mathcal{E}}, \overline{\mathtt{p}}^a, \overline{\mathtt{p}}') \wedge \mathsf{Tr}_{\mathfrak{E}}(\overline{\mathtt{q}}, \overline{\mathtt{p}}^{\mathcal{E}}, \overline{\mathtt{p}}^a, \overline{\mathtt{q}}').$$

7 Experimental results

The proposed model checking method has been implemented in ReactICS – our reaction systems model checking toolkit. The implementation uses C++ as the implementation language and the CUDD library [35] for creating binary decision diagrams and handling the operations on them. The tool uses the encoding presented in Section 6 and implements the algorithms from Section 5.

In the experimental evaluation we use our implementation with the parameters summarised in Table 1.

Paritioning of transition relation. The encoded transition relation of the verified system can be stored and applied in two different ways: by using a *monolithic* or *partitioned* encoding [6]. When computing successors (or predecessors) of a set of states, we compute the conjunction of the formula encoding

parameter	description
x	partitioned transition relation
z	reordering of BDD variables
b	bounded model checking heuristic disabled

Table 1. Parameters of the model checking tool

the set of states and of the formula encoding the transition relation. In the case of the monolithic encoding, there is only one BDD encoding the transition relation, while in the partitioned encoding for each agent we store a separate decision diagram encoding its transition relation, and the conjunction is calculated on the fly.

BDD reordering. The size of a BDD depends on the selected order of the Boolean variables, thus in most cases it will have a significantly impact on the performance. We apply two different approaches to ordering the variables: a fixed interleaving order and an automatic reordering of the variables. For the fixed order we apply the interleaving order, where primed and the corresponding unprimed BDD variables are interleaved. While in the case of the automatic reordering we attempt to adapt the order of the variables with each iteration of the algorithm extending the BDD for the reachable states and the transition relation. To this aim we use the group sifting algorithm of [28] implemented in the CUDD library.

Bounded model checking. The implementation uses the BDD-based bounded model checking heuristic [7] for testing the existential rsCTLK formulae. This allows for early termination of the verification when a witness for the verified formula is found. The approach consists in verifying the formula at each iteration of the algorithm computing the set of the reachable states.

7.1 Train-gate-controller

We test our implementation by verifying the following properties of the train-gate-controller system from Example 3 in all the configurations of the described

implementation parameters:

$$\phi_1 = \bigwedge_{i \in \mathcal{A}} \mathsf{EF}\left(\mathsf{E}_{i.allowed}\mathsf{X}\left(i.in\right)\right),$$

$$\phi_2 = \mathsf{EF}\left(\bigwedge_{i \in \mathcal{A}} i.approach\right),$$

$$\phi_3 = \mathsf{AG}\left(1.in \implies \mathrm{K}_1\left(\bigwedge_{\substack{j \in \mathcal{A}, \\ j>1}} \neg j.in\right)\right),$$

$$\phi_4 = \mathsf{AG}\left(1.in \implies \mathrm{C}_{\mathcal{A}}\left(\bigwedge_{\substack{j \in \mathcal{A}, \\ j>1}} \neg j.in\right)\right).$$

The experimental results are presented in Fig. 1–8. For each scaling parameter and formula we set an execution time limit, i.e., the verification is allowed to run for at most 5 minutes and then the process is terminated. The results are also summarised in Tab. 2–5. Since in our experiments we consider eight different configurations of the model checking tool which might affect the readability of the graphs, we additionally provide the detailed results in Tab. 2–5.

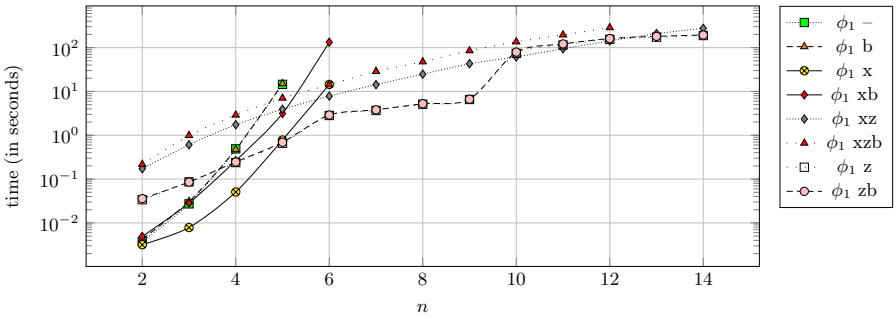

Fig. 1. Verification results for TGC and ϕ_1: execution time

7.2 Distributed abstract pipeline

Here we introduce distributed abstract pipeline (DAP) system similar to the abstract pipeline system introduced in [25].

The background set is defined as $S = \{a, b, c, d, y, dy, r\}$. The system consists of m agents producing dy from a. For dy to be produced, a sequence of reactions must take place and for the sequence to be activated the entity a needs to be provided. The entity a is provided to the i^{th} agent by the context automaton

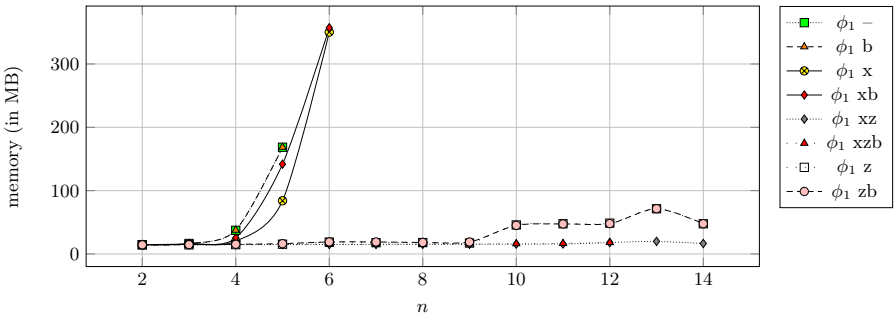

Fig. 2. Verification results for TGC and ϕ_1: memory consumption

	-		b		x		xb		xz		xzb		z		zb	
	time	mem	time	mem	time	mem	time	mem	time	mem	time	mem	time	mem	time	mem
2	0.0	14.58	0.0	14.5	0.0	14.41	0.0	14.73	0.17	14.44	0.22	14.59	0.03	14.4	0.04	14.41
3	0.03	16.51	0.03	16.32	0.01	15.25	0.03	16.25	0.61	14.63	0.99	14.74	0.09	14.38	0.09	14.72
4	0.49	37.31	0.47	37.54	0.05	20.09	0.26	25.38	1.74	14.7	2.9	14.82	0.24	14.72	0.25	15.29
5	14.51	168.5	15.04	168.5	0.78	83.98	3.15	141.8	3.93	14.85	7.05	15.16	0.66	15.41	0.69	16.2
6	-	-	-	-	14.46	350.3	132.8	357.2	7.84	14.96	14.22	15.31	2.83	18.76	2.9	19.01
7	-	-	-	-	-	-	-	-	14.3	15.05	28.69	15.38	3.68	17.87	3.81	18.92
8	-	-	-	-	-	-	-	-	24.8	15.25	47.5	15.43	5.1	18.02	5.24	18.18
9	-	-	-	-	-	-	-	-	42.85	15.29	85.05	15.84	6.55	17.21	6.63	18.91
10	-	-	-	-	-	-	-	-	61.4	15.48	135.6	16.25	73.78	45.81	78.98	45.2
11	-	-	-	-	-	-	-	-	94.47	15.78	194.9	16.59	114.5	47.7	121.3	47.41
12	-	-	-	-	-	-	-	-	142.3	18.01	285.8	18.09	154.8	48.65	160.7	47.95
13	-	-	-	-	-	-	-	-	209.1	19.88	-	-	172.9	71.34	180.8	71.45
14	-	-	-	-	-	-	-	-	274.8	16.47	-	-	188.5	47.64	194.0	47.81

Table 2. Verification results for TGC and ϕ_1

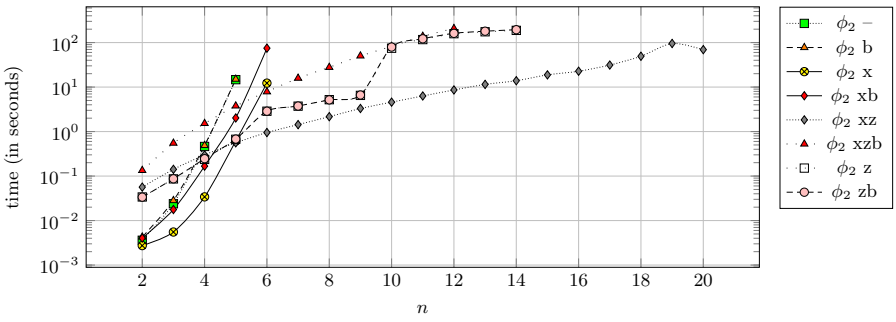

Fig. 3. Verification results for TGC and ϕ_2: execution time

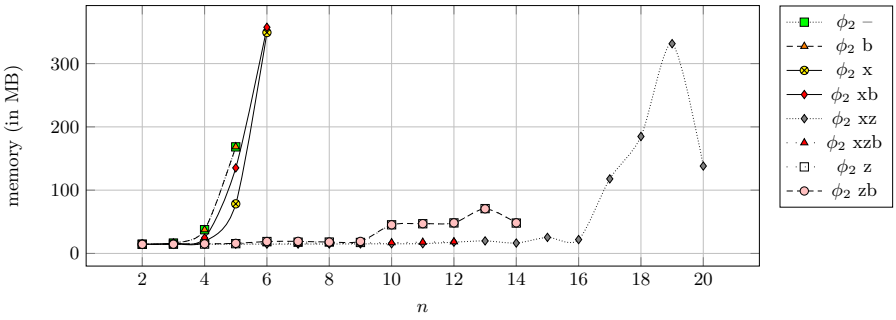

Fig. 4. Verification results for TGC and ϕ_2: memory consumption

	-		b		x		xb		xz		xzb		z		zb	
	time	mem	time	mem	time	mem	time	mem	time	mem	time	mem	time	mem	time	mem
2	0.0	14.49	0.0	14.59	0.0	14.46	0.0	14.55	0.06	14.31	0.13	14.51	0.03	14.41	0.03	14.43
3	0.02	16.39	0.03	16.39	0.01	14.96	0.02	16.08	0.14	14.47	0.55	14.66	0.09	14.55	0.09	14.61
4	0.46	37.53	0.49	37.48	0.03	18.61	0.17	24.45	0.3	14.62	1.52	14.81	0.24	14.77	0.25	15.12
5	14.67	168.4	15.05	168.6	0.61	78.36	2.02	135.2	0.56	14.64	3.77	15.23	0.67	15.46	0.68	15.88
6	-	-	-	-	12.27	349.1	74.86	357.4	0.95	14.69	7.84	15.39	2.87	18.78	2.87	18.84
7	-	-	-	-	-	-	-	-	1.43	14.76	15.83	15.31	3.73	17.81	3.75	19.08
8	-	-	-	-	-	-	-	-	2.16	14.88	27.78	15.7	5.21	17.98	5.16	17.97
9	-	-	-	-	-	-	-	-	3.3	15.06	50.14	16.49	6.51	17.33	6.63	19.0
10	-	-	-	-	-	-	-	-	4.58	15.27	81.99	17.04	74.72	45.21	79.36	45.21
11	-	-	-	-	-	-	-	-	6.31	15.42	138.6	17.79	116.8	46.89	121.6	47.01
12	-	-	-	-	-	-	-	-	8.62	17.75	207.2	18.38	158.5	48.22	161.3	48.3
13	-	-	-	-	-	-	-	-	11.48	19.8	-	-	174.9	70.82	180.3	70.5
14	-	-	-	-	-	-	-	-	13.9	16.4	-	-	188.1	47.99	194.5	48.35
15	-	-	-	-	-	-	-	-	18.8	25.36	-	-	-	-	-	-
16	-	-	-	-	-	-	-	-	22.71	21.89	-	-	-	-	-	-
17	-	-	-	-	-	-	-	-	31.24	117.8	-	-	-	-	-	-
18	-	-	-	-	-	-	-	-	49.24	184.8	-	-	-	-	-	-
19	-	-	-	-	-	-	-	-	95.46	331.5	-	-	-	-	-	-
20	-	-	-	-	-	-	-	-	69.38	138.1	-	-	-	-	-	-

Table 3. Verification results for TGC and ϕ_2

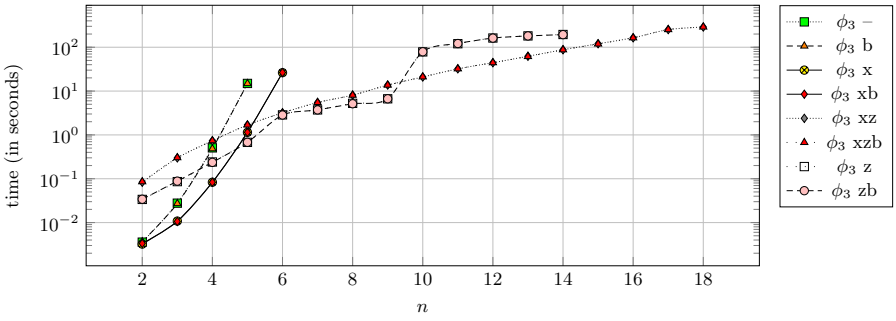

Fig. 5. Verification results for TGC and ϕ_3: execution time

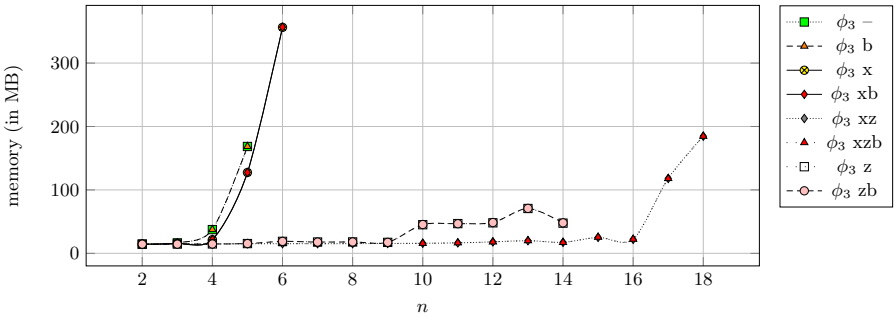

Fig. 6. Memory consumption for TGC and ϕ_3

	-		b		x		xb		xz		xzb		z		zb	
	time	mem	time	mem	time	mem	time	mem	time	mem	time	mem	time	mem	time	mem
2	0.0	14.59	0.0	14.59	0.0	14.52	0.0	14.52	0.08	14.41	0.09	14.53	0.03	14.45	0.03	14.45
3	0.03	16.49	0.03	16.45	0.01	15.57	0.01	15.43	0.3	14.64	0.3	14.63	0.09	14.58	0.09	14.53
4	0.51	37.44	0.48	37.48	0.08	21.62	0.08	21.64	0.74	14.82	0.74	14.74	0.24	14.81	0.24	14.74
5	14.94	168.5	14.86	168.5	1.15	127.5	1.15	127.5	1.69	15.11	1.69	15.26	0.68	15.41	0.68	15.4
6	-	-	-	-	26.32	356.3	26.48	356.7	3.19	15.25	3.2	15.23	2.86	18.89	2.84	18.83
7	-	-	-	-	-	-	-	-	5.48	15.2	5.5	15.2	3.72	17.87	3.73	17.75
8	-	-	-	-	-	-	-	-	8.09	15.29	8.08	15.32	5.25	18.02	5.14	17.94
9	-	-	-	-	-	-	-	-	13.72	15.55	13.69	15.6	6.66	17.28	6.67	17.35
10	-	-	-	-	-	-	-	-	20.91	15.84	21.09	15.85	78.34	45.12	78.85	45.41
11	-	-	-	-	-	-	-	-	32.18	16.51	32.19	16.6	120.2	46.9	121.5	46.72
12	-	-	-	-	-	-	-	-	44.32	18.03	44.39	17.97	162.3	48.11	161.6	48.54
13	-	-	-	-	-	-	-	-	62.25	19.98	61.44	19.95	181.3	70.71	181.3	70.87
14	-	-	-	-	-	-	-	-	87.65	17.35	87.47	17.24	195.3	47.6	193.6	48.13
15	-	-	-	-	-	-	-	-	120.3	25.23	118.7	25.47	-	-	-	-
16	-	-	-	-	-	-	-	-	164.2	22.05	164.1	22.29	-	-	-	-
17	-	-	-	-	-	-	-	-	252.9	118.1	252.7	118.0	-	-	-	-
18	-	-	-	-	-	-	-	-	288.9	184.7	289.8	184.7	-	-	-	-

Table 4. Verification results for TGC and ϕ_3

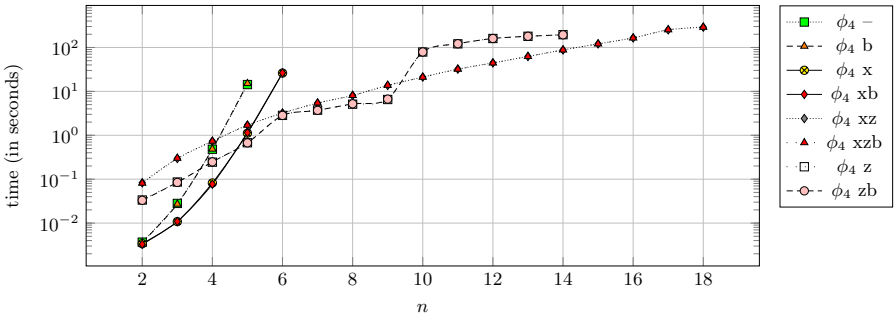

Fig. 7. Verification results for TGC and ϕ_4: execution time

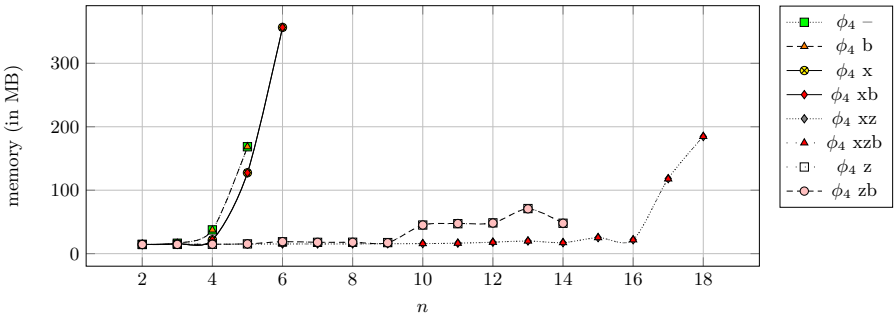

Fig. 8. Verification results for TGC and ϕ_4: memory consumption

	-		b		x		xb		xz		xzb		z		zb	
	time	mem	time	mem	time	mem	time	mem	time	mem	time	mem	time	mem	time	mem
2	0.0	14.59	0.0	14.47	0.0	14.52	0.0	14.56	0.08	14.46	0.08	14.49	0.03	14.45	0.03	14.45
3	0.03	16.44	0.03	16.39	0.01	15.57	0.01	15.52	0.3	14.62	0.3	14.64	0.09	14.6	0.08	14.63
4	0.48	37.47	0.49	37.37	0.08	21.84	0.08	21.63	0.73	14.82	0.73	14.73	0.24	14.82	0.25	14.85
5	14.3	168.5	15.14	168.5	1.12	127.5	1.15	127.5	1.7	15.13	1.7	15.13	0.68	15.38	0.67	15.4
6	-	-	-	-	26.16	356.6	26.38	356.5	3.21	15.25	3.19	15.23	2.84	18.71	2.84	18.89
7	-	-	-	-	-	-	-	-	5.41	15.27	5.46	15.25	3.74	17.81	3.7	17.87
8	-	-	-	-	-	-	-	-	8.09	15.34	8.1	15.28	5.11	17.99	5.21	17.9
9	-	-	-	-	-	-	-	-	13.64	15.63	13.8	15.67	6.54	17.44	6.67	17.21
10	-	-	-	-	-	-	-	-	21.18	15.82	21.07	15.84	78.84	45.1	79.17	45.12
11	-	-	-	-	-	-	-	-	32.14	16.54	32.08	16.47	120.9	47.2	121.7	47.51
12	-	-	-	-	-	-	-	-	44.49	18.03	44.42	17.96	160.2	48.54	160.3	48.27
13	-	-	-	-	-	-	-	-	62.43	19.94	62.41	19.9	180.0	71.06	180.0	70.65
14	-	-	-	-	-	-	-	-	87.99	17.32	87.84	17.38	195.6	47.92	194.3	47.93
15	-	-	-	-	-	-	-	-	120.8	25.29	120.6	25.23	-	-	-	-
16	-	-	-	-	-	-	-	-	165.4	21.86	164.8	21.77	-	-	-	-
17	-	-	-	-	-	-	-	-	252.5	118.2	253.3	117.8	-	-	-	-
18	-	-	-	-	-	-	-	-	290.4	184.6	289.8	184.8	-	-	-	-

Table 5. Verification results for TGC and ϕ_4

when dy is produced in the $(i-1)^{\text{th}}$ agent, or when $i = 1$ and the context automaton is in the initial location. This means that the agents are activated sequentially. The set of agents is defined as $\mathcal{A} = \{1, \ldots, n\}$, where $m = n - 1$ is the last agent producing dy and n is the receiver of the final entity r. The set A_i for $i \in \{1, \ldots, m\}$ consists of the following reactions:

- $(\{a\}, \{s\}, \{y\})$,
- $(\{y\}, \{s\}, \{y\})$,
- $(\{a\}, \{s\}, \{b\})$,
- $(\{b\}, \{s\}, \{c\})$,
- $(\{c\}, \{s\}, \{d\})$,
- $(\{d, y\}, \{s\}, \{dy\})$.

The receiver agent has only one reaction: $A_n = \{(\{r\}, \{s\}, \{r\})\}$. We define $\mathfrak{E}_{dap} = (\mathcal{Q}, \mathfrak{q}_0, R)$ such that $\mathcal{Q} = \{\mathfrak{q}_0, \mathfrak{q}_1\}$, and the set R consists of the following transitions:

1. $\mathfrak{q}_0 \xrightarrow{true,((\{a\},\varnothing,\ldots,\varnothing),\{1\})} \mathfrak{q}_1$,
2. $\mathfrak{q}_1 \xrightarrow{\neg i.dy,((\varnothing,\ldots,\varnothing),\{i\})} \mathfrak{q}_1$ for all $i \in \{1, \ldots, m\}$.
3. for each $i \in \{2, \ldots, m\}$: $\mathfrak{q}_1 \xrightarrow{(i-1).dy,\mathbf{C}} \mathfrak{q}_1$, where: $\mathbf{C}^a = \{i\}$ and for each $j \in \mathcal{A}$:
$$\mathbf{C}^c[j] = \begin{cases} \{a\} & j = i, \\ \varnothing & j \in \mathcal{A} \setminus \{i\}, \end{cases}$$
4. $\mathfrak{q}_1 \xrightarrow{(1.dy)\wedge\ldots\wedge(m.dy),((\varnothing,\ldots,\varnothing,\{r\}),\{n\})} \mathfrak{q}_1$.

Finally, we define CR-$\mathcal{D}_{dap} = ((S, \{A_i\}_{i\in\mathcal{A}}), \mathfrak{E}_{dap})$. We test the following rsCTLK formulae:

$$\phi_1 = \mathsf{EF}\,(n.r)$$
$$\phi_2 = \mathsf{AG}\,(m.d \implies \mathrm{K}_m\,((m-1)\,.y))$$

The formula ϕ_1 expresses the possibility of producing the final entity by the receiver. The formula ϕ_2 means that for all the paths when m has d in its local state, then m knows that $(m-1)$ has y.

The experimental results are presented in Fig. 9–12 and Tab. 6–7. The execution time limit was set to 10 minutes.

7.3 Observations

The experimental results for TGC demonstrate the benefits of using automatic reordering of BDD variables and partitioned transition relation. These heuristics led to smaller decision diagrams which results in lower memory consumption and operations on the diagrams being more efficient. For TGC, in most cases the verification is the most efficient with xz. However, for ϕ_1 using partitioned transition relation results in longer execution times than using only BDD reordering.

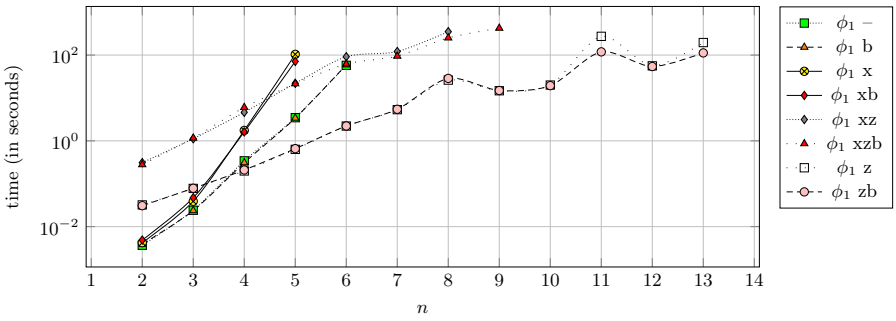

Fig. 9. Verification results for DAP and ϕ_1: execution time

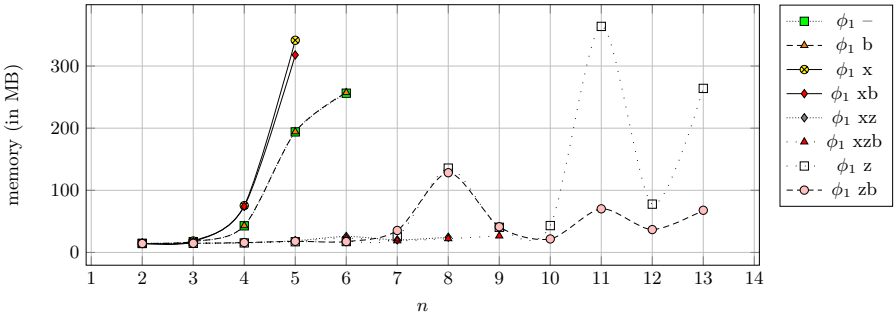

Fig. 10. Verification results for DAP and ϕ_1: memory consumption

	-		b		x		xb		xz		xzb		z		zb	
	time	mem	time	mem	time	mem	time	mem	time	mem	time	mem	time	mem	time	mem
2	0.0	14.5	0.0	14.53	0.0	14.53	0.0	14.71	0.31	14.52	0.28	14.64	0.03	14.43	0.03	14.36
3	0.02	17.42	0.02	17.44	0.04	18.43	0.05	17.33	1.12	14.69	1.16	14.72	0.08	14.61	0.08	14.73
4	0.34	42.84	0.31	43.07	1.75	75.33	1.62	74.25	4.63	15.47	6.03	15.55	0.2	15.32	0.21	15.81
5	3.47	194.0	3.39	194.2	104.0	341.5	71.08	317.8	21.96	18.79	21.47	18.25	0.63	17.2	0.66	17.9
6	57.69	256.3	59.28	257.3	-	-	-	-	91.59	25.73	61.98	22.84	2.22	17.5	2.22	17.47
7	-	-	-	-	-	-	-	-	120.9	20.19	94.46	18.79	5.34	25.18	5.45	35.35
8	-	-	-	-	-	-	-	-	351.2	24.46	251.4	22.18	26.09	135.7	28.6	128.3
9	-	-	-	-	-	-	-	-	-	-	423.9	26.61	14.73	40.47	14.93	41.32
10	-	-	-	-	-	-	-	-	-	-	-	-	20.01	43.21	19.39	21.69
11	-	-	-	-	-	-	-	-	-	-	-	-	273.6	363.8	118.8	70.17
12	-	-	-	-	-	-	-	-	-	-	-	-	56.05	77.72	54.08	36.77
13	-	-	-	-	-	-	-	-	-	-	-	-	195.9	264.0	112.4	67.86

Table 6. Verification results for DAP and ϕ_1

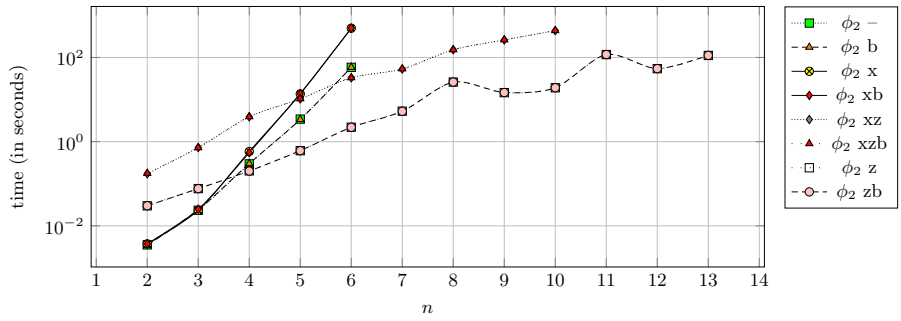

Fig. 11. Verification results for DAP and ϕ_2: execution time

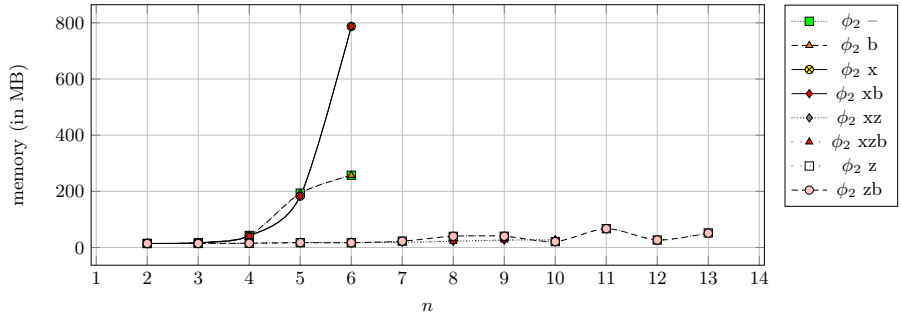

Fig. 12. Verification results for DAP and ϕ_2: memory consumption

	-		b		x		xb		xz		xzb		z		zb	
	time	mem	time	mem	time	mem	time	mem	time	mem	time	mem	time	mem	time	mem
2	0.0	14.62	0.0	14.62	0.0	14.47	0.0	14.59	0.17	14.54	0.18	14.49	0.03	14.33	0.03	14.38
3	0.02	17.46	0.02	17.32	0.02	16.8	0.02	16.74	0.73	14.78	0.72	14.82	0.08	14.53	0.08	14.53
4	0.3	42.87	0.3	42.92	0.58	40.98	0.55	41.01	3.9	15.53	3.92	15.47	0.2	15.17	0.2	15.14
5	3.43	193.6	3.36	193.3	13.65	183.1	13.84	183.2	10.39	17.85	10.46	17.97	0.61	17.28	0.61	17.33
6	58.39	257.4	58.42	257.1	495.7	787.3	496.6	787.3	32.97	17.87	33.07	18.06	2.2	16.93	2.19	17.02
7	-	-	-	-	-	-	-	-	53.05	18.71	53.82	18.66	5.31	22.89	5.26	22.66
8	-	-	-	-	-	-	-	-	153.0	22.13	152.7	22.44	26.04	39.76	26.08	40.11
9	-	-	-	-	-	-	-	-	260.9	26.56	263.0	27.03	14.62	40.21	14.7	40.05
10	-	-	-	-	-	-	-	-	434.9	27.29	435.0	27.02	19.06	21.1	19.11	21.1
11	-	-	-	-	-	-	-	-	-	-	-	-	117.2	67.03	117.5	67.08
12	-	-	-	-	-	-	-	-	-	-	-	-	54.19	26.71	54.3	26.68
13	-	-	-	-	-	-	-	-	-	-	-	-	111.6	51.94	112.0	51.36

Table 7. Verification results for DAP and ϕ_2

The benefit of using the BMC approach can be observed when verifying existential rsCTLK formulae, e.g., in the results for TGC and ϕ_2. However, for ϕ_1 of DAP the BMC heuristic is inefficient, which follows from the fact that the formula is existential and for its witness to be obtained the entire model needs to be explored. This results in redundant checks at each step of unfolding of the model, i.e., when calculating the set of reachable states of the model. Therefore, when the BMC heuristic is disabled for ϕ_1, the execution times and memory consumption are improved. For DAP, reoredering of BDD variables is the most efficient and using partitioning of the transition relation results in longer execution times.

8 Final remarks

We have introduced a generalisation of reaction systems which allows for modelling of distributed systems. We have also extended the notion of context automata which model the environment of reaction systems, to allow for two-way communication, i.e., making the behaviour of the environment also depend on the state of the reaction system.

The introduced formalism can be used for modelling of multi-agent systems. To allow for expressing their temporal and epistemic properties, we introduced rsCTLK which is a logic extending rsCTL with epistemic operators.

For the introduced formalisms we described a symbolic model checking method based on binary decision diagrams. The approach was implemented and evaluated experimentally on two scalable benchmarks.

The formalism of distributed reaction systems and the verification method could be generalised to allow for direct modelling of concentration levels. However, to achieve performance gains, instead of binary decision diagrams a type of decision diagrams which allow for assigning integers to valuations should be used. The provided encoding can also be used in SAT-based bounded model checking.

References

1. Artiom Alhazov. P systems without multiplicities of symbol-objects. *Information Processing Letters*, 100(3):124 – 129, 2006.
2. Sepinoud Azimi, Cristian Gratie, Sergiu Ivanov, Luca Manzoni, Ion Petre, and Antonio E. Porreca. Complexity of model checking for reaction systems. Technical Report 1122, TUCS, October 2014.
3. Sepinoud Azimi, Cristian Gratie, Sergiu Ivanov, and Ion Petre. Dependency graphs and mass conservation in reaction systems. *Theoretical Computer Science*, 598:23–39, 2015.
4. Sepinoud Azimi, Charmi Panchal, Eugen Czeizler, and Ion Petre. Reaction systems models for the self-assembly of intermediate filaments. *Annals of University of Bucharest*, LXII(2):9–24, 2015.
5. Robert Brijder, Andrzej Ehrenfeucht, and Grzegorz Rozenberg. Reaction systems with duration. In *Computation, cooperation, and life*, pages 191–202. Springer, 2011.

6. J. R. Burch, E. M. Clarke, and D. E. Long. Symbolic model checking with partitioned transition relations. pages 49–58. North-Holland, 1991.
7. G. Cabodi, P. Camurati, and S. Quer. Can BDD compete with SAT solvers on bounded model checking? In *Proc. of the 39th Design Automation Conference (DAC'02)*, pages 117–122, 2002.
8. Luca Corolli, Carlo Maj, Fabrizio Marini, Daniela Besozzi, and Giancarlo Mauri. An excursion in reaction systems: From computer science to biology. *Theoretical computer science*, 454:95–108, 2012.
9. Alberto Dennunzio, Enrico Formenti, and Luca Manzoni. Extremal combinatorics of reaction systems. In *International Conference on Language and Automata Theory and Applications*, pages 297–307. Springer, 2014.
10. Andrzej Ehrenfeucht, Jetty Kleijn, Maciej Koutny, and Grzegorz Rozenberg. Reaction systems: A natural computing approach to the functioning of living cells. In *A Computable Universe: Understanding and Exploring Nature as Computation*, pages 189–208. World Scientific, 2013.
11. Andrzej Ehrenfeucht, Jetty Kleijn, Maciej Koutny, and Grzegorz Rozenberg. Evolving reaction systems. *Theoretical Computer Science*, 682:79–99, 2017.
12. Andrzej Ehrenfeucht, Michael Main, and Grzegorz Rozenberg. Combinatorics of life and death for reaction systems. *International Journal of Foundations of Computer Science*, 21(03):345–356, 2010.
13. Andrzej Ehrenfeucht, Michael G. Main, and Grzegorz Rozenberg. Functions defined by reaction systems. *Int. J. Found. Comput. Sci.*, 22(1):167–178, 2011.
14. Andrzej Ehrenfeucht and Grzegorz Rozenberg. Events and modules in reaction systems. *Theoretical Computer Science*, 376(1-2):3–16, 2007.
15. Andrzej Ehrenfeucht and Grzegorz Rozenberg. Reaction systems. *Fundamenta Informaticae*, 75(1-4):263–280, 2007.
16. Andrzej Ehrenfeucht and Grzegorz Rozenberg. Introducing time in reaction systems. *Theoretical Computer Science*, 410(4-5):310–322, 2009.
17. Daniela Genova, Hendrik Jan Hoogeboom, and Nataša Jonoska. A graph isomorphism condition and equivalence of reaction systems. *Theoretical Computer Science*, 701:109–119, 2017.
18. Lila Kari and Grzegorz Rozenberg. The many facets of natural computing. *Commun. ACM*, 51(10):72–83, 2008.
19. Jetty Kleijn and Maciej Koutny. Membrane systems with qualitative evolution rules. *Fundam. Inf.*, 110(1-4):217–230, January 2011.
20. Jetty Kleijn, Maciej Koutny, Lukasz Mikulski, and Grzegorz Rozenberg. Reaction systems, transition systems, and equivalences. In Hans-Joachim Böckenhauer, Dennis Komm, and Walter Unger, editors, *Adventures Between Lower Bounds and Higher Altitudes - Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*, volume 11011 of *Lecture Notes in Computer Science*, pages 63–84. Springer, 2018.
21. Jetty Kleijn, Maciej Koutny, and Grzegorz Rozenberg. *Modelling reaction systems with Petri nets*. Computing Science, Newcastle University, 2011.
22. Albert L. Lehninger. *Bioenergetics: the molecular basis of biological energy transformations*. Biology teaching monograph series. W. A. Benjamin, 1965.
23. Artur Męski, Maciej Koutny, and Wojciech Penczek. Towards quantitative verification of reaction systems. In *Unconventional Computation and Natural Computation: 15th International Conference, UCNC 2016, Manchester, UK, July 11-15, 2016, Proceedings*, pages 142–154, 2016.

24. Artur Męski, Maciej Koutny, and Wojciech Penczek. Verification of linear-time temporal properties for reaction systems with discrete concentrations. *Fundam. Inform.*, 154(1-4):289–306, 2017.
25. Artur Męski, Wojciech Penczek, and Grzegorz Rozenberg. Model checking temporal properties of reaction systems. *Information Sciences*, 313:22–42, 2015.
26. Artur Męski, Wojciech Penczek, Maciej Szreter, Bożena Woźna-Szcześniak, and Andrzej Zbrzezny. BDD- versus SAT-based bounded model checking for the existential fragment of linear temporal logic with knowledge: algorithms and their performance. *Autonomous Agents and Multi-Agent Systems*, 28(4):558–604, 2014.
27. Artur Męski, Maciej Koutny, and Wojciech Penczek. Reaction mining for reaction systems. In *Unconventional Computation and Natural Computation - 17th International Conference, UCNC 2018, Fontainebleau, France, June 25-29, 2018, Proceedings*, pages 131–144, 2018.
28. Shipra Panda and Fabio Somenzi. Who are the variables in your neighborhood. In Richard L. Rudell, editor, *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1995, San Jose, California, USA, November 5-9, 1995*, pages 74–77. IEEE Computer Society / ACM, 1995.
29. Gheorghe Paun. *Membrane computing: an introduction*. Springer-Verlag, Berlin Heidelberg New York, 2002.
30. Gheorghe Paun, Grzegorz Rozenberg, and Arto Salomaa. *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., New York, NY, USA, 2010.
31. W. Penczek and A. Lomuscio. Verifying epistemic properties of multi-agent systems via bounded model checking. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '03, pages 209–216, New York, NY, USA, 2003. ACM.
32. Gheorghe Păun and Grzegorz Rozenberg. A guide to membrane computing. *Theor. Comput. Sci.*, 287(1):73–100, 2002.
33. Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok, editors. *Handbook of Natural Computing*. Springer, 2012.
34. Arto Salomaa. Functional constructions between reaction systems and propositional logic. *International Journal of Foundations of Computer Science*, 24(01):147–159, 2013.
35. F. Somenzi. CUDD: CU decision diagram package - release 2.5.0. http://vlsi.colorado.edu/~fabio/CUDD.