Information Sciences 313 (2015) 22–42

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Model checking temporal properties of reaction systems

Artur Męski [a,b,*], Wojciech Penczek [a,e], Grzegorz Rozenberg [c,d]

[a] Institute of Computer Science, PAS, Jana Kazimierza 5, 01-248 Warsaw, Poland
[b] University of Łódź, FMCS, Banacha 22, 90-238 Łódź, Poland
[c] LIACS, Leiden University, P.O. Box 9512, 2300 RA, The Netherlands
[d] Department of Computer Science, University of Colorado at Boulder, 430 UCB, Boulder, CO 80309-0430, USA
[e] University of Natural Sciences and Humanities, ICS, Siedlce, Poland

ARTICLE INFO

Article history:
Received 13 November 2014
Received in revised form 12 February 2015
Accepted 22 March 2015
Available online 28 March 2015

Keywords:
Reaction system
Model checking
Temporal logic

ABSTRACT

This paper defines a temporal logic for reaction systems (rsCTL). The logic is interpreted over the models for the context restricted reaction systems that generalise standard reaction systems by controlling context sequences. Moreover, a translation from the context restricted reaction systems into boolean functions is defined in order to be used for a symbolic model checking for rsCTL over these systems. The model checking for rsCTL is proved to be PSPACE-complete. The proposed approach to model checking was implemented and experimentally evaluated using four benchmarks.

1. Introduction

Model checking [9] is a method that allows for checking whether or not the system in question satisfies a given formula specifying either a desired or an undesired property of that system. Typically, the verified properties are expressed using some modal logic formalism. This method is fully automatic and to be used in practice it does not require expert knowledge of verification techniques.

The basic idea behind the original motivation of reaction systems as models of processes inspired by the functioning of the living cell (see, e.g., [5,12,13]) is that this functioning is based on the interactions of individual biochemical reactions. Moreover, these interactions are driven by two mechanisms: facilitation/acceleration and inhibition/retardation.

This paper introduces Computation Tree Logic for reaction systems (rsCTL), which is a logic for specifying properties of reaction systems, together with a method for verifying these properties. This is the first paper providing a verification method for reaction systems.

Because the processes of reaction systems are guided by the context sequences (which model an interaction with the environment), to enable the verification we introduce a generalisation of reaction systems which allows to specify context entities generating all the context sequences for the processes of the given reaction system. Moreover, we describe an encoding of the model for reaction systems into boolean formulae that can be used for the symbolic model checking approach. We also provide some complexity results for the problem of model checking reaction systems.

In this paper we extend the results of our technical report [21] with a theoretical and experimental analysis of the proposed approach to verification of reaction systems. Following the introduction of reaction systems [13], their extensions were also studied: [14] introduced time in reaction systems, [18] dealt with quantum and probabilistic reaction systems,

* Corresponding author at: Institute of Computer Science, PAS, Jana Kazimierza 5, 01-248 Warsaw, Poland. Fax: +48 22 380 05 10.
E-mail address: meski@ipipan.waw.pl (A. Męski).

http://dx.doi.org/10.1016/j.ins.2015.03.048
0020-0255/

and [22] approached reaction systems with an automata semantics. However, none of the proposed extensions dealt with restrictions on the context sequences of reaction systems. A significant part of the research on reaction systems is focused on their mathematical properties. Fixed points, attractors, and cycles for reaction systems were investigated in [15,16]. Occurrence and convergence problems were tackled in [17,28,29]. Different classes and properties of reaction systems related to their state sequences were studied in [26,27]. Another strand of research focuses on applying reaction systems to modelling of systems. In [11] the authors modelled the gene regulation mechanism for lactose operon of Escherichia coli and explored modelling possibilities for several well-known computer science problems. A reaction systems model of the eukaryotic heat shock response, which we use in this paper as a benchmark, was described in [4] together with properties of its expected behaviour. There is also an increasing interest in verification of reaction systems. In [2] the authors define several biologically inspired properties, including a mass conservation property, together with the corresponding verification problems. These properties can be expressed in the logic considered in the technical report [21], in which the logic was called RSTL, and this paper (rsCTL). Some of the problems defined in the paper are of lower complexity than for the model checking of rsCTL. The paper focuses mostly on complexity considerations for decision problems related to verification and does not develop any language for specifying properties of reaction systems. In [3] the authors investigate the property of mass conservation using dependency graphs, and introduce a simulator for reaction systems.

The paper is organised as follows. In Section 2 we recall the basic notions of reaction systems, while in Section 3 we define two generalisations of reaction systems. Two basic illustrative examples are given in Section 4. In Section 5 we define the syntax of the logic for reaction systems together with a model used to define the semantics of the logic, based on which, in Section 6 we define a model checking method for reaction systems and prove its complexity. In Section 7 we present the encoding for the model defined in Section 5 – this encoding can be used for symbolic model checking of reaction systems. In Section 8 we introduce a tool which is an experimental implementation of the model checking method for reaction systems. We use the tool to evaluate our approach on four reaction systems revealing various practical aspects of the tackled verification problems. The last section of the paper provides concluding remarks.

2. Reaction systems

Reaction systems are a formal model for processes instigated by the functioning of living cell. Research topics in this research area are motivated either by biological issues or by a need to understand computations/processes underlying the dynamic behaviour of reaction systems. By now reaction systems became an interesting and novel model of computation.

In this section we recall some basic notions of reaction systems that are used in this paper. First of all, we recall the notion of a reaction.

Definition 2.1. A *reaction* is a triplet $b = (R, I, P)$ such that R, I, P are finite nonempty sets with $R \cap I = \emptyset$.

The sets R, I, P are called the *reactant set of b*, the *inhibitor set of b*, and *the product set of b*, respectively – they are also denoted by P_b, I_b, and P_b, respectively. The requirement that all three sets R, I, and P are nonempty is motivated by biological considerations: there is no creation from nothing ($R \neq \emptyset$), each reaction may be inhibited ($I \neq \emptyset$), and if a reaction takes place then this creates a "material" effect – something is produced ($P \neq \emptyset$).

If $R, I, P \subseteq Z$ for a finite set Z, then we say that b is a *reaction in Z*. We use $rac(Z)$ to denote the set of all reactions in Z.

The above formal notion of a reaction corresponds closely to the basic intuition behind a biochemical reaction. Such a reaction will take place if all of its reactants are present and none of its inhibitors is present, and if it takes place it produces its set of products.

Definition 2.2. Let Z be a finite set, and let $T \subseteq Z$.

1. We say that $b \in rac(Z)$ is *enabled* by T, denoted $en_b(T)$, if $R_b \subseteq T$, and $I_b \cap T = \emptyset$. The *result* of b on T, denoted by $res_b(T)$, is defined by: $res_b(T) = P_b$ if $en_b(T)$, and $res_b(T) = \emptyset$ otherwise.
2. For $B \subseteq rac(Z)$, the result of B on T, denoted by $res_B(T)$, is defined by $res_B(T) = \bigcup\{res_b(T) | b \in B\}$.

The intuition underlying the above definition is that T formalises a state of a biochemical system under consideration: it is simply the set of all biochemical entities present in the given state. A reaction b is enabled by T (can take place at T) if all the reactants of b are present in T and none of the inhibitors of b is present in T. Therefore we require that $R_b \cap I_b = \emptyset$ – in this way we do not consider "trivial reactions", i.e., reaction that are never enabled.

Then the result of a set of reactions B is *cumulative*, i.e., it is the union of results of the individual reactions from B – clearly, $res_B(T) = \bigcup\{res_b(T) | b \in B \text{and} en_b(T)\}$.

We are ready now to define the notion of *reaction system*.

Definition 2.3. A *reaction system*, *rs* for short, is an ordered pair $\mathcal{R} = (S, A)$, where S is a finite set and $A \subseteq rac(S)$.

The set S is the *background set* of $\mathcal{R}$. The elements of S are called entities, each subset of S is called a *state* of $\mathcal{R}$, and A is the *set of reactions from* $\mathcal{R}$.

Example 2.4. Consider the set $S = \{1,2,3,4\}$ and the following set A of reactions from $rac(S)$:

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

Then, $\mathcal{R}_1 = (S,\{a_1,a_2,a_3,a_4\})$ is a rs.

Consider the state $T = \{1,3,4\}$. Then reactions a_1, a_3, a_4 are enabled by T, while a_2 is not enabled by T. Consequently $res_A(T) = res_{a_1}(T) \cup res_{a_3}(T) \cup res_{a_4}(T) = \{1,2\} \cup \{1,2\} \cup \{1\} = \{1,2\}$. □

Since the successor state T' of a current state T (thus $T' = res_A(T)$) is the union of the products of all reactions from A which are enabled by T, an entity x from T will vanish (i.e., will not be present in T') unless it is produced (sustained) by a reaction enabled by T. This *non-permanency* of entities in reaction systems is motivated by basic bioenergetics of the living cell: without supply of energy the living cell with all its molecules disintegrates. But absorbing energy is a chemical process achieved through biochemical reactions – thus a molecule (an entity) is sustained if it is sustained by a biochemical reaction. The vanish of a non-sustained entity in the basic model of a reaction system happens within a single transition step (from T to T'). However in other more elaborated models in the broad framework of reaction systems the basic biochemical effect of decay is taken into account and so the vanishing takes place within a number of transition steps (see, e.g., [12]).

Also, since the successor state of a current state T is the union of the products of reactions enabled by T, there are no conflicts between reactions enabled by T. Therefore there is no counting in reaction systems, and so it is a qualitative model. This follows from the level of abstraction adopted for the basic model. However, in the broad framework of reaction systems (see, e.g., [12]) one considers models which include counting.

Hence a reaction system is basically a set of reactions over a finite background set. There is no structure involved (such as tapes, counters, pushdowns) – reactions are primary here. This reflects our point of view that the living cell is basically a reactor in which reactions (from a finite set of reactions) interact. The processes resulting from these interactions underlie the functioning of the living cell. These dynamic processes are formalised as follows.

Definition 2.5. Let $\mathcal{R} = (S,A)$ be a rs and let $n \geqslant 1$.

1. An (n-step) *interactive process* of $\mathcal{R}$ is a pair $\pi = (\gamma,\delta)$ of finite sequences of finite sets such that:
   (a) $\gamma = (C_0, C_1, \ldots, C_n)$ and $\delta = (D_0, D_1, \ldots, D_n)$,
   (b) $C_0, C_1, \ldots, C_n \subseteq S$,
   (c) $D_0, D_1, \ldots, D_n \subseteq S$, with $D_0 = \emptyset$, and
   (d) $D_i = res_A(D_{i-1} \cup C_{i-1})$ for all $i \in \{1,\ldots,n\}$.
2. The *state sequence* of π is the sequence $\tau = (W_0, W_1, \ldots, W_n)$ such that $W_i = C_i \cup D_i$ for all $i \in \{0,\ldots,n\}$.

The sequence γ is the *context sequence of* π and the sequence δ is the *result sequence of* π. The sequence $(C_1, C_2, \ldots, C_n)$ is the *proper context sequence of* π, and the set C_0 is the *initial state of* π. If $C_i \subseteq D_i$ for all $i \in \{1,\ldots,n\}$, then we say that π (and τ) are *context-independent*. Clearly, the context sequence γ of an interactive process $\pi = (\gamma,\delta)$ determines π because the result sequence δ is obtained from γ by reactions of $\mathcal{R}$ (through res_A). The proper context sequence of an interactive process reflects the fact that the living cell is an open system, i.e., its behaviour is influenced by its context/environment.

Note that:

1. The non-permanency of entities carries over to processes in the obvious way: an entity x from a current state W_i is not sustained in the successor state W_{i+1} unless x is produced by a reaction enabled by W_i ($x \in D_{i+1}$) or x is thrown in by the context ($x \in C_{i+1}$).
2. There is no restriction on the context sequence: any sequence of subsets of the background set S can be a context sequence of an interactive process.

It is important to observe that reaction system is a strictly finite system in the sense that the size of each state is a priori limited (by the size of the background set, which is finite).

3. Controlling context sequences

We will consider now a basic method to control/restrict proper context sequences, just by restricting the set of entities that can occur in them. This leads to the following definition.

Definition 3.1. A *context restricted reaction system*, *crrs* for short, is a triple $\mathcal{U} = (S,A,\mathcal{E})$ where:

1. S is the (finite) background set,
2. $A \subseteq rac(S)$ is the set of reactions,
3. $\mathcal{E} \subseteq S$ is the set of *context entities*.

Our next step to control/restrict interactive processes of reaction systems is to allow only some states to be initial states. This yields the following definition.

Definition 3.2. Let $\mathcal{U} = (S, A, \mathcal{E})$ be a crrs. An *initialised context restricted reaction system*, *icrrs* for short, is a pair $\mathcal{I} = (\mathcal{U}, S_0)$, where $S_0 \subseteq 2^S$ is the set of *initial states* such that $S_0 \neq \emptyset$.

We need to modify now the notion of an interactive process for crrs and icrrs so that it reflects the role of the corresponding restrictions on proper context sets and initial states.

Definition 3.3. Let $\mathcal{U} = (S, A, \mathcal{E})$ be a crrs and let $n \geqslant 0$ be an integer. An (n-step) *interactive process* in $\mathcal{U}$ is a pair $\pi = (\gamma, \delta)$ of finite sequences of finite sets such that:

1. $\gamma = (C_0, C_1, \ldots, C_n)$ and $\delta = (D_0, D_1, \ldots, D_n)$,
2. $C_0 \subseteq S, C_1, \ldots, C_n \subseteq \mathcal{E}$,
3. $D_0, D_1, \ldots, D_n \subseteq S, D_0 = \emptyset$, and
4. $D_i = res_A(D_{i-1} \cup C_{i-1})$ for all $i \in \{1, \ldots, n\}$.

For an icrrs $\mathcal{I} = ((S, A, \mathcal{E}), S_0)$ the above definition of an interactive process is augmented by adding the condition $C_0 \in S_0$. Here is a simple example of an icrrs.

Example 3.4. Consider the icrrs $\mathcal{I}_1 = ((S, A, \{4\}), \{\{1, 4\}\})$, where (S, A) is the rs $\mathcal{R}_1$ from Example 2.4. Let $\gamma = (\{1, 4\}, \emptyset, \{4\}, \{4\})$ and $\delta = (\emptyset, \{1, 2\}, \{1, 3, 4\}, \{1, 2\})$. Then, $\pi = (\gamma, \delta)$ is a 3-step interactive process of $\mathcal{R}_1$. It is also an interactive process of $\mathcal{I}_1$, because all proper context sets are subsets of $\{4\}$ and the initial context set equals $\{1, 4\}$. However π is not an interactive process of the crrs $(S, A, \{1, 3\})$, because the context set $\{4\}$ is not a subset of $\{1, 3\}$. □

4. Basic examples

Example 4.1. We consider here an implementation by a rs of a small generic regulatory system $\mathcal{G}$ discussed in [12] (Section 3).

The regulatory system contains three (abstract) genes x, y, z expressing proteins X, Y, Z, respectively, protein U, and protein complex Q formed by X and U. The expression of X by x is inhibited by Y and Z, the expression of Z by z is inhibited by X, and expression of Y by y is inhibited by the protein complex Q.

The background set $S = \{x, \widehat{x}, X, y, \widehat{y}, Y, z, \widehat{z}, Z, Q, U, h\}$, where $\widehat{x}, \widehat{y}$, and $\widehat{z}$ denote RNA polymerase sitting on the promoter of genes x, y, and z, respectively. Here h is a "dummy entity" to be used as an inhibitor whenever we do not specify other inhibitors for a reaction (this is a simplification of a specification of a desired rs which is made for didactic reasons).

Finally, the set of reactions consists of four subsets (A_x, A_y, A_z, and A_Q) defined as follows:

- $A_x = \{(\{x\}, \{h\}, \{x\}), (\{x\}, \{Y, Z\}, \{\widehat{x}\}), (\{x, \widehat{x}\}, \{h\}, \{X\})\}$,
- $A_y = \{(\{y\}, \{h\}, \{y\}), (\{y\}, \{Q\}, \{\widehat{y}\}), (\{y, \widehat{y}\}, \{h\}, \{Y\})\}$,
- $A_z = \{(\{z\}, \{h\}, \{z\}), (\{z\}, \{X\}, \{\widehat{z}\}), (\{z, \widehat{z}\}, \{h\}, \{Z\})\}$, and
- $A_Q = \{(\{U, X\}, \{h\}, \{Q\})\}$.

The intuition behind the reactions above corresponds closely to the biological actions taking place in a genetic regulatory system. For example:

- $(\{x\}, \{h\}, \{x\})$ says that if gene x is present and functional in a current state, then x will be present and functional in the successor state, unless something "bad" (inhibition) happens – since $\mathcal{G}$ does not specify such inhibitions, this inhibition is expressed by the dummy inhibitor h (e.g., h can represent a very high level of radiation).
- $(\{x\}, \{Y, Z\}, \{\widehat{x}\})$ says that if gene x is present and functional in current state, then RNA polymerase will sit on its promoter field meaning that $\widehat{x}$ will be present in the successor state (thus $\widehat{x}$ represents RNA polymerase sitting on the promoter field of gene x). This will happen providing that neither protein Y nor protein Z is present in the current state.
- $(\{x, \widehat{x}\}, \{h\}, \{X\})$ says that if gene x is present and functional in the current state and RNA polymerase sits on its promoter field, then eventually protein X will be expressed. This can be inhibited by a whole set of reasons which are not relevant for our story here, and so (again) we set here the dummy inhibitor h.
- $(\{U, X\}, \{h\}, \{Q\})$ says that if both proteins X and U are present in the current state, then protein complex Q will be present in the successor state, unless inhibited by something (not specified in $\mathcal{G}$) formalised by the dummy h.

Now, let us assume that we want to look into the processes starting from the states that already contain x and y. Then, the icrrs for this model could be defined as

$$\mathcal{I}_{ge} = ((S, A, \{U\}), \{\{x, y\}\}),$$

where: $A = A_x \cup A_y \cup A_z \cup A_Q$ and U is the unique context entity. □

Example 4.2. Now we use an implementation by a rs of an n-bit cyclic binary counter given in [5] (Section 4.1) to provide an icrrs implementing such a counter. A current value of the counter can be increased by one or decreased by one depending on the controller's request. If no such request is present in a current state, then the value of the counter remains the same.

The background set of the reaction system implementing this counter is $S_n = \{p_0, \ldots, p_{n-1}, inc, dec\}$.

The entities $\{p_0, \ldots, p_{n-1}\}$ allow to provide a set representation of numbers from the range $\{0, \ldots, 2^{n-1}\}$ represented in the positional binary notation: entity p_i represents the enabled bit corresponding to the value of 2^i, for $i \in \{0, \ldots, n-1\}$. For example, with $n = 5$, the value 01011 (i.e., 11 in base 10) is represented by the set $\{p_0, p_1, p_3\}$. Thus, for each $W \subseteq S_n$, the value represented by W equals $\sum_{p_i \in W'} 2^i$, where $W' = W \setminus \{inc, dec\}$.

The entities *inc* and *dec* are used to represent the instructions to increase by one (successor), and to decrease by one (predecessor) the value of the current state of the counter. If one attempts to increase and decrease the counter value at the same time, then the value of the counter is reset to zero.

Then we need the following reactions:

- Retention:
  - For all $j \in \{0, \ldots, n-1\}$: $a_j = (\{p_j\}, \{dec, inc\}, \{p_j\})$.
- Increment:
  - $b_0 = (\{inc\}, \{dec, p_0\}, \{p_0\})$,
  - For all $j \in \{1, \ldots, n-1\}$:

    $b_j = (\{inc, p_0, \ldots, p_{j-1}\}, \{dec, p_j\}, \{p_j\})$,
  - For all $j, k \in \{0, \ldots, n-1\}, j < k$:

    $c_{j,k} = (\{inc, p_k\}, \{dec, p_j\}, \{p_k\})$.
- Decrement:
  - For all $j \in \{0, \ldots, n-1\}$:

    $d_j = (\{dec\}, \{inc, p_0, \ldots, p_j\}, \{p_j\})$,
  - For all $j, k \in \{0, \ldots, n-1\}, j < k$:

    $e_{j,k} = (\{dec, p_j, p_k\}, \{inc\}, \{p_k\})$.

Let then:

$$B_n = \{a_j : 0 \leqslant j \leqslant n\} \cup \{b_j : 0 \leqslant j \leqslant n\} \cup \{d_j : 0 \leqslant j \leqslant n\} \cup \{c_{j,k} : 0 \leqslant j < k < n\} \cup \{e_{j,k} : 0 \leqslant j < k < n\}.$$

Finally the desired icrrs is defined as $\mathcal{I}^n_{bc} = ((S_n, B_n, \mathcal{E}), \{\emptyset\})$, where $\mathcal{E} = \{inc, dec\}$. Thus, with this implementation of an n-bit cyclic binary counter by an icrrs, the only allowed initial state (for interactive processes in $\mathcal{I}^n_{bc}$) is the empty set – it represents the state of the corresponding counter where all bits are set to 0 and no controller request (*inc* or *dec*) is present. □

5. Logic for reaction systems

Our aim is to describe properties of reaction systems by using a branching time logic and, later on, to verify these properties by means of a model checking technique. In this section we introduce the syntax and semantics of a logic for reaction systems.

5.1. Syntax and semantics

Let $\mathcal{I} = ((S, A, \mathcal{E}), S_0)$ be an icrrs, and $\mathcal{PV}$ be a nonempty set of propositional variables. Here, we assume that $\mathcal{PV} = S$. The language of *Computation Tree Logic for reaction systems*, rsCTL for short, is defined by the following grammar:

$$\phi := \wp | \neg\phi | \phi \vee \phi | \mathbf{E}_\Psi \mathbf{X}\phi | \mathbf{E}_\Psi \mathbf{G}\phi | \mathbf{E}_\Psi [\phi \mathbf{U} \phi],$$

where $\wp \in \mathcal{PV}$ and $\emptyset \neq \Psi \subseteq 2^{\mathcal{E}}$.

The operators of rsCTL are composed of the path quantifier $\mathbf{E}_\Psi$ and temporal operators ($\mathbf{X}, \mathbf{G}, \mathbf{U}$). The path quantifier $\mathbf{E}_\Psi$ means 'there exists a path over Ψ': the argument Ψ restricts the set of the considered paths by describing the set of actions allowed along the paths. The temporal operators are used to express requirements imposed on the paths selected by the path quantifiers. The $\mathbf{X}\phi$ operator means 'in the next state ϕ holds', $\mathbf{G}\phi$ means 'in each state of the path (globally) ϕ holds'. The $\phi\mathbf{U}\psi$ operator uses two properties and means 'ψ holds eventually, and ϕ must hold at every preceding state'.

With every rsCTL formula we associate its maximal nesting depth, corresponding to the number of levels at which rsCTL subformulae appear. The following notion is used in complexity considerations of model checking algorithms in Section 6.

Definition 5.1. Let ϕ be an rsCTL formula. Then, $\mathsf{d}(\phi)$ is the *depth of* ϕ and is defined recursively as follows:

- if $\phi = \wp$, where $\wp \in \mathcal{PV}$, then $\mathsf{d}(\phi) = 1$,
- if $\phi \in \{\neg\phi', \mathbf{E}_\Psi\mathbf{X}\phi', \mathbf{E}_\Psi\mathbf{G}\phi'\}$, then $\mathsf{d}(\phi) = \mathsf{d}(\phi') + 1$,
- if $\phi \in \{\phi' \vee \phi'', \mathbf{E}_\Psi[\phi'\mathbf{U}\phi'']\}$, then $\mathsf{d}(\phi) = max(\{\mathsf{d}(\phi'), \mathsf{d}(\phi'')\}) + 1$.

For $\Psi \subseteq 2^{\mathcal{E}}$, the number of the context sets in Ψ is denoted by $|\Psi|$.

Definition 5.2. Let ϕ be an rsCTL formula. By $\mathsf{c}(\phi)$ we mean the size of the largest set Ψ of the subformulae of ϕ defined recursively as follows:

- if $\phi = \wp$, where $\wp \in \mathcal{PV}$, then $\mathsf{c}(\phi) = 0$,
- if $\phi = \neg\phi'$, then $\mathsf{c}(\phi) = \mathsf{c}(\phi')$,
- if $\phi \in \{\neg\phi', \mathbf{E}_\Psi\mathbf{X}\phi', \mathbf{E}_\Psi\mathbf{G}\phi'\}$, then $\mathsf{c}(\phi) = max(\{|\Psi|, \mathsf{c}(\phi')\})$,
- if $\phi \in \{\phi' \vee \phi'', \mathbf{E}_\Psi[\phi'\mathbf{U}\phi'']\}$, then $\mathsf{c}(\phi) = max(\{|\Psi|, \mathsf{c}(\phi'), \mathsf{c}(\phi'')\})$.

Next, we define the models for rsCTL that are used for interpreting the rsCTL formulae.

Definition 5.3. Let $\mathcal{I} = ((S, A, \mathcal{E}), S_0)$ be an icrrs. Then, the model for $\mathcal{I}$ is defined as $\mathcal{M}(\mathcal{I}) = (\mathbb{W}, \mathsf{W}_0, \rightarrow, L)$ where:

1. $\mathbb{W} = 2^S$ is the set of the *states*,
2. $\mathsf{W}_0 = \{res_A(\alpha) | \alpha \in S_0\} \subseteq \mathbb{W}$ is the set of the *initial states*,
3. $\rightarrow \subseteq \mathbb{W} \times 2^{\mathcal{E}} \times \mathbb{W}$ is the *transition relation* such that for all $w, w' \in \mathbb{W}, \alpha \in 2^{\mathcal{E}}$: $(w, \alpha, w') \in \rightarrow$ iff $w' = res_A(w \cup \alpha)$,
4. $L : \mathbb{W} \rightarrow 2^{\mathcal{PV}}$ is a *valuation function* such that $L(w) = w$ for all $w \in \mathbb{W}$.

To simplify the notation, for the sequel of this paper we fix $\mathcal{I} = ((S, A, \mathcal{E}), S_0)$ and its model $\mathcal{M}_\mathcal{I} = (\mathbb{W}, \mathsf{W}_0, \rightarrow, L)$. Each element $(w, \alpha, w') \in \rightarrow$ is denoted by $w \xrightarrow{\alpha} w'$.

The following lemma follows immediately from Definition 5.3. It states that the transition relation is serial, i.e., every state of the model has a successor.

Lemma 5.4. *For each $w \in \mathbb{W}$ there exists $\alpha \in 2^{\mathcal{E}}$ and $w' \in \mathbb{W}$ such that $w \xrightarrow{\alpha} w'$.*

The formulae of rsCTL are interpreted in each state, but they express properties of the *paths* initialised in a given state. In CTL [10], the paths are defined as sequences of states. However, the paths in rsCTL contain additional elements which represent context sets. Thus, the paths for rsCTL are defined as sequences of states, interleaved with *actions*, that is, subsets of the set $\mathcal{E}$.

Definition 5.5. A *path* over $\Psi \subseteq 2^{\mathcal{E}}$ is an infinite sequence $\sigma = (w_0, \alpha_0, w_1, \alpha_1, \ldots)$ of states and actions such that: $w_i \xrightarrow{\alpha_i} w_{i+1}$ and $\alpha_i \in \Psi$ for $i \geqslant 0$.

The set of all the paths over Ψ is denoted by Π_Ψ. For each $i \geqslant 0$, 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 $\Pi_\Psi(w)$ we denote the set of all the paths over Ψ that start in $w \in \mathbb{W}$, that is, $\Pi_\Psi(w) = \{\sigma \in \Pi_\Psi | \sigma_s(0) = w\}$.

Let $w, w' \in \mathbb{W}$ and $\Psi \subseteq 2^{\mathcal{E}}$. We say that w' is a Ψ-successor of w (denoted by $w \rightarrow_\Psi w'$) iff there exists $\alpha \in \Psi$ such that $w \xrightarrow{\alpha} w'$.

Next, we introduce the notion of a *reachable state* and, later on, we present an example of the reachable part of a model.

Definition 5.6. Let $\mathcal{I} = ((S, A, \mathcal{E}), S_0)$ be an icrrs and let $\mathcal{M}_\mathcal{I} = (\mathbb{W}, \mathsf{W}_0, \rightarrow, L)$ be the model for $\mathcal{I}$. We say that a state $w \in \mathbb{W}$ is *reachable* over $\Psi \subseteq 2^{\mathcal{E}}$ in $\mathcal{M}_\mathcal{I}$ if there exists $w' \in \mathsf{W}_0$ and a path $\sigma \in \Pi_\Psi(w')$ such that $\sigma_s(i) = w$ for some $i \geqslant 0$.

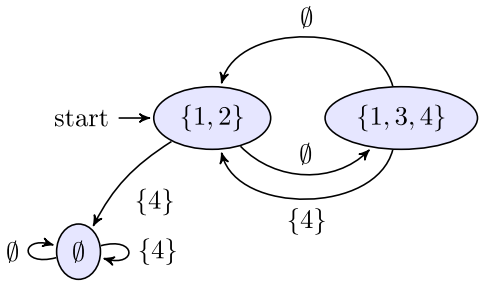

Fig. 1. The reachable part of the model for the icrrs $\mathcal{I}_1$ from Example 3.4.

Example 5.7. Consider the icrrs $\mathcal{I}_1$ from Example 3.4. Then, the reachable part of the model $\mathcal{M}_{\mathcal{I}_1}$ is shown in Fig. 1. The states of the model are depicted as the ellipsis.

Now we are ready to define the semantics of rsCTL.

Definition 5.8. Let $\mathcal{M}_{\mathcal{I}} = (\mathbb{W}, \mathsf{W}_0, \rightarrow, L)$ be a model and $w \in \mathbb{W}$ be a state of $\mathcal{M}_{\mathcal{I}}$. The fact that ϕ holds in the state w of the model $\mathcal{M}_{\mathcal{I}}$ is denoted by $\mathcal{M}_{\mathcal{I}}, w \models \phi$, where the relation $\models$ is defined recursively as follows:

$$
\begin{array}{lll}
\mathcal{M}_{\mathcal{I}}, w \models \wp & \text{iff} & \wp \in L(w) \text{ for } \wp \in \mathcal{PV}, \\
\mathcal{M}_{\mathcal{I}}, w \models \neg\phi & \text{iff} & \mathcal{M}_{\mathcal{I}}, w \not\models \phi, \\
\mathcal{M}_{\mathcal{I}}, w \models \phi \vee \psi & \text{iff} & \mathcal{M}_{\mathcal{I}}, w \models \phi \text{ or } \mathcal{M}_{\mathcal{I}}, w \models \psi, \\
\mathcal{M}_{\mathcal{I}}, w \models \mathbf{E}_{\Psi}\mathbf{X}\phi & \text{iff} & (\exists \sigma \in \Pi_{\Psi}(w))\mathcal{M}_{\mathcal{I}}, \sigma_s(1) \models \phi, \\
\mathcal{M}_{\mathcal{I}}, w \models \mathbf{E}_{\Psi}\mathbf{G}\phi & \text{iff} & (\exists \sigma \in \Pi_{\Psi}(w))(\forall i \geqslant 0)(\mathcal{M}_{\mathcal{I}}, \sigma_s(i) \models \phi), \\
\mathcal{M}_{\mathcal{I}}, w \models \mathbf{E}_{\Psi}[\phi \mathbf{U} \psi] & \text{iff} & (\exists \sigma \in \Pi_{\Psi}(w))(\exists i \geqslant 0)(\mathcal{M}_{\mathcal{I}}, \sigma_s(i) \models \psi \\
 & & \text{and } (\forall 0 \leqslant j < i)\mathcal{M}_{\mathcal{I}}, \sigma_s(j) \models \phi).
\end{array}
$$

We define now derived operators which also introduce the universal path quantifier $\mathbf{A}_{\Psi}$ meaning 'for all the paths over Ψ':

- $true \stackrel{\text{def}}{=} \wp \vee \neg\wp$ for any $\wp \in S$,
- $\phi \wedge \psi \stackrel{\text{def}}{=} \neg(\neg\phi \vee \neg\psi)$,
- $\phi \Rightarrow \psi \stackrel{\text{def}}{=} \neg\phi \vee \psi$,
- $\phi \oplus \psi \stackrel{\text{def}}{=} (\phi \wedge \neg\psi) \vee (\neg\phi \wedge \psi)$,
- $\mathbf{E}_{\Psi}\mathbf{F}\phi \stackrel{\text{def}}{=} \mathbf{E}_{\Psi}[true\mathbf{U}\phi]$,
- $\mathbf{A}_{\Psi}\mathbf{F}\phi \stackrel{\text{def}}{=} \neg\mathbf{E}_{\Psi}\mathbf{G}\neg\phi$,
- $\mathbf{A}_{\Psi}\mathbf{X}\phi \stackrel{\text{def}}{=} \neg\mathbf{E}_{\Psi}\mathbf{X}\neg\phi$,
- $\mathbf{A}_{\Psi}\mathbf{G}\phi \stackrel{\text{def}}{=} \neg\mathbf{E}_{\Psi}[true\mathbf{U}\neg\phi]$.

Moreover, we assume $\Psi = 2^{\mathcal{E}}$ when the set Ψ is unspecified for any of the rsCTL operators, e.g., $\mathbf{EF}\phi \stackrel{\text{def}}{=} \mathbf{E}_{2^{\mathcal{E}}}\mathbf{F}\phi$.

We assume that a formula ϕ *holds in the model* $\mathcal{M}_{\mathcal{I}}$ iff $\mathcal{M}_{\mathcal{I}}, w \models \phi$ for all $w \in \mathsf{W}_0$, that is, the formula ϕ holds in all the initial states. This fact is denoted by $\mathcal{M}_{\mathcal{I}} \models \phi$.

The language of our logic resembles the language of action-restricted CTL (ARCTL) of [23]. However, rsCTL is specialised for reaction systems and it facilitates a direct and intuitive specification of the context sets. This approach also allows, as explained in the following section, for easy specification of context independent sequences by using path quantifiers with $\Psi = \{\emptyset\}$. To the best of our knowledge, none of the existing logics could be directly used for our purpose. This follows from the fact that the classes of models for existing logics, ARCTL in particular, are different than the models for reaction systems. For example, there are models of ARCTL that do not correspond to any reaction system. To restrict the path quantifier we use families of sets of entities instead of propositional formulae. We could have exploited the language of ARCTL by reinterpreting it over the models for reaction systems. Then, the propositional formulae built over the entities could be used in place of families of sets of entities. For each formula of this kind there is a family of sets of entities which defines all the valuations that satisfy that formula. However, given that our tool (presented in Section 8) is intended for non-logicians we decided to use a more intuitive language where we use families of sets.

This approach is also in line with the definition of a reaction in reaction systems, which is also set-based. On the other hand, our tool also accepts propositional formulae over the entities in place of families of sets.

5.2. Examples of properties expressible in rsCTL

The following lemma states that there exists a context-independent sequence, if and only if, there exists a proper context sequence with all the context sets being empty.

Lemma 5.9. *Let $\mathcal{I} = ((S, A, \mathcal{E}), S_0)$ be an icrrs and $\mathcal{M}_\mathcal{I}$ be the model for $\mathcal{I}$. Then, there exists a path σ such that $\sigma_a(i) \subseteq \sigma_s(i)$ for each $i \geqslant 0$, if and only if, there exists a path σ' such that $\sigma'_s(i) = \sigma_s(i)$ and $\sigma'_a(i) = \emptyset$ for each $i \geqslant 0$.*

Proof. First, we assume that there exists a path σ such that $\sigma_a(i) \subseteq \sigma_s(i)$ for each $i \geqslant 0$. We assume $\sigma'_s(i) = \sigma_s(i)$ for each $i \geqslant 0$. We need to show that $\sigma'_s(i) \xrightarrow{\emptyset} \sigma'_s(i+1)$ for each $i \geqslant 0$. From the definition of the transition relation and the fact that $\sigma_a(i) \subseteq \sigma_s(i)$, it follows that $res_A(\sigma_s(i) \cup \sigma_a(i)) = res_A(\sigma_s(i) \cup \emptyset) = res_A(\sigma_s(i))$. From this and the definition of the transition relation, the states of the path σ' may be defined as: $\sigma'_s(i+1) = res_A(\sigma'(i))$, therefore $\sigma'_s(i) \xrightarrow{\emptyset} \sigma'_s(i+1)$, i.e., $\sigma'_a(i) = \emptyset$ for each $i \geqslant 0$. The converse follows immediately. □

This allows to choose only from the paths with all the context sets being empty in order to verify properties over context-independent sequences. This follows from the fact that in σ and σ' we preserve the order of the states, and the rsCTL formulae are interpreted in the states.

The following examples demonstrate how rsCTL can be used for expressing properties of icrrs.

Example 5.10. For the icrrs $\mathcal{I}_{ge}$ defined in Example 4.1 we can describe the following properties interpreted according to their validity in the model $\mathcal{M}_{\mathcal{I}_{ge}}$, i.e., they must hold in the initial states of the model:

1. It is possible that the protein Q will be finally produced:

   $\mathbf{EF}Q$.

2. If Q is present, then the polymerase will not land on the gene y (that is, $\widehat{y}$ will not be present in any of the immediate successors):

   $\mathbf{AG}(Q \Rightarrow (\mathbf{AX}\neg\widehat{y}))$.

3. Always, if the gene x is present, the polymerase lands on x (that is, $\widehat{x}$ is present), and the protein U is supplied in the context, then always when we supply U the protein Q is produced:

   $\mathbf{A}_{\{\{U\}\}}\mathbf{G}((x \wedge \widehat{x}) \Rightarrow \mathbf{A}_{\{\{U\}\}}\mathbf{F}Q)$.

4. It is possible that the protein Q will never be produced:

   $\mathbf{EG}(\neg Q)$.

5. If we do not supply U in the context, then Q will never be produced:

   $\mathbf{A}_\Psi\mathbf{G}(\neg Q)$, where $\Psi = \{\alpha \subseteq \mathcal{E} | U \notin \alpha\} = \{\emptyset\}$.

6. There exists a context-independent sequence (see Lemma 5.9) over which the state where X and Y are present is reachable:

   $\mathbf{E}_{\{\emptyset\}}\mathbf{F}(X \wedge Y)$.

Example 5.11. For the icrrs $\mathcal{I}_{bc}$ from Example 4.2 we can describe the following properties interpreted according to their validity in $\mathcal{M}_{\mathcal{I}_{ge}}$:

1. Always, if the counter is at its minimal value, then it is possible to reach the maximal value by supplying as context sets only *inc* or *dec* entities:

   $\mathbf{AG}((\neg b_0 \wedge \ldots \wedge \neg b_n) \Rightarrow \mathbf{E}_{\{\{inc\},\{dec\}\}}\mathbf{F}(b_0 \wedge \ldots \wedge b_n))$.

2. Always, if the counter reaches its maximal value, then always in the next step the counter will make a transition to the minimal value when we supply only *inc* entity:

   $\mathbf{AG}((b_0 \wedge \ldots \wedge b_n) \Rightarrow \mathbf{A}_{\{\{inc\}\}}\mathbf{X}(\neg b_0 \wedge \ldots \wedge \neg b_n))$.

6. Verification of rsCTL properties of icrrs

In this section we describe a model checking method for verification of the rsCTL properties. The method described here leads to a symbolic model checking problem which we define in Section 7.

To be able to verify rsCTL properties of a given icrrs $\mathcal{I}$, we need the set of the reachable states of the model $\mathcal{M}_{\mathcal{I}}$. Firstly, we describe an algorithm for computing all the reachable states and, later on, we provide a method for computing the set of states, where a given rsCTL 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 $\tau : 2^W \to 2^W$ 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 $m, n \leqslant |W|$ 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 $\mathcal{M}_{\mathcal{I}} = (\mathbb{W}, \mathsf{W}_0, \to, L)$ be a model. From Definition 5.3 it follows that any $w \in \mathbb{W}$ may have many different successors (at most $2^{|\mathcal{E}|}$). Thus we define the function that assigns the set of the Ψ-successors to the states in $W \subseteq \mathbb{W}$:

$$\mathsf{post}_{\Psi}(W) = \{w' \in \mathbb{W} | (\exists w \in W) w \to_{\Psi} w'\},$$

where $\Psi \subseteq 2^{\mathcal{E}}$.

The set of all the reachable states over $2^{\mathcal{E}}$ in $\mathcal{M}_{\mathcal{I}}$ is denoted by $\mathit{Reach}(\mathcal{M}_{\mathcal{I}})$. The set $\mathit{Reach}(\mathcal{M}_{\mathcal{I}})$ can be characterised by the following fixed point equation:

$$\mathit{Reach}(\mathcal{M}_{\mathcal{I}}) = \mu X.\big(\mathsf{W}_0 \cup X \cup \mathsf{post}_{2^{\mathcal{E}}}(X)\big).$$

Algorithm 1. The algorithm for computing the set $\mathit{Reach}(\mathcal{M}_{\mathcal{I}})$

```
1: X := W_0
2: X_p := ∅
3: while X ≠ X_p do
4:     X_p := X
5:     X := X ∪ post_{2^E}(X)
6: end while
7: return X
```

Algorithm 1 implements the fixed-point computation of the reachable states for a given model $\mathcal{M}_{\mathcal{I}} = (\mathbb{W}, \mathsf{W}_0, \to, L)$. Line 7 of the algorithm returns the set X which is equal to $\mathit{Reach}(\mathcal{M}_{\mathcal{I}})$.

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

$$\mathsf{pre}^{\exists}_{\Psi}(W) = \{w \in \mathit{Reach}(\mathcal{M}_{\mathcal{I}}) | (\exists w' \in W) w \to_{\Psi} w'\}.$$

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

- $[\![\neg\phi]\!] \stackrel{\text{def}}{=} \mathit{Reach}(\mathcal{M}_{\mathcal{I}}) \setminus [\![\phi]\!]$,
- $[\![\phi \wedge \psi]\!] \stackrel{\text{def}}{=} [\![\phi]\!] \cap [\![\psi]\!], [\![\phi \vee \psi]\!] \stackrel{\text{def}}{=} [\![\phi]\!] \cup [\![\psi]\!]$,
- $[\![\mathbf{E}_{\Psi}\mathbf{X}\phi]\!] \stackrel{\text{def}}{=} \mathsf{pre}^{\exists}_{\Psi}([\![\phi]\!])$.

The remaining operators are defined as the following fixed points:

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

In the case of $[\![\mathbf{E}_{\Psi}\mathbf{G}\phi]\!]$ the greatest fixed point computation is involved, which in each iteration removes states that do not have a Ψ-predecessor in which ϕ is satisfied. In the case of $[\![\mathbf{E}_{\Psi}[\phi\mathbf{U}\psi]]\!]$ the least fixed point computation is involved, such that in each iteration the Ψ-predecessors, in which ϕ is satisfied are added to the set of states in which ψ is satisfied. See Algorithms 4 and 3 for the pseudocode of the procedures implementing the described fixed point computations.

The overall procedure $\mathsf{check}_{\mathsf{rsCTL}(\phi)}$ for computing the set of states in which a given rsCTL formula ϕ holds is outlined in Algorithm 2.

Algorithm 2. Procedure $\mathsf{check}_{\mathsf{rsCTL}}(\phi)$

1: **if** $\phi \in S$ **then**
2: **return** $\{w \in \mathbb{W} | \phi \in w\}$
3: **end if**
4: **if** $\phi = \neg\phi_1$ **then**
5: **return** $Reach(\mathcal{M}) \setminus \mathsf{check}_{\mathsf{rsCTL}}(\phi_1)$
6: **end if**
7: **if** $\phi = \phi_1 \vee \phi_2$ **then**
8: **return** $\mathsf{check}_{\mathsf{rsCTL}}(\phi_1) \cup \mathsf{check}_{\mathsf{rsCTL}}(\phi_2)$
9: **end if**
10: **if** $\phi = \mathbf{E}_\Psi \mathbf{X}\phi_1$ **then**
11: **return** $\mathsf{pre}^{\exists}_{\Psi}(\mathsf{check}_{\mathsf{rsCTL}}(\phi_1))$
12: **end if**
13: **if** $\phi = \mathbf{E}_\Psi \mathbf{G}\phi_1$ **then**
14: **return** $\mathsf{checkEG}(\Psi, \phi_1)$
15: **end if**
16: **if** $\phi = \mathbf{E}_\Psi[\phi_1 \mathbf{U} \phi_2]$ **then**
17: **return** $\mathsf{checkEU}(\Psi, \phi_1, \phi_2)$
18: **end if**

We consider now the complexity of $\mathsf{check}_{\mathsf{rsCTL}}(\phi)$. The algorithm processes ϕ recursively, at each recursion level dealing with a single subformula of ϕ. The entire algorithm requires time $\mathcal{O}(2^{|S|} \cdot (2^{2\cdot|S|} + \mathsf{c}(\phi)) \cdot |A| \cdot |S| \cdot \mathsf{d}(\phi))$. This follows from the complexity of computations at a single recursion level and the number of recursion levels required to process the subformulae. To process all the subformulae the algorithm requires $\mathcal{O}(\mathsf{d}(\phi))$ recursion levels. Let us consider a single recursion level where a subformula of the form $\mathbf{E}_\Psi[\phi \mathbf{U} \psi]$ is processed (Algorithm 3). In each iteration of the loop we find the new value of X, which is a set consisting of all the states in which ψ holds, and all the Ψ-predecessors of the states in X, in which ϕ also holds (Line 5 of Algorithm 3). Firstly, we consider the complexity of computing the intersection $Y_\phi \cap \mathsf{pre}^{\exists}_{\Psi}(X)$. The operation could be implemented as finding all $w \in [\![\phi]\!]$ by checking if $w' = res_A(w \cup \alpha)$ for each $w \in [\![\phi]\!], \alpha \in \Psi$, and $w' \in X$. This involves three nested loops iterating over $[\![\phi]\!], X \subseteq 2^S$ and $\Psi \subseteq 2^{\mathcal{E}}$, thus the operation requires time $\mathcal{O}((2^{2\cdot|S|} + |\Psi|) \cdot |A| \cdot |S|)$. The result for $res_A(w \cup \alpha)$ is obtained in $\mathcal{O}(|A| \cdot |S|)$, while the computation of the sum with the set Y_ψ has the complexity of $\mathcal{O}(|S|)$, giving us the overall complexity of $\mathcal{O}((2^{2\cdot|S|} + |\Psi|) \cdot |A| \cdot |S|)$ for the sum and the intersection. The loop of the algorithm may iterate at most $2^{|S|}$ times. Therefore, the complexity of the algorithm for computing $[\![\mathbf{E}_\Psi[\phi_1 \mathbf{U} \phi_2]]\!]$ is $\mathcal{O}(2^{|S|} \cdot (2^{2\cdot|S|} + |\Psi|) \cdot |A| \cdot |S|)$. In the case of $\mathbf{E}_\Psi \mathbf{G}\phi$, in each iteration of the loop we find all the Ψ-predecessors of the states in X, in which ϕ also holds (Line 5 of Algorithm 4). Similarly as in the case of $\mathbf{E}_\Psi \mathbf{G}\phi$, this is the same as finding all $w \in [\![\phi]\!]$ by checking if $w' = res_A(w \cup \alpha)$ for each $w \in [\![\phi]\!], \alpha \in \Psi$, and $w' \in X$, which requires time $\mathcal{O}((2^{2\cdot|S|} + |\Psi|) \cdot |A| \cdot |S|)$. Given the main loop iterates at most $2^{|S|}$ times, the procedure requires time $\mathcal{O}(2^{|S|} \cdot (2^{2\cdot|S|} + |\Psi|) \cdot |A| \cdot |S|)$. The remaining operators do not involve fixed point computations and they require at most time $\mathcal{O}((2^{2\cdot|S|} + |\Psi|) \cdot |A| \cdot |S|)$. Therefore, the time complexity of the algorithm for computing $[\![\phi]\!]$ for an rsCTL formula ϕ is $\mathcal{O}(2^{|S|} \cdot (2^{2\cdot|S|} + \mathsf{c}(\phi)) \cdot |A| \cdot |S| \cdot \mathsf{d}(\phi))$.

Algorithm 3. Procedure $\mathsf{checkEU}(\Psi, \phi, \psi)$

1: $X := \emptyset, X_p := Reach(\mathcal{M}_\mathcal{I})$
2: $Y_\phi := \mathsf{check}_{\mathsf{rsCTL}}(\phi), Y_\psi := \mathsf{check}_{\mathsf{rsCTL}}(\psi)$
3: **while** $X \neq X_p$ **do**
4: $X_p := X$
5: $X := Y_\psi \cup (Y_\phi \cap \mathsf{pre}^{\exists}_{\Psi}(X))$
6: **end while**
7: **return** X

Algorithm 4. Procedure checkEG(Ψ, ϕ)

1: $X := Reach(\mathcal{M}_{\mathcal{I}}), X_p := \emptyset$
2: $Y_\phi := \mathsf{check}_{\mathsf{rsCTL}}(\phi)$
3: **while** $X \neq X_p$ **do**
4: $\quad X_p := X$
5: $\quad X := \left(Y_\phi \cap \mathsf{pre}^{\exists}_{\Psi}(X)\right)$
6: **end while**
7: **return** X

Next, we prove that the rsCTL model checking problem for icrrs is PSPACE-complete.

Lemma 6.1. *Given an icrrs $\mathcal{I} = ((S, A, \mathcal{E}), S_0)$ and an rsCTL formula ϕ, the problem of deciding whether $\mathcal{M}_{\mathcal{I}} \models \phi$ is* PSPACE*-hard.*

Proof. The proof is by reduction of QSAT,[1] which is a known PSPACE-complete problem [31], to the rsCTL model checking problem. The construction used for the reduction is similar to the one of [19] for timed automata and TCTL. Let $PV = \{x_1, x_2, \ldots, x_n\}$ be a set of propositional variables, $\beta(x_1, x_2, \ldots, x_n)$ be a boolean formula in 3-CNF, and $\gamma = \mathcal{Q}_1 x_1 \mathcal{Q}_2 x_2 \ldots \mathcal{Q}_n x_n \beta(x_1, x_2, \ldots, x_n)$ be a quantified boolean formula, where $\mathcal{Q}_i = \exists$ if i is odd, $\mathcal{Q}_i = \forall$ if i is even. Then, the QSAT problem consists in deciding if the formula γ is true.

We define an additional set of the negated propositional variables $\overline{PV} = \{\bar{x} | x \in PV\}$ and assume that β is represented as the conjunction of m clauses: $\beta = c_1 \wedge c_2 \wedge \ldots \wedge c_m$, where $c_i = (l_{i,1} \vee l_{i,2} \vee l_{i,3})$ with $l_{i,j} \in (PV \cup \overline{PV})$ for all $0 < i \leqslant m, 1 \leqslant j \leqslant 3$. Moreover, for each clause c we define the set $vars(c) = \{0 < k \leqslant n | x_k \in PV \text{ is in } c\}$ and the set $\overline{vars}(c) = \{0 < k \leqslant n | \bar{x}_k \in \overline{PV} \text{ is in } c\}$. Next, we define the icrrs which we use for the translation. Let $V = \{p_1, \bar{p}_1, \ldots, p_n, \bar{p}_n\}$ be a set of the entities that represent the propositional variables and their negations, and $C = \{\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_m\}$ be the set of the entities that correspond to the clauses. The entity t is used to indicate that under the considered valuation the formula γ is true. The entity h is used as the inhibitor of the reactions where no inhibitors are needed for the translation to work. This guarantees that the inhibitor set is non-empty. Then, the background set is defined as $S = V \cup C \cup \{t, h\}$. Next, we define the following sets of reactions:

- $P_i = \{(\{p_i\}, \{h\}, \{p_i\}), (\{\bar{p}_i\}, \{h\}, \{\bar{p}_i\})\}$ for $0 < i \leqslant n$,
- $L_i = \{(\{p_k\}, \{\bar{p}_k\}, \{\hat{c}_i\}) | k \in vars(i)\} \cup \{(\{\bar{p}_k\}, \{p_k\}, \{\hat{c}_i\}) | k \in \overline{vars}(i)\}$ for $0 < i \leqslant n$,
- $F = \{(\{\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_n\}, \{h\}, \{t\})\}$.

The set P_i contains the reactions responsible for preserving the valuations of the variables along the execution sequences. The reactions of L_i produce entities that indicate whether a single clause is satisfied, whereas the reaction of F produces the entity indicating that all the clauses are satisfied. The set of all the reactions of the icrrs is defined as $A = \bigcup_{i=1}^{n} P_i \cup \bigcup_{i=1}^{n} L_i \cup F$. As the context entities $\mathcal{E}$ we use the entities corresponding to the propositional variables, i.e., $\mathcal{E} = \{p_1, \bar{p}_1, \ldots, p_n, \bar{p}_n\}$. We assume that the set of the initial states contains only the empty set, i.e., $S_0 = \{\emptyset\}$. Then, the icrrs for the described QSAT problem and the formula γ is defined as $\mathcal{I}_{\mathbf{QSAT}} = ((S, A, \mathcal{E}), S_0)$. We define the following rsCTL formulae for all $1 \leqslant i \leqslant n$:

$$\phi_i = \begin{cases} \mathbf{A}_{\{\{p_i\},\{\bar{p}_i\}\}} \mathbf{X} \phi_{i+1} & \text{if } \mathcal{Q}_i = \forall, \\ \mathbf{E}_{\{\{p_i\},\{\bar{p}_i\}\}} \mathbf{X} \phi_{i+1} & \text{if } \mathcal{Q}_i = \exists. \end{cases}$$
$$\phi_{n+1} = \mathbf{E}_{\{\emptyset\}} \mathbf{X} t$$

The formula ϕ_1 consists of n nested next-state operators that restrict the choice of entities either to p_i or $\bar{p}_i$ (no contradictions are allowed) at each level $0 < i \leqslant n$. For the level $n + 1$ in the formula, we check if there exists a successor state in which the entity t exists, indicating that γ is true. The assumed set $\mathcal{E}$ allows us to generate all the valuations that need to be considered. Then, we restrict these valuations using the rsCTL formula according to the QSAT quantification.

Then, it is easy to see that γ is true if and only if $\mathcal{M}_{\mathcal{I}_{\mathbf{QSAT}}} \models \phi_1$. □

Lemma 6.2. *Given an icrrs $\mathcal{I} = ((S, A, \mathcal{E}), S_0)$ and an rsCTL formula ϕ, the problem of deciding whether $\mathcal{M}_{\mathcal{I}} \models \phi$ is in* PSPACE.

Proof. To prove that the problem is in PSPACE we show that there exists a nondeterministic algorithm for deciding whether $\mathcal{M}_{\mathcal{I}} \models \phi$ that requires at most polynomial space in the size of the input, i.e., the formula ϕ and the icrrs $\mathcal{I}$. The proof is similar to the one of [1] for TCTL interpreted over timed graphs.

[1] Quantified SAT (QSAT) is a problem which consists in checking whether a quantified boolean formula is in the language of TQBF (true quantified boolean formulae).

The algorithm uses the recursive procedure $label(w, \phi)$ which returns *true* iff $\mathcal{M}_{\mathcal{I}}, w \models \phi$, where $w \subseteq S$; otherwise, it returns *false*. The encoding of each state requires space $\mathcal{O}(|S|)$ and each successor can be generated in space $\mathcal{O}(|S|)$, whereas the overall algorithm requires space $\mathcal{O}(|S| \cdot \mathsf{d}(\phi))$.

The proof follows by the induction on the length of the formula ϕ. The cases where ϕ does not contain temporal operators or $\phi = \mathbf{E}_{\Psi}\mathbf{X}\phi_1$ are straightforward.

The nondeterministic procedure for checking $\phi = \mathbf{E}_{\Psi}[\phi_1 \mathbf{U} \phi_2]$ in $w \subseteq S$ is outlined in Algorithm 5. It nondeterministically chooses states and actions of a path over Ψ, checks at each step if the state chosen is a successor of the previous state via the action chosen, 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 5. Procedure for checking $\mathbf{E}_{\Psi}[\phi_1 \mathbf{U} \phi_2]$ in $w \subseteq S$

```
1: ŵ := w
2: checking:
3: if label(ŵ, φ2) then
4:    return true
5: end if
6: if ¬label(ŵ, φ1) then
7:    return false
8: end if
9: guessing:
10: guess ŵ′ ⊆ S
11: guess α ∈ Ψ
12: if ŵ′ ≠ res_A(ŵ ∪ α) ∨ ¬label(ŵ′, φ1) then
13:    goto guessing
14: else
15:    ŵ := ŵ′
16:    goto checking
17: end if
```

The procedure for checking $\phi = \mathbf{E}_{\Psi}\mathbf{G}\phi_1$ in $w \subseteq S$ is outlined in Algorithm 6. Similarily to the previous case, the algorithm guesses a path over Ψ, 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 for detecting a loop. 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*.

Algorithm 6. Procedure for checking $\mathbf{E}_{\Psi}\mathbf{G}\phi_1$ in $w \subseteq S$

```
1: ŵ := w
2: L := false
3: if ¬label(ŵ, φ1) then
4:    return false
5: end if
6: guessing:
7: guess ŵ′ ⊆ S
8: guess α ∈ Ψ
9: if ŵ′ ≠ res_A(ŵ ∪ α) ∨ ¬label(ŵ′, φ1) then
10:    goto guessing
11: end if
12: if ¬L then
13:    guess γ ∈ {true, false}
14:    if γ then
15:       ŵ_l := ŵ′
16:       L := true
17:    end if
```

18: **else**
19: **if** $\widehat{w}' = \widehat{w}_l$ **then**
20: **return** true
21: **end if**
22: **end if**
23: $\widehat{w} := \widehat{w}'$
24: **goto** *guessing*

The procedure returns false if no sequence for which the procedure returns *true* could be found. To ensure that the procedure terminates, for each sequence guessed the procedure nondeterministically chooses a state $\widehat{w}_r$ of that sequence. The guessing of the sequence stops when the newly guessed state is $\widehat{w}_r$. For simplicity of the presentation, this part of the procedure is not included in Algorithms 5 and 6.

To check if $\mathcal{M}_{\mathcal{I}} \models \phi$, the procedure *label* is executed for all the initial states to check if ϕ holds for all $w \in S_0$. For each execution, 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 $\mathcal{O}(\mathrm{d}(\phi))$ calls of the *label* procedure, where a single call needs space $\mathcal{O}(|S|)$. The space requirement for the procedure is not affected by the size of Ψ 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 rsCTL has at most two arguments. Thus, the overall space requirement of the procedure is $\mathcal{O}(|S| \cdot \mathrm{d}(\phi))$. However, if we assume that we include the size of the formula in our space complexity considerations, then the procedure needs space $\mathcal{O}((|S| + \mathrm{c}(\phi)) \cdot \mathrm{d}(\phi))$. Therefore, by Savitch's theorem, the deterministic algorithm can be implemented in polynomial space. □ The following theorem follows directly from Lemmas 6.1 and 6.2:

Theorem 6.3. *The model checking problem for rsCTL is* PSPACE*-complete.*

This rsCTL verification method can be performed symbolically using binary decision diagrams (BDDs) [6]. The operations on sets of states, such as union, intersection, and difference, can be carried out efficiently on BDDs representing boolean functions.

7. Encoding icrrs into boolean functions

In this section we provide an encoding of the initialised context restricted reaction system into boolean functions in order to be used for symbolic model checking. We start with defining the *symbolic model checking problem for rsCTL*.

Let $\mathcal{I}$ be an icrrs and ϕ be an rsCTL formula. The symbolic model checking problem for rsCTL consists in deciding whether $\mathcal{M}_{\mathcal{I}} \models \phi$, using boolean formulae for representing the model $\mathcal{M}_{\mathcal{I}}$.

The general idea of the encoding into boolean functions is similar to the one introduced for reaction systems in [13], but it differs in how the encoding of the transitions between states is defined.

Let $\mathcal{I} = ((S, A, \mathcal{E}), S_0)$ be an icrrs and $\mathcal{M}_{\mathcal{I}} = (\mathbb{W}, \mathrm{W}_0, \rightarrow, L)$ be its model. We denote the elements of S by $e_1, \ldots, e_n$, where $n = |S|$. We introduce sets of the propositional variables used in the encoding. The states of the model are encoded using the set $\mathrm{P} = \{\mathrm{p}_1, \ldots, \mathrm{p}_n\}$. The actions representing context entities are encoded with the variables $\mathrm{P}^{\mathcal{E}} = \{\mathrm{p}_1^{\mathcal{E}}, \ldots, \mathrm{p}_n^{\mathcal{E}}\}$. To encode the successors in the transition relation, we use the set $\mathrm{P}' = \{\mathrm{p}'_1, \ldots, \mathrm{p}'_n\}$ of the primed variables. The set of reactions producing $e \in S$ is defined as $Prod(e) = \{a \in A | e \in P_a\}$. For brevity, we use the following vectors of variables: $\bar{\mathbf{p}} = (\mathrm{p}_1, \ldots, \mathrm{p}_n)$, $\bar{\mathbf{p}}^{\mathcal{E}} = (\mathrm{p}_1^{\mathcal{E}}, \ldots, \mathrm{p}_n^{\mathcal{E}})$, $\bar{\mathbf{p}}' = (\mathrm{p}'_1, \ldots, \mathrm{p}'_n)$. Moreover, we define the following functions: $\mathtt{m} : S \to \mathrm{P}$, $\mathtt{m}' : S \to \mathrm{P}'$, $\mathtt{m}^{\mathcal{E}} : S \to \mathrm{P}^{\mathcal{E}}$, such that $\mathtt{m}(e_i) = \mathrm{p}_i, \mathtt{m}'(e_i) = \mathrm{p}'_i$, $\mathtt{m}^{\mathcal{E}}(e_i) = \mathrm{p}_i^{\mathcal{E}}$, for all $1 \leqslant i \leqslant n$. The functions map the background set entities to the corresponding variables of the encoding.

Single state. A state $w \in \mathbb{W}$ is encoded as the conjunction of all the variables corresponding to the entities that are present in w, and the conjunction of all the negations of the variables that are not present in w:

$$\mathrm{St}_w(\bar{\mathbf{p}}) = \left(\bigwedge_{e \in w} \mathtt{m}(e)\right) \wedge \left(\bigwedge_{e \in (S \setminus w)} \neg \mathtt{m}(e)\right).$$

Sets of states. A set $W \subseteq \mathbb{W}$ is encoded as the disjunction of all the encoded states that are in $\mathbb{W}$:

$$\mathrm{SetSt}_W(\bar{\mathbf{p}}) = \bigvee_{w \in W} \mathrm{St}_w(\bar{\mathbf{p}}).$$

Context sets. A set $\alpha \subseteq S$ of context entities is encoded as follows:

$$\mathrm{Ct}_\alpha(\bar{\mathbf{p}}^{\mathcal{E}}) = \left(\bigwedge_{e \in \alpha} \mathtt{m}^{\mathcal{E}}(e)\right) \wedge \left(\bigwedge_{e \in (S \setminus \alpha)} \neg \mathtt{m}^{\mathcal{E}}(e)\right).$$

Sets of context sets. A set $\Psi \subseteq 2^{\mathcal{E}}$ of sets of context entities is encoded as the disjunction of all the encoded sets of Ψ:

$$\mathsf{SetCt}_{\Psi}(\bar{\mathbf{p}}^{\mathcal{E}}) = \bigvee_{\alpha \in \Psi} \mathsf{Ct}_{\alpha}(\bar{\mathbf{p}}^{\mathcal{E}}).$$

Entity production condition. A single entity $e \in S$ can be produced if there exists a reaction $a \in \mathit{Prod}(e)$ that is enabled, that is, all of the variables corresponding to reactants of a (including the variables representing the context) are true and all of the variables corresponding to inhibitors of a (also including the variables representing the context) are false. The formula encoding this is defined as follows:

– if $\mathit{Prod}(e) \neq \emptyset$, then

$$\mathsf{En}_e(\bar{\mathbf{p}}, \bar{\mathbf{p}}^{\mathcal{E}}) = \bigvee_{a \in \mathit{Prod}(e)} \left(\bigwedge_{e' \in R_a} (\mathtt{m}(e') \vee \mathtt{m}^{\mathcal{E}}(e')) \wedge \bigwedge_{e' \in I_a} \neg(\mathtt{m}(e') \vee \mathtt{m}^{\mathcal{E}}(e')) \right);$$

– if $\mathit{Prod}(e) = \emptyset$, then

$$\mathsf{En}_e(\bar{\mathbf{p}}, \bar{\mathbf{p}}^{\mathcal{E}}) = \mathit{false}.$$

Entity production. The function $\mathsf{Pr}_e(\bar{\mathbf{p}}, \bar{\mathbf{p}}^{\mathcal{E}}, \bar{\mathbf{p}}')$ encodes the production of a single entity $e \in S$. When the production of e is enabled, then the variable corresponding to e is required to be true; otherwise, the variable is required to be false.

$$\mathsf{Pr}_e(\bar{\mathbf{p}}, \bar{\mathbf{p}}^{\mathcal{E}}, \bar{\mathbf{p}}') = \left(\mathsf{En}_e(\bar{\mathbf{p}}, \bar{\mathbf{p}}^{\mathcal{E}}) \wedge \mathtt{m}'(e)\right) \vee \left(\neg \mathsf{En}_e(\bar{\mathbf{p}}, \bar{\mathbf{p}}^{\mathcal{E}}) \wedge \neg \mathtt{m}'(e)\right).$$

Permitted context sets. The following formula encodes the permitted context sets by blocking the not allowed context entities.

$$\mathsf{PCt}(\bar{\mathbf{p}}^{\mathcal{E}}) = \bigwedge_{e \in (S \setminus \mathcal{E})} \neg \mathtt{m}^{\mathcal{E}}(e).$$

Transition relation. To encode the transition relation, we define the function that is the conjunction of the entity production encodings for all the background set entities and restricts the allowed context sets.

$$\mathsf{Tr}(\bar{\mathbf{p}}, \bar{\mathbf{p}}^{\mathcal{E}}, \bar{\mathbf{p}}') = \bigwedge_{e \in S} \mathsf{Pr}_e(\bar{\mathbf{p}}, \bar{\mathbf{p}}^{\mathcal{E}}, \bar{\mathbf{p}}') \wedge \mathsf{PCt}(\bar{\mathbf{p}}^{\mathcal{E}}).$$

The described encoding method can be used for the symbolic model checking problem for rsCTL. In the following theorem we state the complexity of this problem.

Theorem 7.1. *The symbolic model checking problem for rsCTL is* PSPACE*-complete.*

Proof. The theorem follows from the fact that the symbolic model checking for CTL is PSPACE-complete [20] and that the transition relation of an icrrs can be represented (encoded in polynomial time) with a boolean function, as demonstrated above. Moreover, the complexity of the verification algorithm for rsCTL does not change with respect to CTL. This follows from the fact that rsCTL only restricts the choice of the transitions to be considered. Therefore, in the symbolic model checking algorithm we replace the function computing the predecessors of a given state with the function $\mathsf{pre}^{\exists}_{\Psi}(X)$ which restricts the choice of the predecessors to only those accessible via Ψ (see Section 6). In the symbolic model checking algorithm this requires one more conjunction to be computed at each iteration, therefore the modification introduces only a difference of a constant in the overall complexity. □

8. Experimental results

We have implemented a tool for verification of icrrs, based on the encoding presented in Section 7, and binary decision diagrams (BDDs) for storing and manipulating the encoded boolean functions. The tool allows for verification of rsCTL properties, as described in Section 5. It was implemented using the C++ programming language, and it uses CUDD library [30] for operations on BDDs.

In our implementation the transition relations can be stored and applied in two different ways, by using either a *monolithic* or *partitioned* encoding [7]. In the monolithic encoding, when computing successors (or predecessors) of a set of states, we compute the conjunction of the formula encoding the set of states and of the single formula encoding the transition relation. In the partitioned encoding, the formula encoding the transition relation is partitioned into separate formulae, where each formula encodes the production of a single entity. Then, the conjunction is computed with each partition of the formula.

The order of the BDD variables may have a significant impact on the performance of the operations on BDDs. Therefore, 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 unprimed BDD variables are interleaved. In the case of the automatic reordering of the variables we employ Rudell's sifting algorithm [25] implemented in CUDD.

Moreover, the implementation uses the BDD-based bounded model checking approach [8] for testing the existential 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.

We test our implementation[2] using four different benchmarks which we describe next, together with the experimental results.

8.1. Heat shock response model

Firstly, we test our implementation using the qualitative model of the eukaryotic heat shock response (HSR) introduced in [4]. HSR is an internal repair mechanism triggered when a cell is subjected to an environmental stressor – increased temperature that is not ideal for its functioning.

A temperature exceeding the ideal temperature causes the proteins (*prot*) of a cell to misfold (*mfp*), which in turn may cause its malfunctioning. To facilitate refolding of the proteins, heat shock response proteins (*hsp*) are produced, which are molecular chaperones for the misfolded proteins. The production of *hsp* is initiated by heat shock factors (*hsf*) which are, dimerised (hsf_2), and then trimerised (hsf_3). Next, hsf_3 activates *hsp* production by binding to the heat shock element (*hse*) which is the promoter-site of the gene encoding the heat shock proteins.

The background set of the icrrs modelling HSR is defined as $S = \{hsp, hsf, hsf_2, hsf_3, hse, mfp, prot, hsf_3 : hse, hsp : hsf, hsp : mfp, stress, nostress, h\}$. The entities used in the model are summarised in Table 1. Then, the set A of the reactions is composed of the following elements:

- $(\{hsf\}, \{hsp\}, \{hsf_3\})$,
- $(\{hsf, hsp, mfp\}, \{h\}, \{hsf_3\})$,
- $(\{hsf_3\}, \{hsp, hse\}, \{hsf\})$,
- $(\{hsp, hsf_3, mfp\}, \{hse\}, \{hsf\})$,
- $(\{hsf_3, hse\}, \{hsp\}, \{hsf_3 : hse\})$,
- $(\{hsp, hsf_3, mfp, hse\}, \{h\}, \{hsf_3 : hse\})$,
- $(\{hse\}, \{hsf_3\}, \{hse\})$,
- $(\{hsp, hsf_3, hse\}, \{mfp\}, \{hse\})$,
- $(\{hsf_3 : hse\}, \{hsp\}, \{hsp, hsf_3 : hse\})$,
- $(\{hsp, mfp, hsf3 : hse\}, \{h\}, \{hsp, hsf_3 : hse\})$,
- $(\{hsf, hsp\}, \{mfp\}, \{hsp : hsf\})$,
- $(\{hsp : hsf, stress\}, \{nostress\}, \{hsf, hsp\})$,
- $(\{hsp : hsf, nostress\}, \{stress\}, \{hsp : hsf\})$,
- $(\{hsp, hsf_3\}, \{mfp\}, \{hsp : hsf\})$,
- $(\{hsp, hsf_3 : hse\}, \{mfp\}, \{hse, hsp : hsf\})$,
- $(\{stress, prot\}, \{nostress\}, \{mfp, prot\})$,
- $(\{nostress, prot\}, \{stress\}, \{prot\})$,
- $(\{hsp, mfp\}, \{h\}, \{hsp : mfp\})$,
- $(\{mfp\}, \{hsp\}, \{mfp\})$,
- $(\{hsp : mfp\}, \{h\}, \{hsp, prot\})$.

Next, we define the icrrs corresponding to the HSR model as follows:

$$\mathcal{I}_{hsr} = ((S, A, \{stress, nostress\}), S_0),$$

where $S_0 = \{\{hsf, prot, hse, nostress\}, \{hse, prot, hsp : hsf, stress\}, \{hsp, prot, hsf_3 : hse, mfp, hsp : mfp, nostress\}\}$. The definition of S_0 corresponds to the initial context sets used in the processes analysed in [4].

In [4], six properties essential for the functioning of the HSR model were formulated. It is assumed that either *stress* or *nostress* is supplied by the context sets, but never both. Then, the following properties are specified:

P1. *mass-conservation*: if *hse* or $hsf_3 : hse$ is present, then allways *hse* or $hsf_3 : hse$ will be present in the next state;
P2. *a single form of hse*: if *hse* and $hsf_3 : hse$ are not present simultaneously, then they will never be present in the next state;
P3. *mass-conservation of prot*: if *prot* is present, then it will be present in the next state as well;
P4. *misfolded proteins must be addressed*: presence of *mfp* must always result in *mfp* of $hsp : mfp$ in the next state;
P5. *a single form of hsf*: if at most one form of *hsf* is present ($hsf, hsf_3, hsf_3 : hse, hsp : hsf$), then it will also be present in at most one form in the next state;

[2] The implementation together with the benchmarks may be downloaded from http://www.ipipan.waw.pl/~meski/rs.

Table 1
Summary of the entities used in the heat shock response model. The entities used in the context sets are marked with ♢.

Entity	Description
hsp	heat shock protein
hsf	heat shock factor
hsf_2	dimerised heat shock factor
hsf_3	trimerised heat shock factor
hse	heat shock element
mfp	misfolded protein
prot	protein
$hsf_3 : hse$	hsf_3 bound with *hse*
$hsp : mfp$	*hsp* bound with *mfp*
$hsp : hsf$	complex consisting of *hsp* and *hsf*
stress	presence of the heat shock (42 °C)♢
nostress	absence of the heat shock (37 °C)♢
h	dummy inhibitor

P6. *stability of hsp*: in the absence of *stress*, if $hsp : hsf$ is present, then it is preserved in the next state as well;

The above properties can be formalised using rsCTL. Following the assumption about context sets, we use the set $C = \{\{stress\}, \{nostress\}\}$ which appears in some formulae as the set of the allowed context sets specifying that the system is either under stress or it is operating in normal conditions, i.e., conditions which do not trigger the heat shock response. The following formulae express the properties specified in **P1**–**P6**:

- **P1**: $\psi_1 = \mathbf{A}_C\mathbf{G}(hse \vee hsf3 : hse \Rightarrow \mathbf{A}_C\mathbf{X}(hse \vee hsf_3 3 : hse))$,
- **P2**: $\psi_2 = \mathbf{A}_C\mathbf{G}(\neg(hse \wedge hsf_3 : hse) \Rightarrow \mathbf{A}_C\mathbf{X}\neg(hse \wedge hsf3 : hse))$,
- **P3**: $\psi_3 = \mathbf{A}_C\mathbf{G}(prot \Rightarrow \mathbf{A}_C\mathbf{X}prot)$,
- **P4**: $\psi_4 = \mathbf{A}_C\mathbf{G}(mfp \Rightarrow \mathbf{A}_C\mathbf{X}(mfp \vee hsp : mfp))$,
- **P5**: $\psi_5 = \mathbf{A}_C\mathbf{G}(((hsf \oplus hsf_3 \oplus hsf_3 : hse \oplus hsp : hsf) \vee \neg(hsf \vee hsf_3 \vee hsf_3 : hse \vee hsp : hsf)) \Rightarrow \mathbf{A}_C\mathbf{X}((hsf \oplus hsf_3 \oplus hsf_3 : hse \oplus hsp : hsf) \vee \neg(hsf \vee hsf_3 \vee hsf_3 : hse \vee hsp : hsf)))$,
- **P6**: $\psi_6 = \mathbf{A}_{\{\{nostress\}\}}\mathbf{G}(hsp : hsf \Rightarrow \mathbf{A}_{\{\{nostress\}\}}\mathbf{X}hsp : hsf)$.

The specified properties were verified using our tool and proved to hold in the defined model. The verification of the heat shock response model appeared to be computationally easy for the tool – each property was successfully verified in under one second with less than 25 MB of memory use.

In the remainder of this section we introduce scalable benchmarks which allow us to test the efficiency of our implementation in more demanding settings.

8.2. Binary counter

We test our implementation by verifying properties of the bit counter described in Example 4.2 using the following formulae:

- $\psi_1 = \mathbf{AG}((\neg p_0 \wedge \ldots \wedge \neg p_{n-1}) \Rightarrow \mathbf{E}_{\{\{inc\},\{dec\}\}}\mathbf{F}(p_0 \wedge \ldots \wedge p_{n-1}))$,
- $\psi_2 = \mathbf{AG}((\neg p_0 \wedge \ldots \wedge \neg p_{n-1}) \Rightarrow \mathbf{A}_{\{\{inc\}\}}\mathbf{X}(p_0 \wedge \ldots \wedge p_{n-1}))$,
- $\psi_3 = \mathbf{E}_{\{\{inc\}\}}\mathbf{F}(p_0 \wedge \ldots \wedge p_{n-7} \wedge \neg p_{n-8} \wedge \ldots \wedge \neg p_{n-1})$ for $n > 8$,
- $\psi_4 = \mathbf{AG}(p_{n-1} \Rightarrow \mathbf{EF}\neg p_{n-1})$.

The first two formulae that we use in our benchmarks were specified in Example 5.11. The formula ψ_3 expresses that it is possible to finally reach the value 2^{n-8} by only increasing the counter, whereas ψ_4 means that it is always possible to finally change the value of the last bit from enabled to disabled.

In the case of ψ_1 (Fig. 2), ψ_2 (Fig. 3), and ψ_4 (Fig. 5) it was necessary to compute all the reachable states of the verified model and in these cases the verification results were similar in terms of time and memory consumption. The use of the partitioned transition relation induced an obvious time consumption penalty in the analysed cases, with no significant memory consumption improvements. In the case of ψ_3 (Fig. 4), the bounded model checking heuristic was applied, because the formula is in the existential form. This allowed for earlier termination of the state-space search resulting in a significant speed up – when the monolithic transition relation encoding was used, all the verification results for up to 25 bits were obtained in under one minute. The use of the automatic reordering of the BDD variables did not significantly affect the performance in any of the analysed cases of the bit counter system.

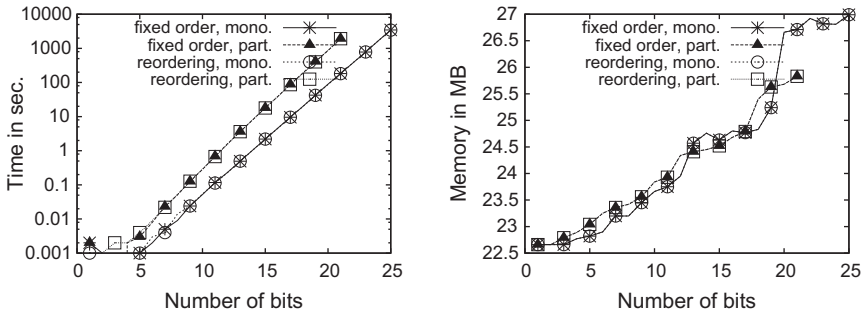

Fig. 2. Bit counter, formula ψ_1.

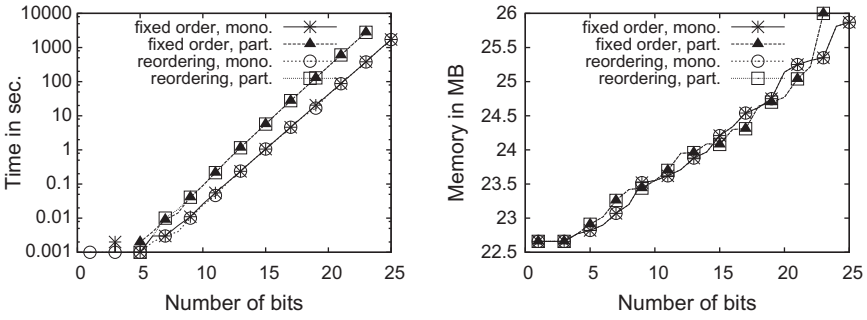

Fig. 3. Bit counter, formula ψ_2.

8.3. Mutual exclusion protocol

Here we use icrrs to define a mutual exclusion protocol. The system consists of n processes competing for exclusive access to the critical section. The background set of the icrrs modelling the mutual exclusion protocol is defined as:

$$S_n = \{out_1, \ldots, out_n, req_1, \ldots, req_n, in_1, \ldots, in_n, lock, done, act_1, \ldots, act_n, s\}.$$

Initially all the processes are not in their critical sections and are not requesting the access, which is indicated by the presence of out_i for each $i \in \{1, \ldots, n\}$. The context may supply any subset of $\{act_1, \ldots, ack_n\}$. We assume that if the context contains act_i for some $i \in \{1, \ldots, n\}$, then it is the i-th process' turn to perform an action. The i-th process requests an access to its critical section by producing req_i. Then, it is possible for the process to enter the critical section when it is allowed to perform an action and the critical section is not locked (the *lock* entity is not present). In the case of entering a critical section, to avoid the situation where two processes enter their critical sections synchronously, the assumption on act_i is stricter: only one act_i for some $i \in \{1, \ldots, n\}$ is allowed to be present for the process to enter the critical section. When a process enters its critical section, the critical section is locked by production of the *lock* entity. The *lock* entity is preserved until the entity *done* appears, which is produced when a process leaves its critical section. Any reaction in the system may be inhibited by the s entity.

Let P_i be the set of reactions of the i-th process, for $i \in \{1, \ldots, n\}$. Then, P_i consists of the following reactions:

- $(\{out_i, act_i\}, \{s\}, \{req_i\})$,
- $(\{out_i\}, \{act_i\}, \{out_i\})$,
- $(\{req_i, act_i, act_j\}, \{s\}, \{req_i\})$ for each $j \in \{1, \ldots, n\}$ such that $i \neq j$,
- $(\{req_i\}, \{act_i\}, \{req_i\})$,
- $(\{req_i, act_i\}, \{act_j | j \in \{1, \ldots, n\} \text{and} j \neq i\} \cup \{lock\}, \{in_i, lock\})$,
- $(\{in_i, act_i\}, \{s\}, \{out_i, done\})$,
- $(\{in_i\}, \{act_i\}, \{in_i\})$.

The set of reactions is defined as $A_n = \bigcup_{i=1}^{n} P_i \cup \{(\{lock\}, \{done\}, \{lock\})\}$. Then, the icrrs for the system of n processes is defined as follows:

$$\mathcal{I}_{me}^{n} = ((S_n, A_n, \{act_1, \ldots, act_n\}), \{\{out_1, \ldots, out_n\}\}).$$

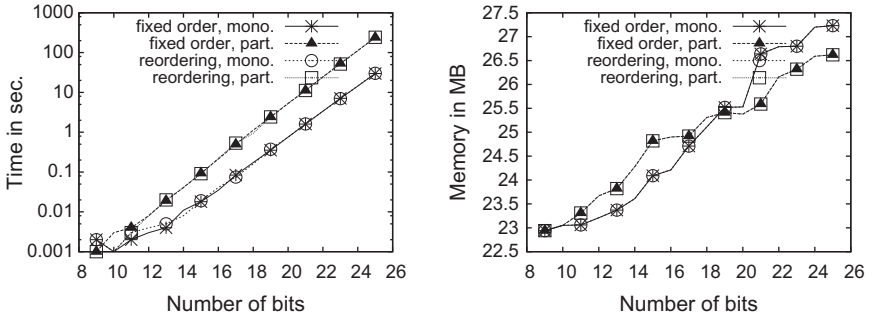

Fig. 4. Bit counter, formula ψ_3.

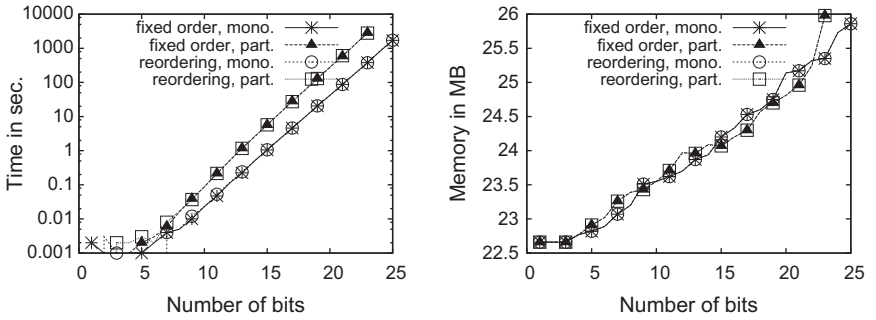

Fig. 5. Bit counter, formula ψ_4.

In our experiments, we test the following formulae:

- $\psi_1 = \mathbf{A}_{\{\{act_1\}\}}\mathbf{F}in_1$,
- $\psi_2 = \mathbf{E}_{\{\{act_1\}\}}\mathbf{F}in_1$,
- $\psi_3 = \mathbf{AG}\left(\bigwedge_{i\neq j}^{n} \neg(in_i \wedge in_j)\right)$.

The formula ψ_1 means that for all the executions with the act_1 supplied by the context sets, meaning that the first process is requesting access to its critical section, it is possible that the first process enters its critical section. The formula ψ_2 expresses the existential counterpart of the same property. The formula ψ_3 expresses the mutual exclusion property, i.e., in all the possible executions (with no context restrictions), globally it is the case that two processes are never in their critical sections at the same time.

The obtained results are presented in Figs. 6–8. Our tool appears to be the most efficient, when the monolithic encoding of the transition relation is combined with the reordering of the variables. Such an approach allowed for a significant memory consumption reduction and improved the verification time. In the case of ψ_2, partitioning of the transition relation proved to be superior to the monolithic encoding, when reordering of the variables was disabled.

8.4. Abstract pipeline system

We consider an abstract system which resembles a pipeline protocol [24]. The aim of the system is to produce the entity r from the entity x. This is performed by n modules which produce intermediate entities needed to produce x. The system consisting of n modules is modelled by the icrrs $\mathcal{I}_p^n = ((S_n, A_n, \mathcal{E}_n), S_0)$, where the background set S_n is defined as:

$$S_n = \{x, s, a_1, \ldots, a_{n+1}, b_1, \ldots, b_n, c_1, \ldots, c_n, d_1, \ldots, d_n, y_1, \ldots, y_n, r\}.$$

To define A_n we first give the following sets of reactions:

$$\begin{aligned} T = {} & \{(\{x\}, \{s\}, \{a_0\})\}, \\ M_i = {} & \{(\{a_i\}, \{s\}, \{y_i\}), (\{y_i\}, \{s\}, \{y_i\}), \\ & (\{a_i\}, \{s\}, \{b_i\}), (\{b_i\}, \{s\}, \{c_i\}), (\{c_i\}, \{s\}, \{d_i\}), \\ & (\{d_i, y_i\}, \{s\}, \{a_{i+1}\})\} \text{ for } i \in \{1, \ldots, n\}, \\ R_n = {} & \{(\{a_{n+1}, y_1, \ldots, y_n\}, \{s\}, \{r\})\}. \end{aligned}$$

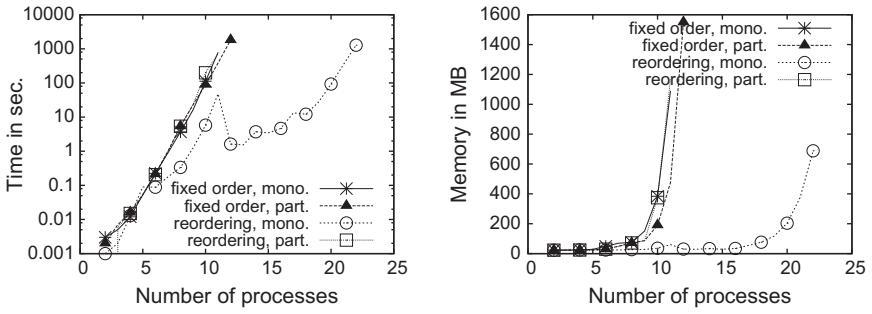

Fig. 6. Mutual exclusion protocol, formula ψ_1.

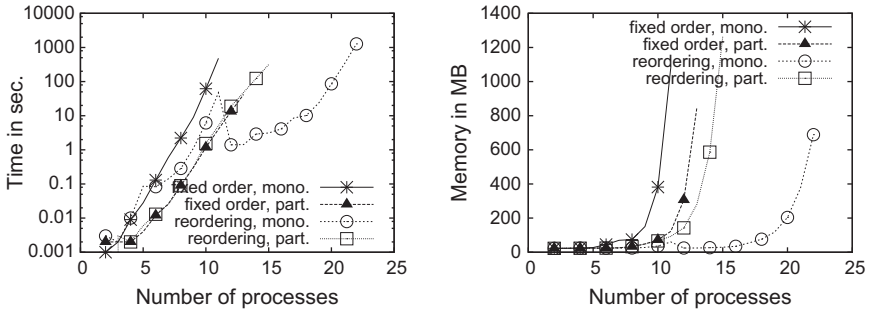

Fig. 7. Mutual exclusion protocol, formula ψ_2.

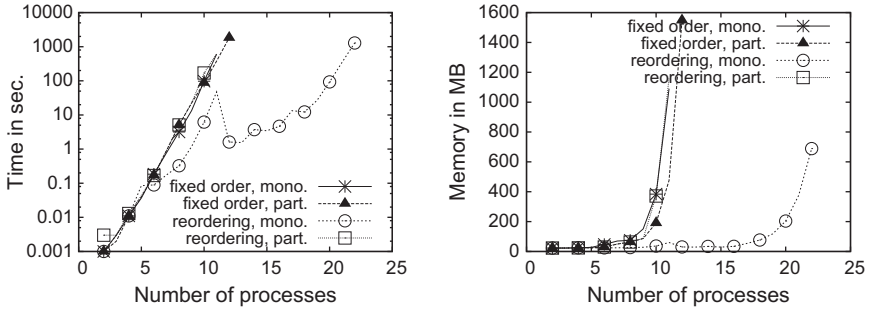

Fig. 8. Mutual exclusion protocol, formula ψ_3.

The reaction of T initialises the process of producing x, the reactions of M_i for $0 < i \leqslant n$ define the reactions of the modules, and the reaction of R_n gathers the products of the modules and produces r which is the final product.

Then, the set of reactions of $\mathcal{I}_p^n$ is defined as $A_n = T \cup \bigcup_{i=1}^{n} M_i \cup R_n$. The initial context set contains only x, i.e., $S_0 = \{\{x\}\}$. We consider two variants of the system, where:

1. $\mathcal{E}_n = \{s\}$, or
2. $\mathcal{E}_n = \{s\} \cup \{a_i | 0 < i \leqslant n$ and i is even$\}$.

For the tests, we have verified the formula $\mathbf{E}_{\{\emptyset\}}\mathbf{F}r$ expressing that it is possible to reach r regardless of the context.

Figs. 9 and 10 present the results, respectively, for the first and the second variant of the system. In the second variant, partitioning of the transition relation resulted in a lower memory consumption when more than nine modules were verified. Verification of the second variant of the system proved to be much more demanding for our tool. When considering the differences between the two variants of the system, we observe that in the second variant there are more reactions enabled at every stage of the state-space exploration of the system due to the fact that some entities are provided earlier by the context sets. This results in the inferior performance.

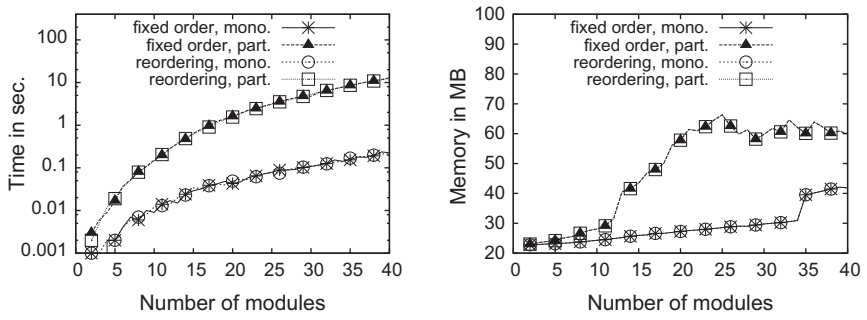

Fig. 9. Abstract pipeline (variant 1).

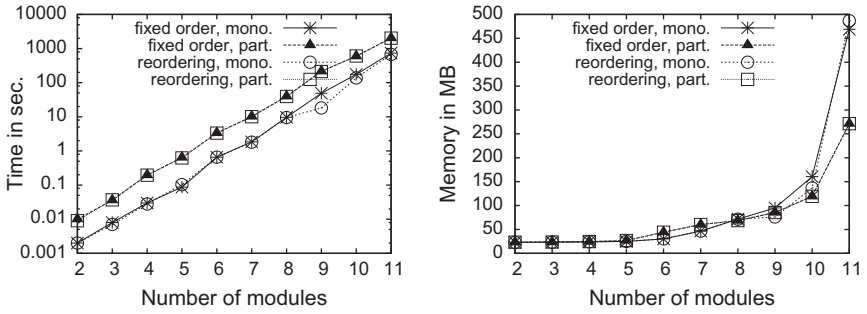

Fig. 10. Abstract pipeline (variant 2).

8.5. Summary

In most cases, the monolithic encoding of the transition relation proved to be the most efficient when combined with the automatic reordering of the BDD variables. Partitioning of the transition relation in some cases may further improve the memory efficiency. However, in most cases the use of partitioned transition relation comes with a time penalty. When dealing with existential formulae, the bounded model checking heuristic can be used to improve the performance by terminating the state-space search as soon as a witness for the verified property is found without computing the entire state-space first.

9. Conclusions

We have introduced a temporal logic for reaction systems (rsCTL) allowing to express properties that depend on the context sequences specified by the context sets in the temporal operators. Our results are specific to reaction systems. The syntax and the semantics of rsCTL have been carefully defined to fit the models of reaction systems, to capture their most important properties, and to allow for intuitive specification. We have also described a symbolic verification method for rsCTL and provided complexity results for the problems of model checking and symbolic model checking for rsCTL. The model checking problem for rsCTL was proved to be PSPACE-complete. The proposed symbolic model checking approach has been implemented. This allowed for an experimental evaluation of our approach which proved, despite the high complexity of the verification problem, that our approach is promising for possible applications of the proposed verification method.

In our future work we are going to provide a logical characterisation of the proposed logic as well as focus on other model checking methods for reaction systems.

References

[1] R. Alur, C. Courcoubetis, D. Dill, Model checking in dense real-time, Inf. Comput. 104 (1) (1993) 2–34.
[2] S. Azimi, C. Gratie, S. Ivanov, L. Manzoni, I. Petre, A.E. Porreca, Complexity of Model Checking for Reaction Systems, Tech. Rep. 1122, TUCS, October 2014.
[3] S. Azimi, C. Gratie, S. Ivanov, I. Petre, Dependency Graphs and Mass Conservation in Reaction Systems, Tech. Rep., TUCS, October 2014.
[4] S. Azimi, B. Iancu, I. Petre, Reaction system models for the heat shock response, Fundam. Inf. 131 (3–4) (2014) 299–312.
[5] R. Brijder, A. Ehrenfeucht, M.G. Main, G. Rozenberg, A tour of reaction systems, Int. J. Found. Comput. Sci. 22 (7) (2011) 1499–1517.
[6] R.E. Bryant, Graph-based algorithms for boolean function manipulation, IEEE Trans. Comput. 35 (8) (1986) 677–691.
[7] J.R. Burch, E.M. Clarke, D.E. Long, Symbolic Model Checking with Partitioned Transition Relations, North-Holland, 1991. pp. 49–58.

[8] G. Cabodi, P. Camurati, S. Quer, Can BDD compete with SAT solvers on bounded model checking? in: Proc. of the 39th Design Automation Conference (DAC'02), 2002, pp. 117–122.
[9] E. Clarke, E.A. Emerson, A.P. Sistla, Automatic verification of finite state concurrent systems using temporal logic specifications: a practical approach, ACM Trans. Program. Lang. Syst. 8 (2) (1986) 244–263.
[10] E. Clarke, O. Grumberg, D. Peled, Model Checking, MIT Press, 1999.
[11] L. Corolli, C. Maj, F. Marini, D. Besozzi, G. Mauri, An excursion in reaction systems: from computer science to biology, Theor. Comput. Sci. 454 (2012) 95–108.
[12] A. Ehrenfeucht, J. Kleijn, M. Koutny, G. Rozenberg, Reaction systems: a natural computing approach to the functioning of living cells, in: H. Zenil, (Ed.), A Computable Universe, Understanding and Exploring Nature as Computation, 2012, pp. 189–208.
[13] A. Ehrenfeucht, G. Rozenberg, Reaction systems, Fundamenta Informaticae 75 (1–4) (2007) 263–280.
[14] A. Ehrenfeucht, G. Rozenberg, Introducing time in reaction systems, Theor. Comput. Sci. 410 (4-5) (2009) 310–322.
[15] E. Formenti, L. Manzoni, A.E. Porreca, Cycles and global attractors of reaction systems, in: Descriptional Complexity of Formal Systems – 16th International Workshop, DCFS 2014, LNCS, 2014, pp. 114–125.
[16] E. Formenti, L. Manzoni, A.E. Porreca, Fixed points and attractors of reaction systems, in: Language, Life, Limits – 10th Conference on Computability in Europe, CiE 2014, LNCS, vol. 8493, Springer, 2014, pp. 194–203.
[17] E. Formenti, L. Manzoni, A.E. Porreca, On the complexity of occurrence and convergence problems in reaction systems, Nat. Comput. (2014) 1–7.
[18] M. Hirvensalo, On probabilistic and quantum reaction systems, Theoret. Comput. Sci. 429 (C) (2012) 134–143.
[19] F. Laroussinie, N. Markey, P. Schnoebelen, Model checking timed automata with one or two clocks, in: CONCUR, 2004, pp. 387–401.
[20] K.L. McMillan, Symbolic Model Checking, Kluwer Academic Publishers, 1993.
[21] A. Meski, W. Penczek, G. Rozenberg, Model Checking Temporal Properties of Reaction Systems, Tech. Rep. 1028, ICS PAS, April 2014.
[22] F. Okubo, S. Kobayashi, T. Yokomori, Reaction automata, Theor. Comput. Sci. 429 (2012) 247–257.
[23] C. Pecheur, F. Raimondi, Symbolic model checking of logics with actions, in: MoChArt, 2006, pp. 113–128.
[24] D. Peled, All from one, one for all: On model checking using representatives, in: Proceedings of the 5th Int. Conf. on Computer Aided Verification (CAV'93), LNCS, vol. 697, Springer-Verlag, 1993, pp. 409–423.
[25] R. Rudell, Dynamic variable ordering for ordered binary decision diagrams, in: Proceedings of the 1993 IEEE/ACM International Conference on Computer-aided Design, ICCAD '93, IEEE Computer Society Press, 1993, pp. 42–47.
[26] A. Salomaa, Functions and sequences generated by reaction systems, Theor. Comput. Sci. 466 (2012) 87–96.
[27] A. Salomaa, On state sequences defined by reaction systems, in: Logic and Program Semantics, 2012, pp. 271–282.
[28] A. Salomaa, Functional constructions between reaction systems and propositional logic, Int. J. Found. Comput. Sci. 24 (1) (2013) 147–160.
[29] A. Salomaa, Minimal and almost minimal reaction systems, Nat. Comput. 12 (3) (2013) 369–376.
[30] F. Somenzi, CUDD: CU Decision Diagram Package – Release 2.5.0. <http://vlsi.colorado.edu/~fabio/CUDD>.
[31] L.J. Stockmeyer, A.R. Meyer, Word problems requiring exponential time: preliminary report, in: Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30 – May 2, 1973, Austin, Texas, ACM, USA, 1973, pp. 1–9.